Preprint Cambridge Centre for Computational Chemical Engineering ISSN 1473 – 4273

Semantic 3D City Agents – an intelligent automation for Dynamic Geospatial Knowledge Graphs

Arkadiusz Chadzynski[2], Shiying Li[4], Ayda Grisiute[4], Feroz Farazi[1], Casper Lindberg[2], Sebastian Mosbach[1], Pieter Herthogs[4], Markus Kraft[1,2,3]

released: November 8, 2021

[1] Department of Chemical Engineering
and Biotechnology
University of Cambridge
Philippa Fawcett Drive
Cambridge, CB3 0AS
United Kingdom

[2] CARES
Cambridge Centre for Advanced
Research and Education in Singapore
1 Create Way
CREATE Tower, #05-05
Singapore, 138602

[3] School of Chemical
and Biomedical Engineering
Nanyang Technological University
62 Nanyang Drive
Singapore, 637459

[4] Singapore-ETH Centre
at CREATE
1 Create Way
CREATE Tower, #06-01
Singapore, 138602

Preprint No. 283

UNIVERSITY OF CAMBRIDGE

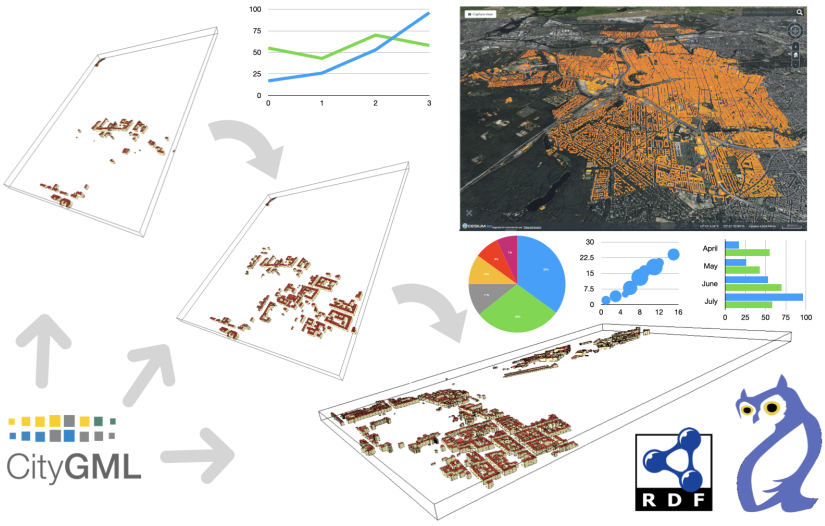
Keywords: Cognitive Architecture, Artificial Intelligence, Knowledge Graph, Automation, City Modelling, Geospatial

Edited by
Computational Modelling Group
Department of Chemical Engineering and Biotechnology
University of Cambridge
Philippa Fawcett Drive
Cambridge, CB3 0AS
United Kingdom

E-Mail: mk306@cam.ac.uk
World Wide Web: https://como.ceb.cam.ac.uk/

Abstract

This paper presents a system of autonomous intelligent software agents, based on a cognitive architecture, capable of automated instantiation, visualisation and analysis of multifaceted City Information Models in dynamic geospatial knowledge graphs. Design of JPS Agent Framework and Routed Knowledge Graph Access components was required in order to provide backbone infrastructure for an intelligent agent system as well as technology-agnostic knowledge graph access enabling automation of multi-domain data interoperability. Development of CityImportAgent, CityExportAgent and DistanceAgent showcased intelligent automation capabilities of the Cities Knowledge Graph. The agents successfully created a semantic model of Berlin in LOD 2, compliant with CityGML 2.0 standard and consisting of 419 909 661 triples described using OntoCityGML. The system of agents also visualised and analysed the model by autonomously tracking interactions with a web interface as well as enriched the model by adding new information to the knowledge graph. This way it was possible to design a geospatial information system able to meet demands imposed by Industry 4.0 and link it with the other multi-domain knowledge representations of The World Avatar.



Highlights

- Industry 4.0 capabilities provided by means of sustainable digitisation practices.
- Cognitive Architecture of JPS Agent Framework and Routed Knowledge Graph Access components.
- Automated City Information Model instantiation, visualisation and analysis by Semantic 3D City Agents.

Contents

1 Introduction

General context and problem space

Present needs for sustainable digitisation [53] of geospatial features [11] at the country level are already recognised by Australia, Austria, Belgium, Canada, Estonia, Finland, France, Germany, Ireland, Japan, Luxembourg, Netherlands, UK, USA, Poland, Singapore, Switzerland [54], Turkey [5], Taiwan [12, 37], and Oman [2], amongst others. It has also been realised that standards-based 3D city models, apart from aiding more traditional urban planning efforts [52], could play an important role in multi factor optimisation scenarios [44]. Such complex simulations on interoperable data, spanning multiple domains of interest, could be targeted to get closer to answering the set of problems regarded by the United Nations as 'the biggest threat ever faced by modern humanity as a whole' [51]. Namely, the climate change, which could act as a *crisis multiplier* and, to countries like Singapore, even poses an *existential threat* [36].

Cities Knowledge Graph (CKG) [13–15] is envisioned as a subsystem of The World Avatar (TWA) - a general, all-encompassing dynamic knowledge graph, built in accordance with semantic web standards and recommendations provided by the W3C, and capable of multi-domain knowledge representation [1, 16–18, 20, 21, 32, 58, 59], which is being developed as a collaborative research effort between Cambridge Centre for Advanced Research and Education in Singapore (CARES) and Singepore ETH Centre (SEC). As a dynamic geospatial knowledge graph, based on the Semantic 3D City Database [11], CKG is designed to produce and process multi dimensional representations of urban environments. Knowledge graph architecture allows to easily combine them with the J-Park Simulator (JPS) [29, 39, 40, 56, 57, 60] - an agent-based subsystem, capable of simulating emissions dispersion from various types of air pollution sources as well as optimising designs of Eco-Industrial Parks (EIPs) with respect to their carbon footprint, within TWA. The Parallel World Framework [19], capable of complex scenario analysis and adding time varying aspects to knowledge graphs, could be utilised as a basis for the above mentioned multi factor optimisation scenarios that also take into account other domain representations and further enrich insight scope.

Unlike endeavours such as TWA and CKG, presented in this paper, currently available information systems that do not implement at least some of the elements of cognitive architecture [33] encounter a number of problems due to their human-computer interaction orientated design that are typical to the pre-fourth industrial revolution computer systems architectures. Contrary to that, CKG would be able to close some of the gaps, commonly found in today's Geographic Information System (GIS) applications and infrastructure that are more elaborated on in the next few paragraphs.

GIS and Industry 4.0

Smart Cities that make use of as well as provide interoperable sources of various types of multi-domain data are one of the flagship Industry 4.0 applications [35]. While many of the GIS available at the moment serve as means for creating digital twins of geospatialy describable elements of cities, they generally lack of automation that is regarded as a second most important characteristics of the information systems that are capable of meeting demands imposed by the fourth industrial revolution [34]. Because of that, they very

often do no longer meet modern needs for intelligent systems, capable of autonomously analysing information and taking into account multiple domains [46]. Existing software relies on human-computer interaction and very often there is more human than computer involved in performing analytical tasks.

Creating and updating City Information Models (CIM [22]) via existing techniques relies on time consuming and error prone manual data curating processes. Legacy GIS also lack dynamics, as existing data formats and modelling techniques make models hard to keep up to date. Such systems were designed to work with data spread over multitude of various flat files. There is no easy way to learn lessons from such static models that remove historical aspect and, because of that, do not allow to get insights about evolution, stagnation or deterioration of cities. They commonly also do not allow to analyse CIM changes without a complex process of importing and exporting multiple types of files for an entire city.

Examples of such CIMs are XML file based models describing various urban elements in the *CityGML* standard, provided by the Open Geospatial Consortium (OGC) [23]. They are commonly used as a data exchange standard for city landscape management and planning systems or even as a file-based data source for applications visualising 3D city landscapes on the web. *3D City Database* was developed at the Technische Universität München (TUM) with intention of adding flexibility and scalability to the CityGML based models by transforming XML into Relational Database Management System (RDBMS) [55]. Better data interoperability is supported by implementation of domain specific extensions as well [50]. However, data transformation processes for CIM creation and visualisation utilise its Importer/Exporter tool application, making them manual and error prone, especially when larger models are taken into consideration.

Synthesis

The *Semantic 3D City Database* [11], based on a semantic triple store back-end instead of RDBMS, enabled dynamic geospatial knowledge graph capabilities in TWA. This innovation removed data interoperability limits of the original 3D City Database imposed by the default Closed World Assumption (CWA) in relational databases [49]. The Semantic 3D City Database's adoption of the Open World Assumption (OWA) implies that it can operate as a semantic knowledgebase, enabling reasoning and truth maintenance capabilities via inference engines, together with OntoCityGML as its schema. Secondly, it also added geospatial search features that allow to efficiently retrieve CIM data from specific regions bounded by a set of coordinates [7]. However, in the proof of concept described by Chadzynski et al. [11], data import as well as export still relied on the appropriately augmented Importer/Exporter tool, and remained manual. While the manual approach of the proof of concept stage allowed to successfully produce a semantic twin of the Charlottenburg-Wilmersdorf district of Berlin based on a CityGML 2.0 LOD2 model, the lack of automation of that approach became an issue while attempting to instantiate the remaining eleven districts of Berlin in the knowledge graph, and link the instantiated city data into other knowledge domains.

The purpose of this paper is to present elements of a cognitive architecture applied in CKG that enable automated data processing tasks and automated analytical capabilities. These functionalities are demonstrated by a set of software agents that automatically load

and produce a semantic representation of the entire city of Berlin (all 12 districts) in TWA, automatically visualise it using an adapted version of the 3D City Database's web map client, as well as automatically calculate and visualise various distances of interest between particular city objects by tracking external interactions with the representation. The most general elements of the architecture are discussed in section 2 describing the JPS agent framework (section 2.1) and Routed Knowledge Graph Access (section 2.2) in more detail. Concrete implementations of intelligent autonomous agents based on those elements are presented in section 3, which discusses and demonstrates an automated CIM creation workflow using the City Import Agent (section 3.1), dynamic visualisation capabilities in the CKG facilitated by the City Export Agent (section 3.2), and automated distance-related analytical capabilities, deriving new knowledge based on external interactions using the Distance Agent (section 3.3). Empirical evaluation of the Berlin CIM autonomously instantiated, visualised and analysed by the Semantic 3D City Agents in the CKG as well as potential research direction are described in the last section 4.

2 Cognitive Architecture in The World Avatar

Information system architecture of TWA is designed to fulfill the needs of a general knowledge graph operated on by a system of intelligent autonomous agents. Such agents typically consist of methods allowing them to intercept some inputs (*sensors*), *programs* that perform information processing based on those inputs, as well as previously acquired knowledge representing the external environment, and methods resulting in certain actions based on the outcomes of that information processing (*actuators*) [48]. According to Langley et al. [33], a cognitive architecture includes aspects of a cognitive agent that are constant over time and across different application domains. These typically include:

- the short-term and-long term memories that store content about the agent's beliefs, goals and knowledge;
- the representation of elements that are contained in these memories and their organisation into larger-scale mental structures;
- the functional processes that operate on these structures.

The knowledge graph of TWA provides an infrastructure for intelligent autonomous agents to store and retrieve memories built on representations of elements from multiple domains. All knowledge is stored in a form of semantic triples, in accordance with semantic web standards and recommendations by W3C. The JPS Agent Framework, together with Routed Knowledge Graph Access components, provides interfaces for such agents to intercept inputs and efficiently operate on a highly distributed multi-domain knowledge representations, enabling them to perform certain actions that either augment previously acquired knowledge or have an effect on the external environment. Therefore, such a combination of a knowledge graph and an agent system forms a set of basic architectural building blocks allowing TWA to replicate and automate, at least some of, the fundamental behaviours and functions found in other cognitive systems, including human-like intelligence [31].

2.1 JPS Agent Framework

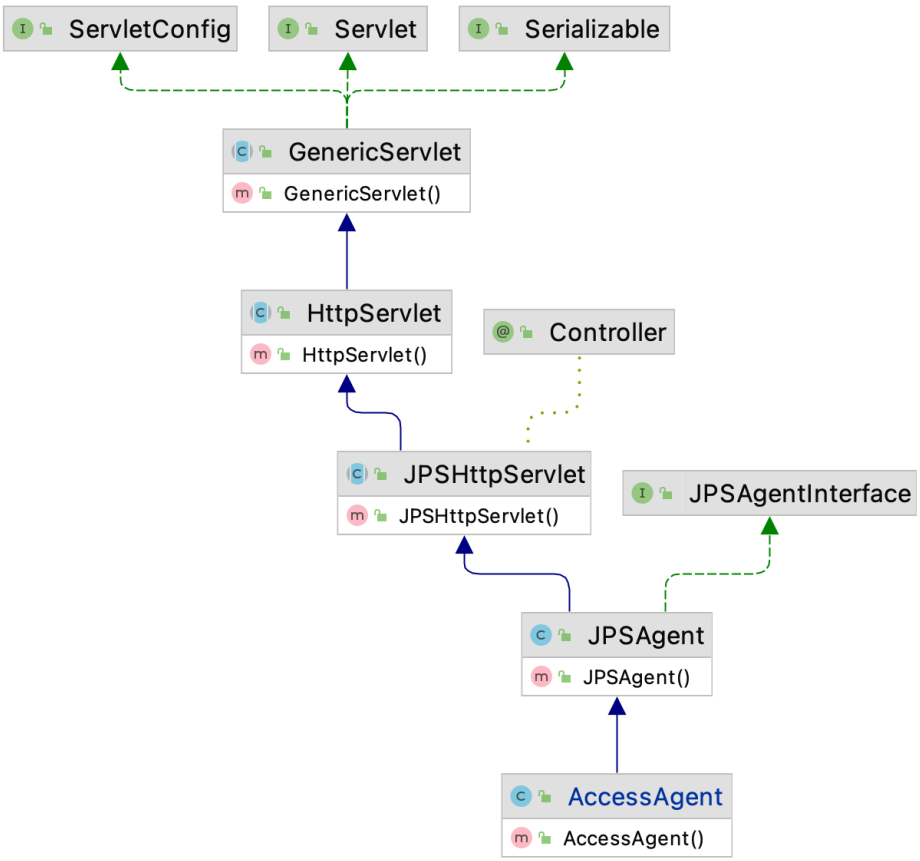


Figure 1: *A class diagram of fundamental building blocks of the cognitive architecture in TWA. An abstract JPSAgent, based on Java HTTP Servlet technology, is a base class for all the other agent classes. This includes AccessAgent facilitating data store and retrieval functionality for the knowledge graph. This design ensures maximum backwards compatibility with all the other JPS and TWA infractructure as well as offers flexibility to integrate with plethora of existing web APIs working over HTTP protocol.*

The *JPS Agent Framework* is a result of generalisations of an agent system actively worked on within The World Avatar (TWA) project under its J-Park Simulator (JPS) since 2014 [39]. It is a part of *JPSBaseLib* library, developed in the Java programming language and widely used within TWA as a whole [28]. The agents as well as the library have been designed around Microservices Architecture [41, 42] while following the Minimal Service Model [30, 43]. This approach to modularity ensures scalability of a highly distributed system and improves change management. It also enables sustainable service evolution [42] by extending system components to be reusable and replaceable at the same time. Those architectural properties of the agent system in TWA make automated service integration and composition [59] default for heterogeneous linked data interoperability. Semantically encoded data corresponds to the elements contained in knowledge represen-

tations of multiple domains and is worked on by its agents regarded as linked semantic web services [43].

As depicted in Fig. 1, JPSAgents in TWA inherit from standard Java HttpServlet class [26]. This makes HTTP a default inter-agent communication protocol that allows for synchronous and asynchronous information exchange using common serialisation formats, like XML or JSON, on the same machine, local network, or over the Internet. Such scalability makes it easier to implement TWA agent system in a highly distributed form as well as leaves it open to a smooth integration with a plethora of external web services available on the web. The JPS Agent Framework makes agent programs independent of the inter-agent communication protocol by extracting HTTP request parameters into a JSON format that is passed to the agent as the input. In other words, agents built on the framework intercept symbolic inputs [33] via their virtual sensors. Following such separation of concerns during design makes agents' programs relatively easily portable to frameworks that use other protocols for messaging. Agents that work with physical sensor devices, measuring parameters such as temperature, humidity or the amount of light, receive symbolic inputs in the same manner by intercepting requests coming from appropriate middleware.

The framework's *JPS Agent Interface* requires all agents to perform syntactic input validation in the next step after receiving their inputs. Accordingly to the cybersecurity best practices and recommendations, this step is performed to ensure that only properly formed data is entering the workflow in an information system, preventing malformed data from persisting in the knowledge graph and triggering malfunction of various downstream components [38]. This way, inputs such as the sentence "All swans are white.", the number 11235813, or the IRI "http://www.some.com/input/" are accordingly evaluated as a valid string, a valid integer, and a valid IRI, regardless of whether the sentence is true, the number represents the beginning of the Fibonacci Sequence and the IRI is actually resolvable to an existing resource, etc. The agents' programs, in turn, include various functional processes that operate on representations of elements from multiple domains and perform semantic validation of the symbolic inputs by comparing them with existing knowledge and the agent system's beliefs stored in the knowledge graph. An agent could thus respectively check the knowledge graph for an existing instance of an animal that is a bird with black feathers belonging to the family Anatidae within the genus Cygnus, an existing mechanism that allows to detect mathematical patterns in sequences of numbers, and an existence of the knowledge routing graph entry matching a certain resource, in order to semantically validate its inputs before performing any further information processing or including them in the knowledgebase.

The JPS Agent Framework allows for flexibility in developing agents' programs and does not require further conformance to any specific interfaces. This way, agents process their inputs by comparing them against existing memories and knowledge, transform these inputs via code included in various libraries, execute any specialised software required to perform more sophisticated operations, and contact other agents as well as existing web APIs via the inter-agent communication protocol. Agents' virtual actuators either contact other agents, external APIs or physical actuators' middleware concurrently with augmenting existing knowledge, or add new elements to the knowledge graph in separate sets of tasks. Existing memories are retrieved by agents for such processing purposes, as well

as stored in the knowledge graph by the means of technology agnostic routed knowledge graph access mechanisms, described in the next subsection. Appropriate methods to do so are included in the interface that all the JPS Agents conform to by default. The knowledge graph made available via this mechanism provides agents with messaging context that is needed in order to guide the semantics so that congruence about the semantics can be achieved between the sender and receiver and to enable an agent to orient the semantics to a specific application or circumstance [45].

2.2 Routed Knowledge Graph Access

The routed knowledge graph access is a set of TWA components that allow agents to navigate through resources in its distributed multidomain knowledge graph. It consists of the *StoreRouter* and *StoreClients* that provide a storage technology agnostic way of information management to the other agents. Those components are integrated into the *AccessAgent* that acts as a knowledge graph entry point to them. They contact it whenever the representations of certain elements are needed for their own purposes and are not concerned with the storage technologies used to persist those representations within the knowledge graph.

The *StoreRouter* in tandem with the *StoreClients* are able to locate a data store by a resource identifier as well as retrieve, insert, update or delete corresponding representations upon an agent's request, regardless of whether the representations are recorded in a form of a flat file, a relational database table or a graph in a semantic triple store. It makes JPS Agents' designs independent of the underlying TWA knowledge graph storage layer and allows for the parallel evolution of those layers within the system as a whole. It also automates inter-agent data interoperability by making representations of multiple domains accessible via an uniform interface to the all-encompassing TWA knowledge graph, when looked at from the point of view of all agents.

2.2.1 Knowledge Graph Routing via OntoKGRouter

To enable agents to operate on classes, properties, instances and data collated from multiple domains and represented in TWA that is distributed over several servers with the possibility of migrating to newly setup servers due to ever-increasing demand of improved performance and storage capacity, a server agnostic approach is required in accessing the knowledge-graph. To this end, a *StoreRouter* is built with the instantiation of an ontology called OntoKGRouter developed for describing routing information in the form of triples in a routing table consisting of subject, predicate and object columns, where the subject refers to the relative IRI of a domain, the object refers to the absolute IRI, and the predicate links the subject and object.

As depicted in Fig. 2, when the *StoreRouter* receives a request from an agent with the relative IRI of a triple store or the absolute IRI of an RDF/OWL file to establish access to the domain of interest, it validates the request and detects the corresponding store type of domain. Any request targeting a triple store or an RDF/OWL file invokes the query builder to form a query to retrieve the available, absolute IRI of the triple store endpoint

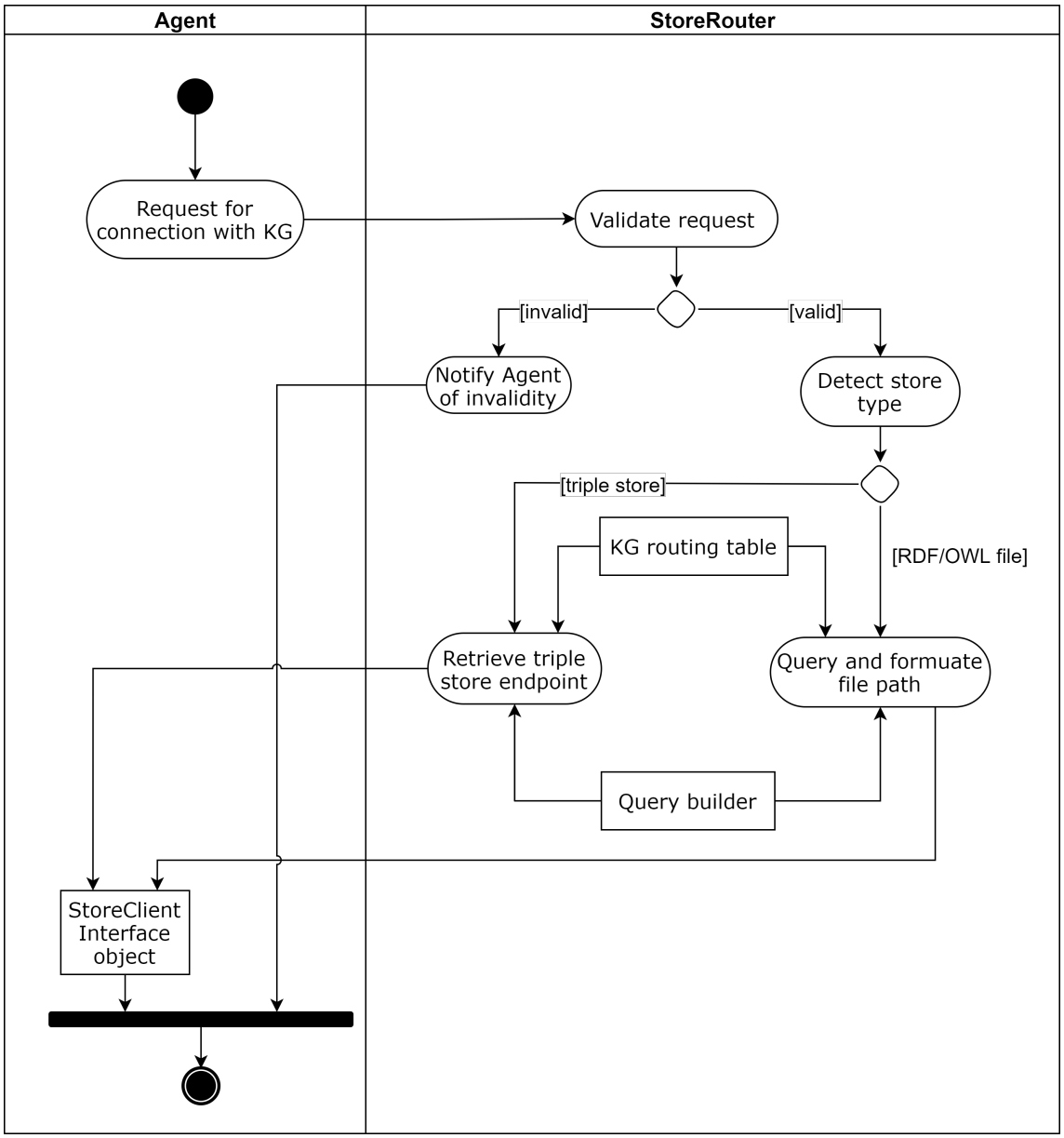


Figure 2: *An activity diagram of the knowledge-graph router built with the instantiation of an OntoKGRouter ontology developed for describing routing information in the form of semantic triples. The routing table consists of subject, predicate and object columns, where the subject refers to the relative IRI of a domain, the object refers to the absolute IRI, and the predicate links the subject and object. StoreRouter that receives a request from an agent with the relative IRI of a store validates the request and detects the corresponding store type for a domain.*

or base IRI of the file store from the KG routing table. By combining the base IRI and the absolute IRI of a file, the absolute file path is formulated, which is indispensable to execute update operations on a file.

Finally, the *StoreRouter* instantiates a *StoreClient* object of the *StoreClientInterface* type and returns it to the requesting agent for querying or updating the target resource within the knowledge graph. An agent can issue multiple requests to set up combined access to different domains stored either in triple stores or files or both.

2.2.2 Technology-Agnostic Knowledge Graph Access via Store Clients

Storage technology agnostic access to the knowledge graph is provided by the *RemoteStoreClient* and *FileBasedStoreClient*. Both *StoreClients* implement the *StoreClientInterface*, which defines abstract methods to retrieve, insert, SPARQL query and SPARQL update data stored in the knowledge graph. Abstraction of the clients through the *StoreClientInterface* allows to decouple the knowledge storage technology from JPS Agents, which are concerned only with representations of respective domains, regardless of the technology. This design allows JPS Agents to access knowledge about multiple domains, for which different storage technologies are most suitable and make it interoperable via uniform interface. All JPS Agents utilise this design to retrieve and store required representations from multiple namespaces of the CKG, which separate information about built environments described in separate coordinate systems.

The *RemoteStoreClient*, presented in Fig. 3, is agnostic to the technology behind the remote SPARQL endpoint. It uses Jena JDBC SPARQL over the JDBC framework that is also utilised by the CityImportAgent and CityExportAgent, making the knowledge storage and retrieval interfaces uniform across all the JPS Agents. The *FileBasedStoreClient* allows access to RDF data stored in a flat file format using Jena RDFConnection, which provides a unified set of operations for working on RDF data loaded from file-based storage into a Jena dataset. Instantiation of a *RemoteStoreClient* or *FileBasedStoreClient* is handled by the *StoreRouter*.

3 Semantic 3D City Agents

Semantic 3D City Agents that are part of the CKG, a knowledge graph dedicated to representations of built environments in the TWA, are based on the components and principles of the cognitive architecture described in the previous section. They demonstrate application of the architecture and potential for intelligent automation applied to city modelling:

- Import process from CityGML 2.0 into CKG model does not require a person operating desktop software anymore. Instead, *CityImportAgent* performs data validation and city objects instantiation in the knowledge graph as soon as it finds a .gml file in a hard drive directory which it was instructed to watch for the appearance of such files
- KML export process for visualisation also does not require anyone operating export

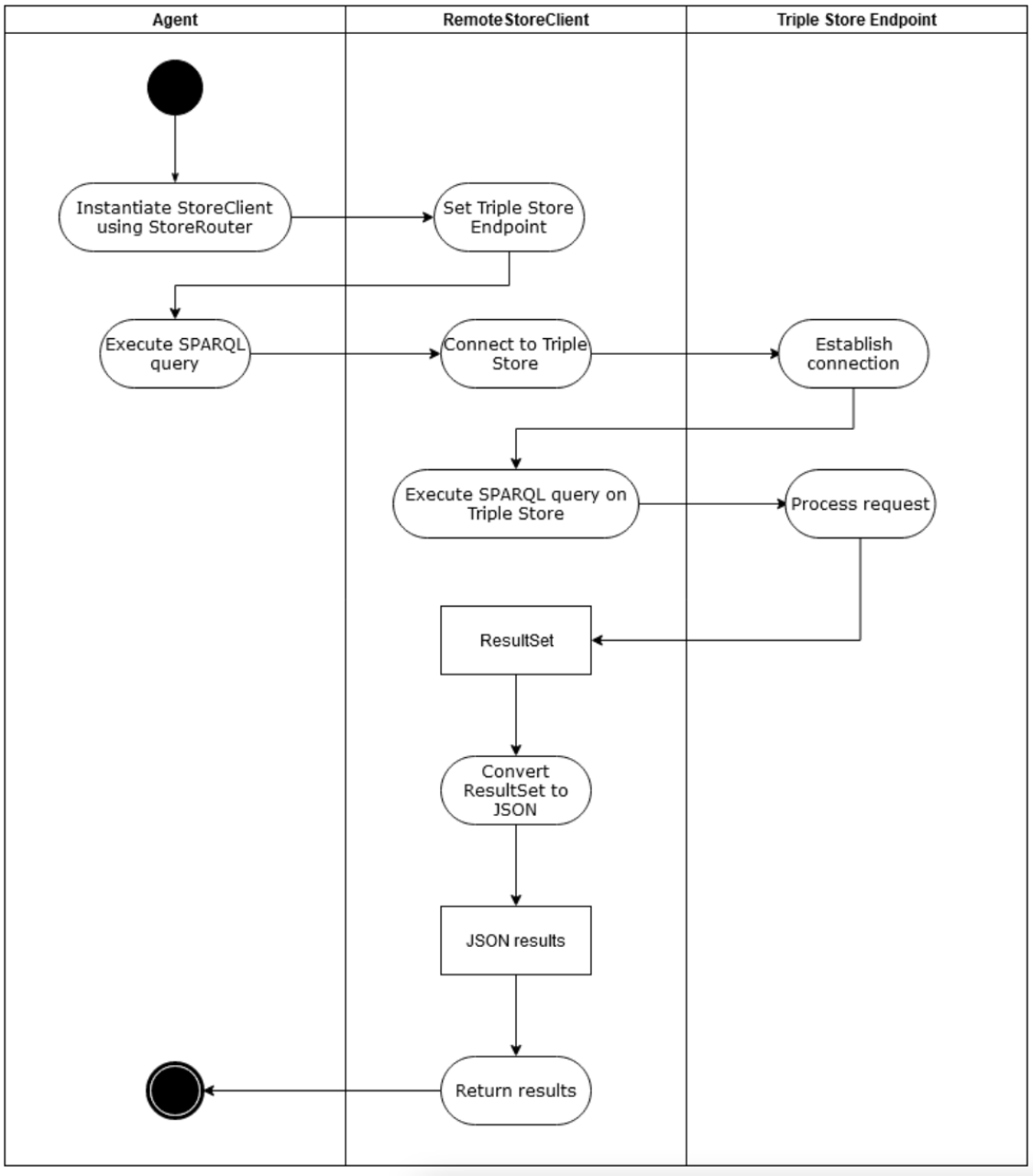


Figure 3: *An activity diagram of an agent using a RemoteStoreClient to perform a SPARQL query on a remote triple store. The RemoteStoreClient, an instance of the StoreClientInterface type, is agnostic to the technology behind the SPARQL endpoint. Instantiation of the RemoteStoreClient is performed by the StoreRouter.*

application manually. Instead, *CityExportAgent* creates visualisation data either for the whole model or a part of it automatically upon receiving a request to do so. It is also capable of updating KML for individual city objects, which adds truly dynamic visualisation capabilities to the city modelling software.

- *DistanceAgent* demonstrates potential for intelligent analytical capabilities of city modelling software. It autonomously calculates distances of city objects, which were interacted with on the web map client and displays this information whenever it is ready. There is no manual process involved in the analytics, which occurs independently of the interaction process. It is possible to extend this design pattern to support more advanced analytical features.

3.1 City Import Agent

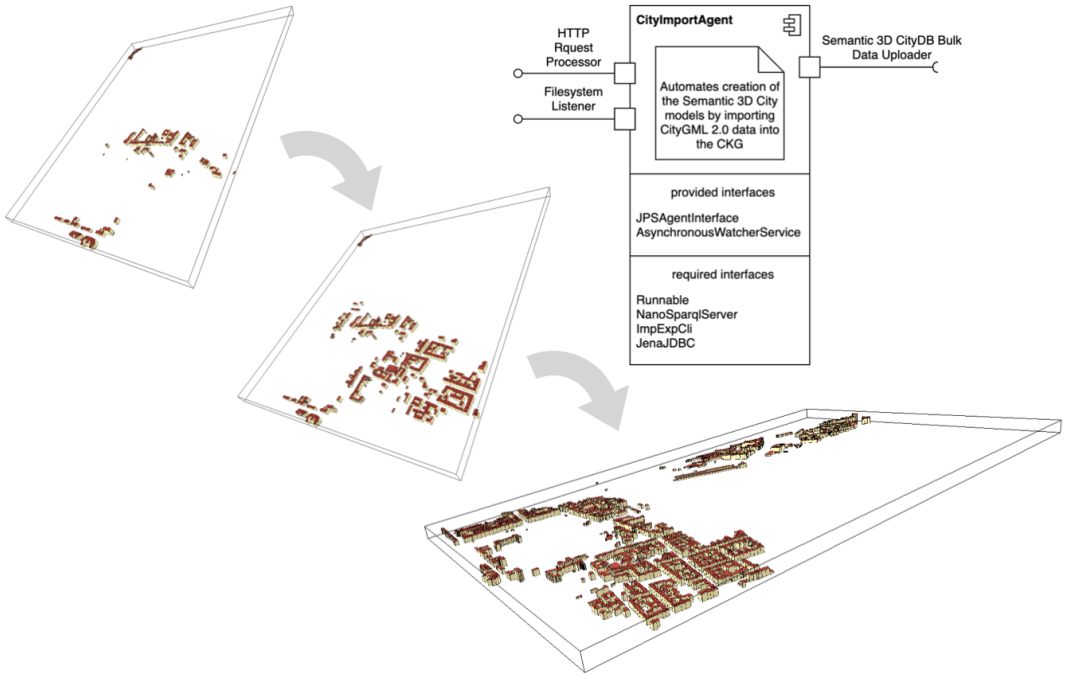


Figure 4: *CityImportAgent automates instantiation of city models in TWA, by listening on two IRIs. Upon receiving a request on the Listen IRI, it calls JPS Asynchronous Watcher Service and instructs it to watch for appearance of new GML files in a directory specified by the request, in a separate thread. Upon receiving a request on the Action IRI, it splits the file to smaller and more manageable chunks before importing each chunk using four tasks running in separate threads: BlazegraphServerTask, ImporterTask, NquadsExporterTask and NquadsUploaderTask. This way it incrementally creates semantic representations for the whole cities.*

The *CityImportAgent* automates creation of semantic representations of cities that are used by all the other CKG agents. As depicted in Fig. 4, it instantiates elements of cities

in a form of Semantic 3D City database [11] using OntoCityGML as a schema for linked data structure spread across multiple named graphs [10] that store CityGML 2.0 features in different levels of detail. The agent operates in two modes, depending on which of the two IRIs it intercepts the request at. Upon receiving a request on the Listen IRI it gets activated in the filesystem observation mode by the means of the *Asynchronous Watcher Service* code capable to watch for appearance of new GML files in a directory specified by the request, in a separate thread. Upon receiving a request on the Action IRI, it creates instances of the city model elements, found in a GML file using the listening mode, by the means of augmented 3D City Database Importer/Exporter tool [11], originally developed at TUM. Prior to that the agent splits the file to smaller and more manageable chunks before importing each chunk using four tasks running in separate threads. Detailed activity diagram is available in appendix A.

JPS *Asynchronous Watcher Service* [4] is a self contained Java library that could be either imported by other agents and called by their programs directly or run as a standalone web service on an embedded Apache Tomcat [3] server. The service is designed to listen to watch requests pointing it into specific filesystem directories and types of files. It also requires callback URL parameter in order to work in an asynchronous manner. Upon input validation the service immediately responds to a calling agent with either information about bad request, in case of invalid input, or creation of file watcher Java object observing a directory specified in the request for appearance of new files of certain type or modifications to the existing files. This way the calling agent does not have to wait until such file-system event occurs but it could be performing other tasks while waiting to be contacted on the callback URL or simply stay dormant until it receives a request on that URL. The *CityImportAgent* makes use of the service as a library called directly from his program while operating in the listen mode. The agent creates filesystem watcher objects by the means of this code and instructs them to call itself back on the Action IRI whenever an event of new GML file appearing in an import directory occurs. Although it makes the import agent asynchronously self referential, any of the watcher tasks terminates as soon as such filesystem event occurs and just after issuing successful callback request. This avoids infinite self reference loops.

Interception of the callback request on the Action IRI triggers the *CityImportAgent* to attempt automated data validation and processing of the GML file in order to instantiate a new CIM in the Cities Knowledge Graph. The agent uses four tasks running in a separate threads to accomplish this goal. It also uses two *BlockingQueues* form Java Collections Framework [24] in order to facilitate concurrent information exchange between the tasks. They inform each other about completion of a particular piece of work on a given data chunk and signal start and termination of a task dedicated to that chunk so it could be picked up by other tasks before it is finally sent over to TWA knowledge graph in a semantic form. All tasks implement Java *Runnable* [25] interface extended by adding methods that allow to detect whether the task is currently running as well as to stop it.

BlazegraphServerTask creates local instances of the *NanoSparqlServer* and puts them on a *BlockingQueue* to be picked up by the *ImporterTask*. At first it creates Blazegraph configuration file with a name corresponding to a given data chunk. Next it copies the file from a template as well as updates its content with the Blazegraph journal name containing data chunk sequence number. It also sets up all the required system properties and server

properties with appropriate configuration file, created at the first step, before starting *StandaloneNanoSparqlServer* [8]. It places the server at the queue and keeps observing the server for its termination until it stops itself as well when such an event is detected.

ImporterTask at first keeps observing the queue for appearance of *NanoSparqlServer* instances. Whenever a new instance appears in the queue, the task takes it out of there and sets up 3D City Database Importer/Exporter tool's project.xml file by updating server details and import namespace variables accordingly to the given data chunk number. After that it calls the importer's main entry point method with chunk and project configuration file paths for a given chunk as arguments. When the import process is finished it stops the server, which also signals the previous task to terminate. Then the task creates new empty *n*-quads file indicating that the data imported into the Blazegraph journal could be exported before being sent over to the remote location. After the server is stopped and the file is created the task terminates itself.

NquadsExporterTask uses the *ExportKB* Blazegraph [6] code to create *n*-quads file containing data transformed by the importer to the semantic form. Local IRIs are replaced with TWA IRIs at this point. The server, host and port information for IRIs is taken from the project file used for importing the given chunk with the same sequence number as the *n*-quads file prepared by the task. After the IRI replacement the task removes all helper files generated by the previous tasks processing the given data chunk and puts the file on the queue before terminating itself.

NquadsUploaderTask reads the updated *n*-quads file that it finds in the queue and uploads it to the *BulkDataLoad* endpoint of TWA. When the upload request finishes the task terminates itself. In case of a given *n*-quads file corresponds to the last chunk of the data to be imported, it creates a timestamped audit trail archive for the whole dataset import process.

This work shows potential of a complex task automation by an agent based on the JPS Agent Framework that results in creation of multifaceted CIMs in the CKG. The *CityImportAgent* was used to transform 14.9GB Berlin LOD2 building data from GML into semantic form uploaded into TWA without anyone manually operating the original TUM tool. The whole semantic representation contains 419 909 661 **S**ubject-**P**redicate-**O**bject triples in total, partitioned into named graphs that separate from and link to each other:

- 3 475 683 City Objects
- 15 258 678 Generic Attributes of City Objects
- 539 274 Buildings
- 587 109 Addresses
- 540 660 External References
- 2 936 408 Thematic Surfaces
- 9 558 218 Surface Geometries

As it is demonstrated in the next two subsections, such large-scale mental structures encoded in a semantic form provide knowledgebase open to analysis and augmentation as well as linking to and comparing with other domain representations by all the other TWA agents. In terms of cognitive architecture, the agent demonstrates following capabilities: recognition, decision making, choice, monitoring, execution, action, interaction, communication as well as acquisition, representation, refinement and organisation of knowledge. When evaluated, it also proves to be: efficient, scalable, reactive, persistent, autonomous and improvable [33].

3.2 City Export Agent

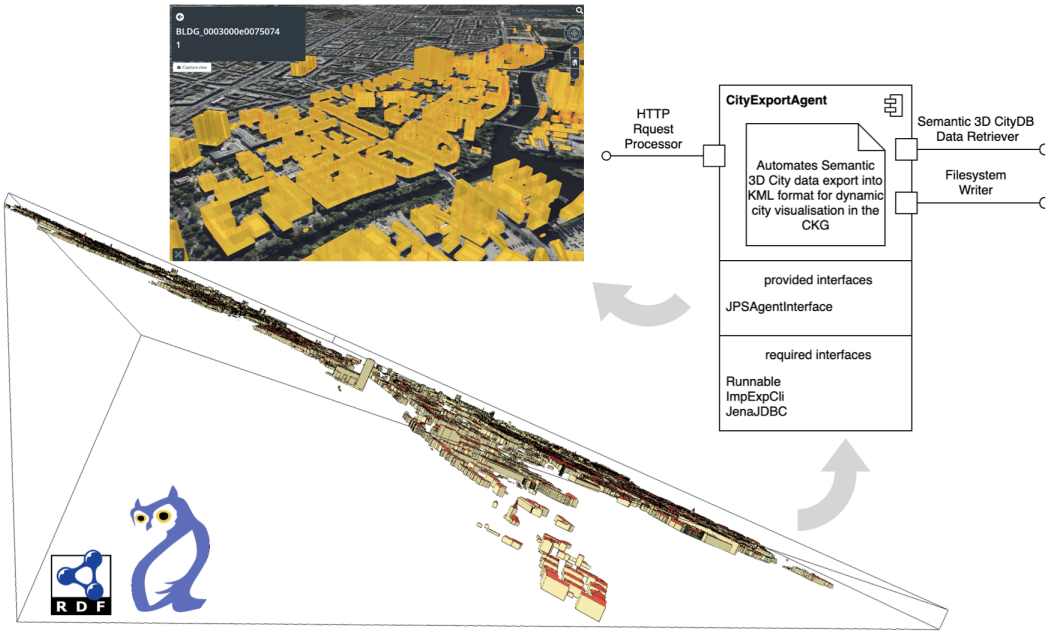


Figure 5: *The City Export Agent automates the export of the city model data needed for visualisation. The data could be exported for the whole model, different areas found via geospatial search, as well as individual city object members. The original 3DCityDB Importer/Exporter SQL queries are translated to SPARQL using the SQL2SPARQL transformer when the agent is connected to the Semantic 3D City Database. After the execution of the queries and the post-processing of returned results by the GeoSpatialProcessor, a KML file is generated for visualisation of the city model.*

The *CityExportAgent* exports the semantically encoded geospatial representation of city objects stored in the Semantic 3D City Database of CKG for CIM visualisation and external interactions in an automated manner. This process is depicted in Fig. 5. The data could be exported for the whole model, different areas found via geospatial search, as well as individual city object members stored in the the knowledge graph. Similar to the *CityImportAgent*, it also makes use of the augmented TUM Importer/Exporter tool (ImpExp), that offers a Command Line Interface (CLI) to facilitate automatic execution of various

tasks via commands and parameters.

The *CityExportAgent* starts operating when a request is intercepted at a pre-defined IRI. The export operation performed by the ImpExp requires certain program arguments as inputs, such as the gml IDs of the city objects to be exported, the output destination of the generated KML file, and the location of configuration file for the tool. When an export activity is required, a HTTP POST request with the required parameters, presented in a JSON format, is sent to the *CityExportAgent*. After a successful validation, the parameters are added to the configuration file, which also contains the database connection information for the export activity. As the agent is designed to operate in a automated manner, the output destination of the KML file and the path to the configuration file are considered predefined parameters and read from a Java property file.

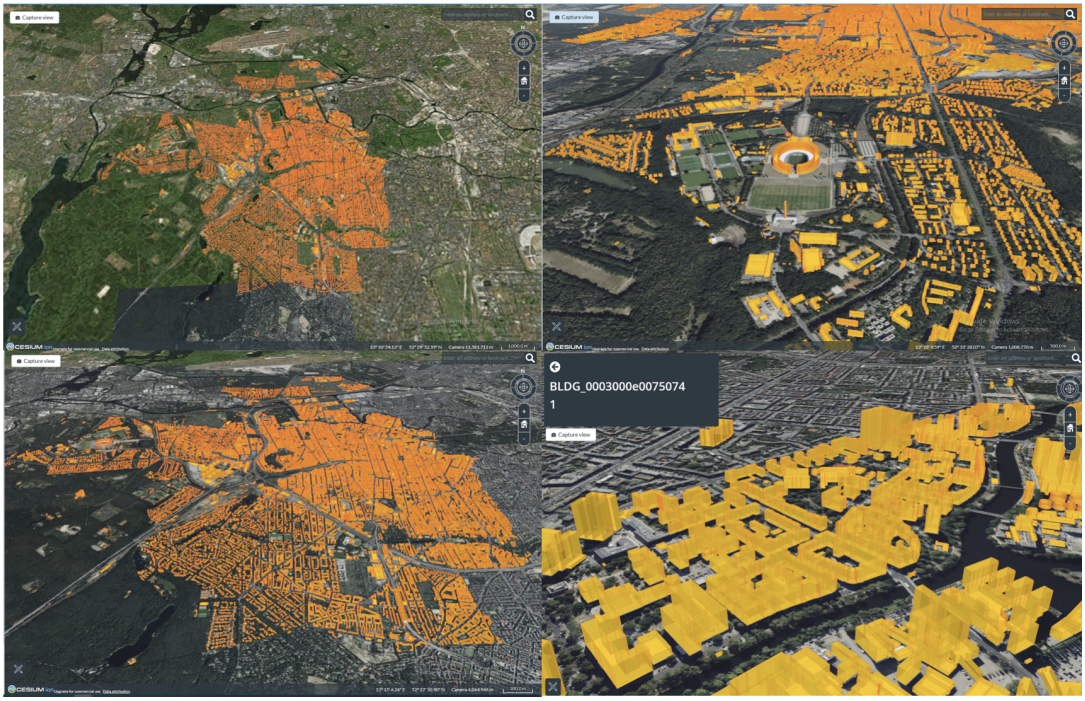


Figure 6: *Different views of a Charlottenburg-Wilmersdorf dataset visualised using KML files exported by the CityExportAgent from the Semantic 3D City Database of the Cities Knowledge Graph. The KML files contain the city model in LOD2 in extruded footprint display form.*

Once all the program parameters are in place, the execution of the exporter task is triggered by the agent that chooses one of the three options: export a single object, export multiple objects, and export the whole database (denoted with "*" as input). Each input variant leads to different combinations of queries used to retrieve the geometry information of the city objects in the database via SPARQL.

The original TUM Importer/Exporter tool is designed to work with relational databases like PostGIS and Oracle using SQL queries. In this augmented version of the tool, the KML export process (KmlExporter) has been extended to export the city model by executing SPARQL statements using the OntoCityGML schema against the Semantic 3D City Database of the CKG [11]. The connectivity to the respective database has been

implemented using Jena JDBC, a SPARQL-over-JDBC driver framework [27]. In order to maximise the re-usability of the existing code and preserve the initial functionalities, two main components are implemented for the export operation against semantic graph databases: the *SQL2SPARQL Transformer* and the *GeoSpatial Processor*.

The *SQL2SPARQL Transformer* has been implemented to translate SQL statements to equivalent SPARQL statements that use the OntoCityGML schema when the tool is connected to the semantic graph database. The second component, the *GeoSpatial Processor*, has been introduced specifically to enable the system to provide the same information when generating KML files as PostGIS provides when making use of built-in functions. Particular SQL statements make use of built-in geospatial functions provided by PostGIS and Oracle databases, but these are not present in the current version of Blazegraph, the graph database used as the backend for the Semantic 3D City Database of the CKG. For example, geospatial funtions like ST_Transform, ST_Area, ST_IsValid are embedded in original SQL statements and evaluated directly by the database engine. Such functions are predominantly used to filter intermediate query results. The *GeoSpatial Processor* has been implemented to provide similar geospatial functionalities to post-process the query results. By means of this system component, the translated complex SPARQL statements are broken down into multiple simple SPARQL statements and sent to the database in such a way that the returned intermediate results are filtered by it as well as used for the next query statement. The outcome of this process provides identical information for the KML generation process as the original PostGIS. A detailed flow of activities is presented in the UML diagram included in appendix B.

These two components of the *CityExportAgent* address problems that arise upon attempting to manage CIMs larger than a single city. For example, Buyukdemircioglu and Kocaman [9], who worked on a model of Turkey, stated that: 'Currently, the main issue in web-based visualization of 3D city models using open-source software is the model updating. Since the visualized model is static and not directly visualized from a database or a similar dynamic source, the whole model must be generated again every time there is an update on the model. Serving a city model with topography directly from a spatial database management system (DBMS) to the web interface would eliminate this problem. Efficient geospatial database solutions should be developed for high-performance visualization of city and terrain models, which is of great importance for updating such models.' The Semantic 3D City Database [11] provides such dynamic spatial management system in the CKG, and the *CityExportAgent* automates data management for the visualisation of such dynamic geospatial models. It is hence possible to trigger an automated KML export by the agent whenever there is a change in the underlying knowledge graph, and dynamically reflect that change in the visualisation layer. Figure 6 provides an illustration of the visualisations exported by the CityExportAgent.

In terms of cognitive architecture, the agent demonstrates following capabilities: recognition, decision making, choice, monitoring, execution, action, interaction, and communication. When evaluated, it also proves to be: efficient, scalable, reactive, persistent, autonomous and improvable [33].

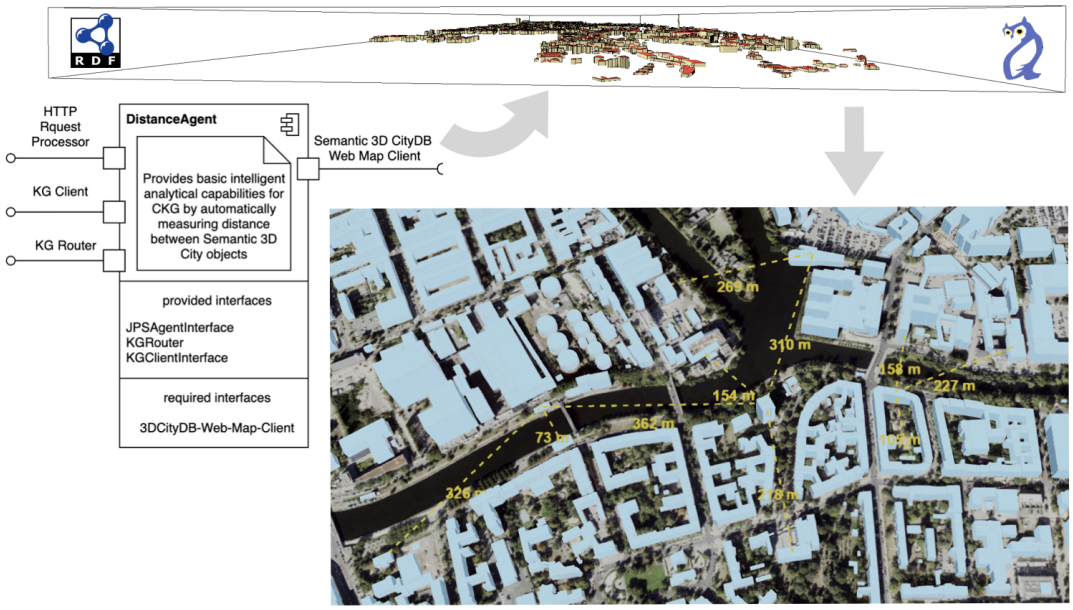


Figure 7: *The DistanceAgent automatically calculates distances between city object representations, which were interacted with on the web map client. It dynamically manifests the acquired knowledge by displaying the learned information about spatial relationships through connection lines and distance values whenever it is ready.*

3.3 Distance Agent

The *DistanceAgent* is an example of an autonomous agent that operates on semantic structures found in representations of elements of a built environment - it automatically computes physical distances between city objects stored in the Semantic 3D City Database of the CKG. The agent design is based on the cognitive information architecture of TWA (section 2) and contains methods to locate a data store by means of a resource, allowing it to retrieve or insert relevant information using *StoreRouter* and *RemoteStoreClient*. The agent also contains methods and interfaces for information processing. All methods are presented in the UML diagram included in appendix C.

The agent makes sense of the represented environment that it operates on by independently tracking information about events occurring on the 3DCityDB-Web-Map-Client. When triggered by an interaction event, the *DistanceAgent* receives a HTTP POST request with the request parameters - city object IRIs - in JSON format. Upon successful request validation, the agent checks whether the corresponding distance information already exists in the CKG, which it does by invoking the query builder to form a query to retrieve that particular distance. If the executed query does not return any distance information, the following set of tasks are performed by the agent in order to compute the distance.

The *DistanceAgent* executes query statements with the OntoCityGML schema against the CKG and retrieves the city object envelopes linked to IRIs in the HTTP request. As the envelope essentially defines a bounding box for any type of geospatial CityGML2.0 object

– 1D points, 2D polygons or 3D objects – the agent handles heterogeneous spatial data universally. Envelopes are used to extract city objects' centroids, between which distances are computed. Using envelopes with only four unique coordinates, reduces the centroid computation effort when original geometries are complex or when the triple store contains heterogeneous geospatial data. Additionally, to ensure accurate results, a query statement is executed to retrieve the native namespace's Coordinate Reference System (CRS) and set the centroids to a uniform (Cartesian) target CRS. When computing the distance, the agent also considers Z coordinate values, further improving the accuracy, as well as broadening the potential scope of an application. The returned distance values between city object pairs are formalised and described in terms of Units of Measure Ontology (OM) [47] and, by executing SPARQL statements, added to the CKG in a separate named graph.

The *DistanceAgent* can compute distances for city objects spread across multiple named graphs in separate namespaces, as a city object TWA IRI structure carries the necessary information, such as namespace and graph name, to retrieve its precise location in the triple store. It also works independently from the Importer/Exporter tool (i.e. the semantically augmented version utilised by the *CityImportAgent* and the *CityExportAgent*), as it directly tracks interactions occurring on the 3DCityDB-Web-Map-Client. It manifests the acquired knowledge (i.e. visualises distances) by highlighting the elements of the city objects' representation between which distances are computed, as well as learned information about relationships between them (Fig. 7).

In principle, it is possible to extend the design pattern of the *DistanceAgent* to support more advanced analytical features in the future. For example, it could analyse topological relationships between representations of elements found in a built environment based on distances, or analyse the prevalence of computed distances for an object over time as a proxy for interest in that object. Nevertheless, when its cognitive architecture is taken into account, the agent already demonstrates the following capabilities: recognition, decision making, choice, monitoring, execution, action, interaction, communication as well as acquisition, representation, refinement and organisation of knowledge. When evaluated, it also proves to be: efficient, scalable, reactive, persistent, autonomous and improvable [33].

4 Conclusions and Future Work

A system of intelligent autonomous agents, such as above described Semantic 3D City Agents, in combination with a knowledge graph, such as CKG, provide a suite of solutions to address common gaps in current geospatial information systems that make them unsuitable to meet demands imposed by the fourth industrial revolution. However, adhering to the existing standards as well as reusing, adapting and integrating existing components of such systems stays in accordance with the sustainable digitisation practices. This paper presented one of the ways of achieving such goal.

Cognitive architecture of the JPS Agent Framework, elaborated on in section 2, provides a backbone for flexible creation of intelligent autonomous agents, compliant with existing web standards, in TWA. Routed Knowledge Graph Access components described in subsection 2.2, namely the Store Router (subsection 2.2.1) and the Store Clients (sub-

Table 1: *City planning related questions answered by the live CKG subsystem of TWA.*

Query No.	Question	No. of Solutions	Elapsed Time
1.	Ask if a certain building function exists in the dataset.	1	31 ms.
2.	Ask if a certain street name exists in the dataset.	1	141 ms.
3.	Return all data about a building in a specified address.	117	4063 ms.
4.	Return all distinct generic attribute names found in the dataset.	56	19141 ms.
5.	Return all distinct building function codes in the dataset.	208	3828 ms.
6.	Return all generic attribute names and their values found in the dataset.	56	58296 ms.
7.	Return specified generic attribute "Qualitaet" with all its values found in the dataset and order solutions by the value in descending order.	74919	91145 ms.
8.	Return all distinct street names found in the dataset. Count the number of buildings in every street.	8874	4063 ms.
9.	Return all distinct street names that have building function code "1134" in it. Count how many times that function occurred and order streets by the number of function occurrence in descending order.	67	3703 ms.
10.	Return all distinct street names that have building function code "1444" in it. Count the ratio of the specified function in each street. Order results by the ratio in descending order.	799	23314 ms.

section 2.2.2), provide means to navigate through elements that represent objects spanning multiple domains of knowledge encoded in a form compliant with the semantic web standards and recommendations by the W3C. Section 3 demonstrated capabilities of the a system of Semantic 3D City Agents based on the framework that make use of the access components. The CityImportAgent (subsection 3.1) demonstrated capability of fully automated creation of multifaceted CIMs in an ontological form, compliant with the CityGML 2.0 standard at the same time. The model of Berlin, autonomously created by the agent, consists of 419 909 661 triples and shows satisfactory performance on the set of sample city planning questions presented in Tables 1 and 2. The CityExportAgent, described next (subsection 3.2), shows possibility of data management for visualisation of the CIMs stored in dynamic geospatial knowledge graphs on the fly and autonomously reflecting changes in them on the web map client. The DistanceAgent (subsection 3.3) demonstrates example autonomous analytical capabilities for such semantic CIMs by independently tracking interactions with their elements on the web visualisation. The agent's virtual sensors capture symbolic inputs that correspond to those interactions in the background while its programs enrich the underlying CIMs with the newly derived knowledge, stored in the dynamic geospatial knowledge graph of the CKG, and presented back on the web map client autonomously by sending the symbolic outputs to it via virtual

actuators.

Table 2: *City planning questions answered by the live CKG subsystem of TWA. (continued from Table 1)*

Query No.	Question	No. of Solutions	Elapsed Time
11.	Return all street names that have building function code "1171" in it. Return other existing functions in those streets. Count and order other functions by their occurrence in descending order.	1574	29520 ms.
12.	Return the average height of each function code found in the dataset and order functions by average height in descending order.	208	3875 ms.
13	Return all distinct street names found in the dataset. Count average, minimum and maximum height of each street. Order solutions by average height a descending order."	8211	8829 ms.
14.	Return all distinct streets found in the dataset. Order solutions by the number of buildings in the street and by the average height in ascending order.	8211	8874 ms.
15.	Return all distinct street names that have at least one of the specified function codes. Order solutions by the number of matched function codes in descending order and by the average street height in ascending order.	604	4375 ms.
16.	Return all addresses of buildings with function code "2921" found in the dataset. Order results by street name and number in ascending order.	2226	3969 ms.
17.	Return all addresses and envelope coordinates of buildings with function code "2921" found in the dataset.	2215	4250 ms.
18.	Count all city object Ids found within a given boundary.	1	16 ms.
19.	Return all building footprints found within a boundary.	258	31 ms.
20.	Count all city objects found in the dataset based on its type: GroundSurface, RoofSurface, WallSurface, BuildingPart, Building and CityObjectGroup.	6	3125 ms.

Apart from reusing existing components this work required design and development of the following novel elements:

- JPS Agent Framework that generalised upon the past agent development in the J-Park Simulator and provided a set of uniform interfaces to all TWA agents allowing them to implement cognitive capabilities as well.

- Routed Knowledge Graph Access components that allow the JPS Agents to navigate TWA knowledge graph by consulting routing information encoded in the OntoKGRouter ontology, introduced for this purpose, and access the heterogenous multi-domain data in a technology agnostic manner.

- Semantic 3D City Agents that demonstrate intelligent automation for dynamic geospatial knowledge graphs and adhere to the sustainable digitalisation practices as well as allow to adapt existing geospatial information systems' components into applications that meet the needs imposed by Industry 4.0.

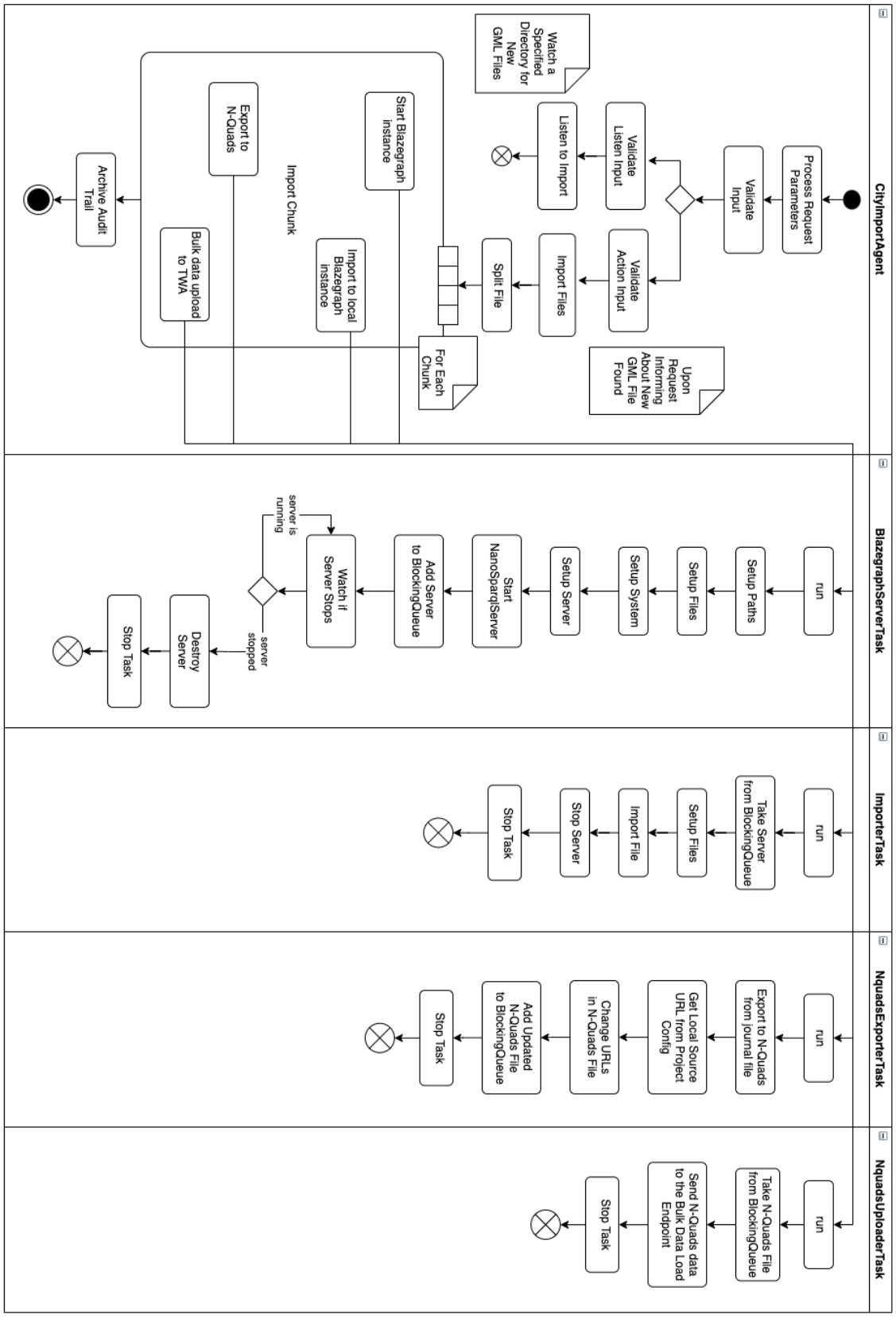
Automated CIM instantiation, dynamic visualisation and autonomous analytics solely are not sufficient to conduct complex multi-factor optimisations, that would allow to assist with solving the most pressing problems faced by the countries across the globe at the present time, even when such datasets are combined with other domain data within TWA. In order to get closer to some of the answers, its Parallel World Framework needs to be adapted to work with large scale simulations combining the CIM data with other datasets representing different domains. To achieve this, agents and mechanisms need to be developed that allow efficient modification of the CIM as well as elimination of some of its components. Although it is currently possible to simply override existing models with new data, the efficient update and delete capabilities are not yet present in the TWA. Such large scale simulations would result in the increased demand for data storage. In order to minimise the impact of it, improved agent communication in a form of ontology that provides vocabularies necessary for the agents to perform forms of inferencing while exchanging symbolic inputs could be developed as well. This would form a basis of an agent communication language allowing JPS Agents to organise the underlying datasets, required to perform such large scale simulations. This future work could be also regarded as a next brick on paving the road towards self-sustainable knowledge graphs.

Acknowledgements

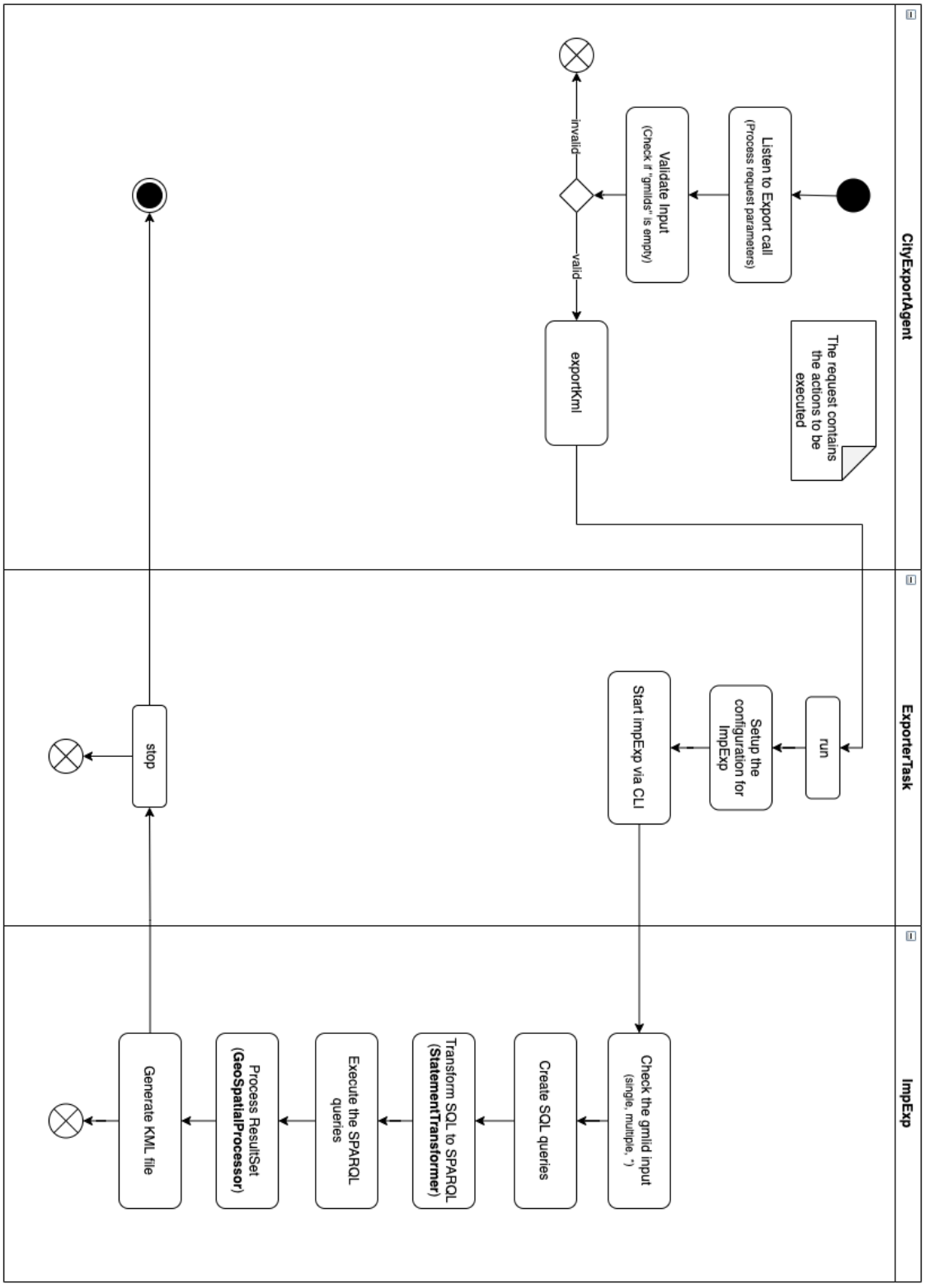
This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme. Markus Kraft gratefully acknowledges the support of the Alexander von Humboldt foundation.

The research was conducted as part of an Intra-CREATE collaborative project involving CARES (Cambridge Centre for Advanced Research and Education in Singapore), which is University of Cambridge's presence in Singapore, and Future Cities Laboratory at the Singapore-ETH Centre, which was established collaboratively between ETH Zurich and the National Research Foundation Singapore.

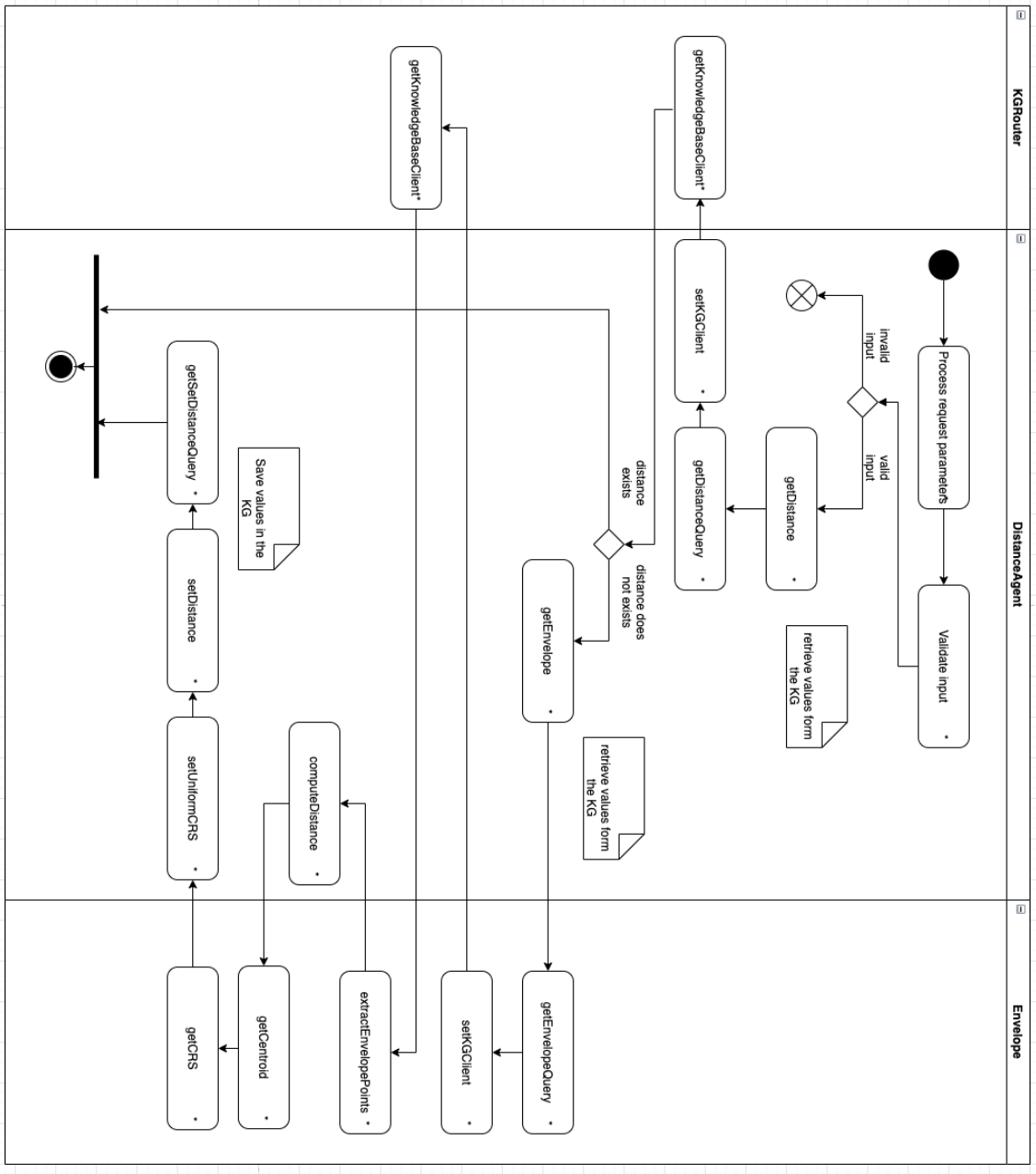
A City Import Agent – UML Activity Diagram



B City Export Agent – UML Activity Diagram



C Distance Agent – UML Activity Diagram



List of Abbreviations

EIP	**E**co-**I**ndustrial **P**ark
ETH	**E**idgenössische **T**echnische **H**ochschule
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
URL	**U**niform **R**esource **L**ocator
URI	**U**niform **R**esource **I**dentifier
IRI	**I**nternationalized **R**esource **I**dentifier
JPS	**J**-**P**ark **S**imulator
JSON	**J**ava**S**cript **O**bject **N**otation
OWL	**W**eb **O**ntology **L**anguage
RDF	**R**esource **D**escription **F**ramework
TUM	**T**echnische **U**niversität **M**ünchen
RDBMS	**R**elational **D**atabase **M**anagement **S**ystem
SQL	**S**tructured **Q**uery **L**anguage
SPARQL	**S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage
W3C	**W**orld **W**ide **W**eb **C**onsortium
XML	E**x**tensible **M**arkup **L**anguage
GML	**G**eography **M**arkup **L**anguage
JDBC	**J**ava **D**ata**b**ase **C**onnectivity
TWA	**T**he **W**orld **A**vatar
OWA	**O**pen **W**orld **A**ssumption
CWA	**C**losed **W**orld **A**ssumption
DL	**D**escription **L**ogic
OBDA	**O**ntology-**B**ased **D**ata **A**ccess
OGC	**O**pen **G**eospatial **C**onsortium
WKT	**W**ell-**K**nown **T**ext
CKG	**C**ities **K**nowledge **G**raph
CARES	**C**entre for **A**dvanced **R**esearch and **E**ducation in **S**ingapore
SEC	**S**ingapore-**E**TH **C**entre
LOD2	**L**evel **O**f **D**etail 2
CIM	**C**ity **I**nformation **M**odel
DT	**D**igital **T**win

References

[1] J. Akroyd, S. Mosbach, A. Bhave, and M. Kraft. Universal digital twin – a dynamic knowledge graph. *Data-Centric Engineering*, 2:e14, 2021. doi:10.1017/dce.2021.10.

[2] K. Al Kalbani and A. B. Abdul Rahman. Appropriateness of using CityGML standard version 2.0 for developing 3D city model in Oman. In M. Ben Ahmed, İ. Rakıp Karaş, D. Santos, O. Sergeyeva, and A. A. Boudhir, editors, *Innovations in Smart Cities Applications Volume 4*, pages 1332–1343, Cham, 2021. Springer International Publishing.

[3] Apache Tomcat, 2021. URL https://tomcat.apache.org/. Accessed 21st October 2021.

[4] Asynchronous Watcher Service, 2021. URL https://github.com/cambridge-cares/TheWorldAvatar/tree/develop/AsynchronousWatcherService. Accessed 18th October 2021.

[5] S. Ates Aydar, J. Stoter, H. Ledoux, E. Demir Ozbek, and T. Yomralioglu. Establishing a national 3D geo-data model for building data compliant to CityGML: Case of Turkey. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLI-B2:79–86, 2016. doi:10.5194/isprs-archives-XLI-B2-79-2016.

[6] Blazegraph - DataMigration, 2021. URL https://github.com/blazegraph/database/wiki/DataMigration. Accessed 20th October 2021.

[7] Blazegraph - GeoSpatial, 2021. URL https://github.com/blazegraph/database/wiki/GeoSpatial. Accessed 20th April 2021.

[8] Blazegraph - NanoSparqlServer, 2021. URL https://github.com/blazegraph/database/wiki/NanoSparqlServer. Accessed 20th October 2021.

[9] M. Buyukdemircioglu and S. Kocaman. Reconstruction and efficient visualization of heterogeneous 3D city models. *Remote Sensing*, 12(13), 2020. doi:10.3390/rs12132128.

[10] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs. *Journal of Web Semantics*, 3(4):247–267, 2005. doi:10.1016/j.websem.2005.09.001. World Wide Web Conference 2005 – Semantic Web Track.

[11] A. Chadzynski, N. Krdzavac, F. Farazi, M. Q. Lim, S. Li, A. Grisiute, P. Herthogs, A. von Richthofen, S. Cairns, and M. Kraft. Semantic 3D city database – an enabler for a dynamic geospatial knowledge graph. *Energy and AI*, 6:100106, 2021. doi:10.1016/j.egyai.2021.100106.

[12] H.-C. Chiang. Data modelling and application of 3D cadastre in Taiwan. In *Proceedings of the 3rd International Workshop on 3D Cadastres: Developments and Practices, Shenzhen, China*, 2012. URL https://www.oicrf.org/-/data-

modelling-and-application-of-3d-cadastre-in-taiwan. Accessed 20th September 2021.

[13] Cities Knowledge Graph - CARES, 2021. URL https://www.cares.cam.ac.uk/research/cities/. Accessed 15th April 2021.

[14] Cities Knowledge Graph - ResearchGate, 2021. URL https://www.researchgate.net/project/Cities-Knowledge-Graph. Accessed 15th April 2021.

[15] Cities Knowledge Graph - SEC, 2021. URL https://fcl.ethz.ch/research/research-projects/cities-knowledge-graph.html. Accessed 15th April 2021.

[16] A. Devanand, M. Kraft, and I. A. Karimi. Optimal site selection for modular nuclear power plants. *Computers & Chemical Engineering*, 125:339–350, 2019. doi:10.1016/j.compchemeng.2019.03.024.

[17] A. Devanand, G. Karmakar, N. Krdzavac, R. Rigo-Mariani, E. Y. S. Foo, I. A. Karimi, and M. Kraft. OntoPowSys: A power system ontology for cross domain interactions in an eco industrial park. *Energy and AI*, 1:100008, 2020. doi:10.1016/j.egyai.2020.100008.

[18] A. Eibeck, M. Q. Lim, and M. Kraft. J-Park Simulator: An ontology-based platform for cross-domain scenarios in process industry. *Computers & Chemical Engineering*, 131:106586, 2019. doi:10.1016/j.compchemeng.2019.106586.

[19] A. Eibeck, A. Chadzynski, M. Q. Lim, L. K. Aditya, L. Ong, A. Devanand, G. Karmakar, S. Mosbach, R. Lau, I. A. Karimi, E. Y. S. Foo, and M. Kraft. A parallel world framework for scenario analysis in knowledge graphs. *Data-Centric Engineering*, 1:e6, 2020. doi:10.1017/dce.2020.6.

[20] F. Farazi, J. Akroyd, S. Mosbach, P. Buerger, D. Nurkowski, M. Salamanca, and M. Kraft. OntoKin: An ontology for chemical kinetic reaction mechanisms. *Journal of Chemical Information and Modeling*, 60(1):108–120, 2020. doi:10.1021/acs.jcim.9b00960.

[21] F. Farazi, M. Salamanca, S. Mosbach, J. Akroyd, A. Eibeck, L. K. Aditya, A. Chadzynski, K. Pan, X. Zhou, S. Zhang, M. Q. Lim, and M. Kraft. Knowledge graph approach to combustion chemistry and interoperability. *ACS Omega*, 5 (29):18342–18348, 2020. doi:10.1021/acsomega.0c02055.

[22] J. Gil. City Information Modelling: A Conceptual Framework for Research and Practice in Digital Urban Planning. *Built Environment*, 46(4):501–527, 2020. doi:10/ghqp88.

[23] G. Gröger, T. H. Kolbe, C. Nagel, and K. H. Häfele. OGC City Geography Markup Language (CityGML) En-coding Standard, 2012. URL https://www.ogc.org/standards/citygml. Accessed 11th March 2021.

[24] Java Collections Framework, 2021. URL https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html. Accessed 21st October 2021.

[25] Java Runnable Interface, 2021. URL https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html. Accessed 21st October 2021.

[26] Java Servlet Technology Overview, 2021. URL https://www.oracle.com/java/technologies/servlet-technology.html. Accessed 11th October 2021.

[27] Jena JDBC - A SPARQL over JDBC driver framework, 2021. URL https://jena.apache.org/documentation/jdbc/. Accessed 27th October 2021.

[28] JPS Base Lib, 2021. URL https://github.com/cambridge-cares/TheWorldAvatar/tree/develop/JPS_BASE_LIB. Accessed 06th October 2021.

[29] M. J. Kleinelanghorst, L. Zhou, J. Sikorski, E. Y. S. Foo, K. Aditya, S. Mosbach, I. Karimi, R. Lau, and M. Kraft. J-Park Simulator: Roadmap to smart eco-industrial parks. In *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*, 2017. doi:10.1145/3018896.3025155.

[30] J. Kopecký, K. Gomadam, and T. Vitvar. hRESTS: An HTML microformat for describing RESTful web services. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '08, page 619–625, USA, 2008. IEEE Computer Society. doi:10.1109/WIIAT.2008.379.

[31] I. Kotseruba and J. K. Tsotsos. 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review*, 53(1):17–94, 2020. doi:10.1007/s10462-018-9646-y.

[32] N. Krdzavac, S. Mosbach, D. Nurkowski, P. Buerger, J. Akroyd, J. Martin, A. Menon, and M. Kraft. An ontology and semantic web service for quantum chemistry calculations. *Journal of Chemical Information and Modeling*, 59(7):3154–3165, 2019. doi:10.1021/acs.jcim.9b00227.

[33] P. Langley, J. E. Laird, and S. Rogers. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, 10(2):141–160, 2009. doi:10.1016/j.cogsys.2006.07.004.

[34] Y. Liao, F. Deschamps, E. de Freitas Rocha Loures, and L. F. P. Ramos. Past, present and future of Industry 4.0 – a systematic literature review and research agenda proposal. *International Journal of Production Research*, 55(12):3609–3629, 2017. doi:10.1080/00207543.2017.1308576.

[35] Y. Lu. Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration*, 6:1–10, 2017. doi:10.1016/j.jii.2017.04.005.

[36] Ministry of the Environment and Water Resources. Zero Waste Masterplan Singapore, 2019. URL https://www.towardszerowaste.gov.sg/images/zero-waste-masterplan.pdf. Accessed 20th September 2021.

[37] OGC Asia Forum - ShowCases, 2021. URL http://www.ogc.org.tw/en-US/ShowCases. Accessed 06th October 2021.

[38] OWASP - Input Validation, 2021. URL https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html. Accessed 12th October 2021.

[39] M. Pan, J. Sikorski, C. A. Kastner, J. Akroyd, S. Mosbach, R. Lau, and M. Kraft. Applying Industry 4.0 to the Jurong Island eco-industrial park. *Energy Procedia*, 75:1536–1541, 2015. doi:10.1016/j.egypro.2015.07.313.

[40] M. Pan, J. Sikorski, J. Akroyd, S. Mosbach, R. Lau, and M. Kraft. Design technologies for eco-industrial parks: From unit operations to processes, plants and industrial networks. *Applied Energy*, 175:305–323, 2016. doi:10.1016/j.apenergy.2016.05.019.

[41] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis. Microservices in practice, part 1: Reality check and service design. *IEEE Software*, 34(1): 91–98, 2017. doi:10.1109/MS.2017.24.

[42] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis. Microservices in practice, part 2: Service integration and sustainability. *IEEE Software*, 34 (2):97–104, 2017. doi:10.1109/MS.2017.56.

[43] C. Pedrinaci and J. Domingue. Toward the next wave of services: Linked services for the web of data. *Journal of Universal Computer Science*, 16(13):1694–1719, 2010. doi:10.3217/jucs-016-13-1694.

[44] A. Perera, K. Javanroodi, and V. M. Nik. Climate resilient interconnected infrastructure: Co-optimization of energy systems and urban morphology. *Applied Energy*, 285:116430, 2021. doi:10.1016/j.apenergy.2020.116430.

[45] S. Poslad. Specifying protocols for multi-agent systems interaction. *ACM Transactions on Autonomous and Adaptive Systems*, 2(4):15–es, 2007. doi:10.1145/1293731.1293735.

[46] F. Prandi, F. Devigili, M. Soave, U. Di Staso, and R. De Amicis. 3D web visualization of huge CityGML models. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-3/W3:601–605, 2015. doi:10.5194/isprsarchives-XL-3-W3-601-2015.

[47] H. Rijgersberg, M. Van Assem, and J. Top. Ontology of units of measure and related concepts. *Semantic Web*, 4(1):3–13, 2013.

[48] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 3rd edition, 2010.

[49] M. Sir, Z. Bradac, and P. Fiedler. Ontology versus database. *IFAC-PapersOnLine*, 48(4):220–225, 2015. doi:10.1016/j.ifacol.2015.07.036. 13th IFAC and IEEE Conference on Programmable Devices and Embedded Systems.

[50] A. Stadler, C. Nagel, G. König, and T. H. Kolbe. *Making Interoperability Persistent: A 3D Geo Database Based on CityGML*, pages 175–192. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi:10.1007/978-3-540-87395-2_11.

[51] UN Security Council. Climate change 'biggest threat modern humans have ever faced', world-renowned naturalist tells security council, calls for greater global cooperation, 2021. URL https://www.un.org/press/en/2021/sc14445.doc.htm. Accessed 20th September 2021.

[52] A. von Richthofen, P. Herthogs, M. Kraft, and S. Cairns. Semantic City Planning Systems (SCPS): A Literature Review. c4e-Preprint Series Technical Report 270, Singapore ETH Centre, Cambridge, 2021. URL https://como.ceb.cam.ac.uk/preprints/270/.

[53] U. Winkelhake. *Roadmap for Sustainable Digitisation*, pages 127–178. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-71610-7_6.

[54] O. Wysocki. Awesome CityGML, 2021. URL https://github.com/OloOcki/awesome-citygml/. Accessed 17th September 2021.

[55] Z. Yao, C. Nagel, F. Kunde, G. Hudra, P. Willkomm, A. Donaubauer, T. Adolphi, and T. H. Kolbe. 3DCityDB – a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML. *Open Geospatial Data, Software and Standards*, 3(1):5, 2018. doi:10.1186/s40965-018-0046-7.

[56] C. Zhang, A. Romagnoli, L. Zhou, and M. Kraft. Knowledge management of eco-industrial park for efficient energy utilization through ontology-based approach. *Applied Energy*, 204:1412–1421, 2017. doi:10.1016/j.apenergy.2017.03.130.

[57] L. Zhou, M. Pan, J. J. Sikorski, S. Garud, L. K. Aditya, M. J. Kleinelanghorst, I. A. Karimi, and M. Kraft. Towards an ontological infrastructure for chemical process simulation and optimization in the context of eco-industrial parks. *Applied Energy*, 204:1284–1298, 2017. doi:j.apenergy.2017.05.002.

[58] L. Zhou, C. Zhang, I. A. Karimi, and M. Kraft. An ontology framework towards decentralized information management for eco-industrial parks. *Computers & Chemical Engineering*, 118:49–63, 2018. doi:10.1016/j.compchemeng.2018.07.010.

[59] X. Zhou, A. Eibeck, M. Q. Lim, N. Krdzavac, and M. Kraft. An agent composition framework for the J-Park Simulator – a knowledge graph for the process industry. *Computers & Chemical Engineering*, 130:106577, 2019. doi:10.1016/j.compchemeng.2019.106577.

[60] X. Zhou, M. Q. Lim, and M. Kraft. A smart contract-based agent marketplace for the J-Park Simulator – a knowledge graph for the process industry. *Computers & Chemical Engineering*, 139:106896, 2020. doi:10.1016/j.compchemeng.2020.106896.