POLITECNICO
MILANO 1863

Laurea Magistrale in Computer Science & Engineering
Scuola di Ingegneria Industriale e dell'Informazione
POLITECNICO DI MILANO

ON MODEL-DRIVEN DESIGN OF CITY SPACES

A bidirectional transformations approach to City Spaces design and operations

ENNIO VISCONTI
M. 874987

Relatore: Prof. Carlo Ghezzi
Correlatore: Dr. Christos Tsigkanos

Anno Accademico 2018 – 2019

A Sergio e Antonella

SOMMARIO

La proliferazione di ambienti intelligenti nella società contemporanea ha fatto sorgere l'esigenza di strumenti di analisi che ne supportino un'ingegnerizzazione sistematica. Il progresso tecnologico che ha dato forma a questa realtà ha creato un mondo in cui elementi fisici e computazionali interagiscono e possono esibire comportamenti complessi, come quelli di *smart city* e *smart building*. Le tecniche basate su modelli si sono affermate come un metodo per facilitarne lo sviluppo e l'analisi. Nello spirito di tali approcci, il modello dello spazio fisico ereditato dall'architettura e dall'ingegneria civile è trasformato in un modello *analizzabile*, all'interno del quale vengono innestate funzionalità intelligenti. Tali modelli possono allora essere analizzati formalmente per valutare il design di un sistema composito. Tra le rappresentazioni specifiche di dominio, questa tesi si focalizza su modelli dello spazio fisico specificati nel linguaggio standard CityGML e su come questi possano essere trasformati in modelli apprezzabili da un punto di vista analitico. Si vuole mostrare inoltre, come tali modelli possano essere sincronizzati automaticamente con quelli originali, qualora si verificassero aggiornamenti in ambo le parti. Come modelli *analizzabili*, abbiamo considerato gli spazi cyber-fisici, una rappresentazione che combina elementi fisici e computazionali. Presentiamo il problema in un'impostazione formale, definiamo la relazione di consistenza tra due tipi di modelli e presentiamo un software per la trasformazione automatica bidirezionale tra modelli CityGML e spazi cyber-fisici. Esibiamo, infine, la trasformazione su dei modelli di città reali e concludiamo con una discussione sulle possibili limitazioni ed estensioni.

ABSTRACT

The proliferation of smart environments in contemporary societies has led to demands for analysis facilities to support their systematic engineering. Technological advancements shaping such environments have led to a world in which physical and computational elements interact and may exhibit complex behavior, such as within smart buildings and smart cities. Model-based techniques have been proposed to ease development and analysis. In such an approach, the model of physical space coming from architecture and civil engineering disciplines is transformed into an analyzable model upon which smart functionalities can be embedded. Such models can then be formally analyzed to assess a composite system design. Among domain-specific descriptions, this thesis focuses on how a model of physical space specified in the CityGML standard language can be transformed into a model amenable to analysis and how the two models can be automatically kept in sync after possible changes. The analyzable model is a cyber-physical space, a representation combining physical and computational elements. We discuss the problem in a formal setting and define a consistency relation between the two types of models, and present a software tool for automatic bidirectional transformations between CityGML models and cyber-physical spaces. We showcase transformations over real city models and conclude with a discussion on limitations and possible extensions.

Whenever a theory appears to you
as the only possible one, take this as a sign
that you have neither understood the theory
nor the problem which it was intended to solve.

— Sir Karl Raimund Popper [38]

RINGRAZIAMENTI

Vorrei ringraziare il prof. Carlo Ghezzi, relatore di questa tesi, non solo per gli stimoli e il supporto durante questo lavoro di ricerca, ma soprattutto per la fiducia nei miei confronti, e per essere stato un ineguagliabile riferimento di passione per la conoscenza e vivacità intellettiva. Vorrei ringraziare il dr. Christos Tsigkanos, correlatore e riferimento fondamentale, per avermi guidato in questo processo: i suoi contributi sono stati di inestimabile valore intellettivo e pratico, e il suo ottimismo è stato indispensabile per concludere con efficacia questo percorso. E' doveroso anche ringraziare il prof. Zhenjiang Hu, che mi ha accolto con enorme disponibilità nel suo laboratorio di Tokyo, e la cui fiducia, la curiosità e gli insegnamenti sono stati elementi imprescindibili nel guidare la mia ricerca. Il più grande ringraziamento va necessariamente a mia madre e mio padre, miei instancabili sponsor, il cui sostegno, praticamente incondizionato, rimane per me uno stupefacente esempio di amore. Ringrazio le mie sorelle Gabriella, ma soprattutto Camilla, per avermi sopportato e supportato, con affetto e comprensione. Voglio inoltre ringraziare Luca, con cui ho iniziato l'avventura al Politecnico. E con lui Giovanni, Debora, Paolo, Massimiliano, Simone, Marco, Edoardo e Gianluca, ma anche Carlo, Gennaro, Davide, Simona e tanti altri amici e colleghi, più o meno storici, con cui ho condiviso la crescita professionale e intellettuale di questi anni. Alla mia seconda e numerosissima famiglia, Svoltastudenti, che ringrazio tutta, da Vincenzo e Valentina, che hanno ispirato la mia crescita politica, a Carlo e Alessandro, che hanno brillantemente raccolto il testimone, va la mia immensa gratitudine. Ringrazio inoltre le tantissime persone straordinarie che ho conosciuto al Politecnico, dai professori ai dirigenti, al personale amministrativo, ai tanti studenti e ricercatori. E, infine, grazie a Milano, per avermi accolto e regalato le esperienze di crescita uniche che mi portano qui oggi.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

ADE Application Domain Extension

ADT Abstract Data Type

API Application Programming Interface

BIM Building Information Modeling

BX Bidirectional Transformations

CPS Cyber-physical Systems

CPSp Cyber-physical Spaces

DSL Domain-specific Language

GIS Geographical Information Systems

GCS Geometric Constraint Satisfaction

UML Unified Modeling Language

USC United Smart Cities

WCCD World Council on City Data

XSD XML Schema Definition

1 INTRODUCTION

Modern cities are often convoluted environments, in which people interact continuously with extensive facilities for transportation, housing, sanitation, communication, and many others. Many different entities populate them like buildings, streets, gardens, lights or bridges, having several purposes for the community, but also with very different characterizing features: some of them can be historical, some others modern, or naturalistic, or temporary, and so forth. The increasing relocation of people to urban areas, shown in Figure 1.1, has generated a growth for some cities to unprecedented sizes, posing new challenges in designing city services that can accommodate the needs of massive amounts of inhabitants.

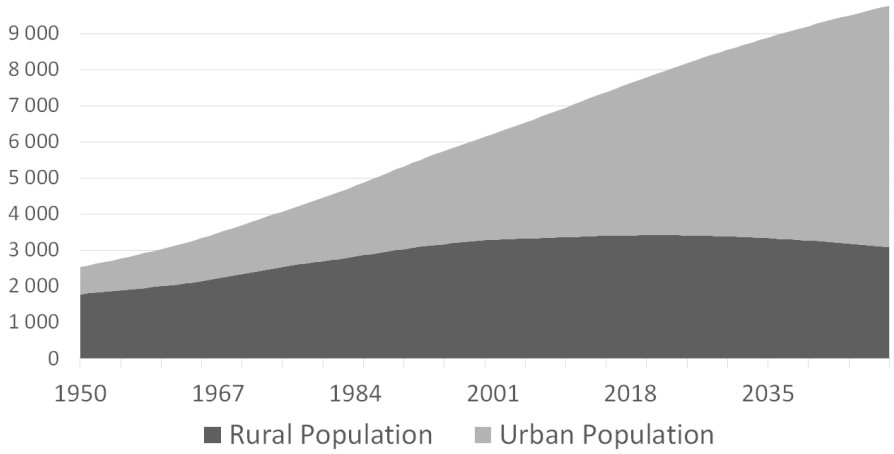


Figure 1.1: World Population (in millions), from 1950 to 2050 (projected from 2018). United Nations data from [53].

Moreover, the evolution of social standards and policies, has brought an intricate network of rules which cover environment preservation, cultural heritage, noise control and energy efficiency among the many, with the ultimate goal of raising the quality of life of people. Within this environment, companies have been deploying new generation devices, augmenting with networking and computation capabilites, to previously non-technological objects. Such systems set novel challenges when designing new buildings and urban areas as well as renovating old ones, requiring to deal with computational and physical aspects at the same time, treating them as *cyber-physical systems*. The development of such space-dependent, cyber-physical systems

demands for software engineering support facilities that span their lifecycle, from design to operation.

1.1 PROBLEM SETTING

As complexity grows, the engineering of cyber-physical systems requires to take into account information from multiple domains together with certainties about the overall system's behavior. We expect from modern buildings and urban areas to be better than old ones: architects and engineers should be able to design them by achieving more security, energy-efficiency, cost-effectiveness, but also astonishing looks that fit in the history and spirit of the city and the people living them.

While the tools used by professionals have evolved significantly over the past decades, the methodology is more or less the same. New demands are addressed in the same old way, resulting in a process that gets harder and more error-prone.

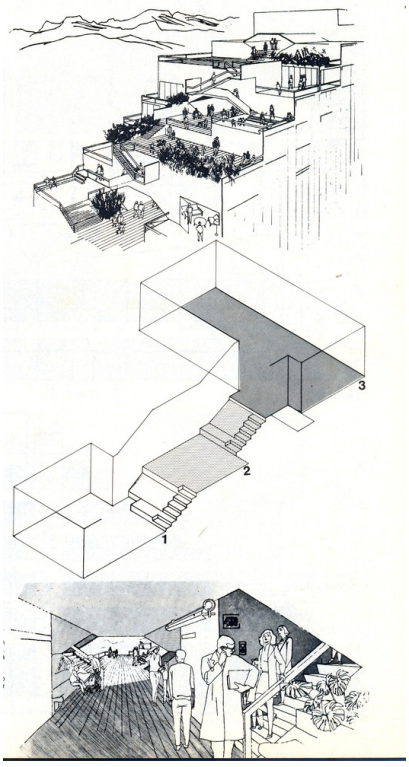


Figure 1.2: Different aspects evaluated at the design stage of a building [21].

We believe the time is right for a new methodological revolution in the way architects, engineers, and domain experts address their work. Since models already play a crucial role in the mental process of domain experts, we belive a new level of automatic analysis can be achieved in terms of software support and model-driven engineering.

The engineering can, in principle, be enabled with model representations of their spatial environment: such representations can be sourced from domain models originating in other disciplines and dominated by their practices, tools and domain knowledge. Although relying on international standards and accessible in machine-readable formats, such physical space descriptions are still often intended for static documentation or domain-specific purposes. The resulting models are therefore of non-easily analyzable types, which hinders their consideration for engineering software-intensive, composite cyber-physical systems. Our idea is to use exactly the same spatial domain models used by practitioners to represent urban areas, buildings and city spaces and project from them some abstract and more computationally convenient representation, which can be transformed back to the original one when needed.

1.2 CONTRIBUTIONS

While advancements in cyber-physical systems have shown great analytical potential, they collide with the complexity of representing this reality. Conversely, domain experts have been extending the capabilities of the representation they use, to support the integration of newer sources of information. Our goal is to obtain the best of both worlds by building the infrastructure to bridge this gap. This resulted in a comprehensive work composed of: (i) the analysis of the process and the available technologies; (ii) the definition of a formal relationship among these representations and (iii) the implementation of a development framework to bridge this gap automatically.

We must point out that the novelty of our approach is testified by the absence of other work on this topic to date, with the only exception of [54] from the same authors of this thesis. Indeed, we take from it the vast majority of the contents of this work, as well as the results, and we refer to that for a more concise view, albeit earmarked for a knowledgeable audience.

The interested reader may refer to `https://topo.city` for models and code that accompany this thesis. Contributions of this thesis where presented in the 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS) conference in the paper *Model-Driven Design of City Spaces via Bidirectional Transformations* by Ennio Visconti, Christos Tsigkanos, Zhenjiang Hu and Carlo Ghezzi.

1.3 OUTLINE

The rest of this work is structured as follows.

- **Chapter 2** introduces the field on which our research grounds, presenting the main concepts and goals characterizing our approach.
- **Chapter 3** provides the appropriate support to describe the concepts previously introduced, supplying the background to properly understand the research.
- **Chapter 4** constitutes probably the core of our contribution: it formally specifies the transformation among the models, presents a transformation strategy to satisfy such specification and analyzes theoretical limitations of the approach.
- **Chapter 5** presents the software artifact developed as a prototypical tool based on our approach.
- **Chapter 6** shows the results of our research on some real-world scenarios, analyzes performances and discusses the major advantages and drawbacks of our approach.
- **Chapter 7** classifies the current state of the art in the research areas linked to our work.
- **Chapter 8** concludes the work and highlights some possible areas for future developments.

2 MODEL-BASED ENGINEERING OF CITY SPACES

Given the long life of urban areas and the impact they have on society, it is essential to model all aspects of a building carefully. As anticipated in Section 1.1, in fact, models have been essential in the design and development process of architects, urban planners, and civil engineers. However, the rise of cyber-physical and smart systems has led to computational facilities being embedded in physical spaces to support complex functionalities. These composite systems demand life-cycle support, from design and validation to runtime reasoning.

We will investigate these aspects by following three directives. First, we will consider the perspective of domain experts in Section 2.1. Following, we will acknowledge the analytical possibilities of recent results in Computer Science in Section 2.2. Finally, we will explain the problem of bridging this gap, and the issues we found in our implementation, respectively, in Sections 2.3 and 2.4.

2.1 THE DOMAIN PERSPECTIVE

Since the very beginning, both architects and engineers have been relying on representations of the physical space. However, software design tools have enabled the production of new artifacts, which are much more complex, more comfortable to build, and less prone to errors. These may contain geometrical or geographical information to describe various kinds of physical objects, like buildings, streets, or even city areas.

Experts community have been reaching consensus on new standardized international representations, to simplify communication and interoperability [18, 28]. Among these efforts, some have emerged for their high expressivity, and for being available in machine-readable formats. A popular choice for modeling building is the Building Information Modeling (BIM) [18] format, which provides a rich set of features in a 3D representation defined over a relative coordinate system. Conversely, more on the side of geographical wide-area representations, CityGML [15] has become very popular. Its strength founds on the capability of describing multiple city objects by using a high-preciseness geographical coordinate system and with different levels of details.

The development cycle of a system at the design phase rests on some of these, depending on the goal. CityGML outstands alternatives, providing a unified representation for all of them. Think of an architect altering the design of a building or an urban planner changing the road

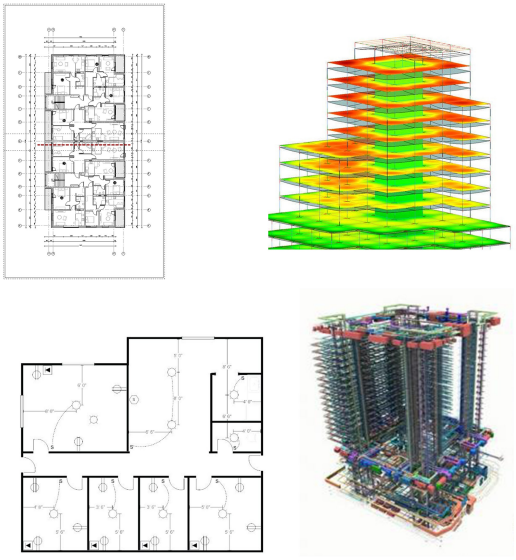


Figure 2.1: Various representation of digital models built by domain experts.

transport routes; these are domain-specific actions, ruled by domain knowledge of the particular matter (e.g., transportation) enabled by these models.

Despite useful in their fields, the latter mainly serve for static documentation or domain-specific purposes. Problems like scale, adaptability, or runtime reasoning, which are primary concerns in composites cyber-physical environments, have never been designing goals of these representations. This lack becomes evident when they try to satisfy some quality attributes related to more complex applications, like changing the design in order to arrange an evacuation plan.

2.2 THE ANALYSIS PERSPECTIVE

On the other side of the spectrum, Computer Science research has achieved significant results in the field of Model-Driven Engineering and Cyber-physical Systems (CPS) [49]. A compelling kind of CPS is Cyber-physical Spaces (CPSp), which consider cases where the system is within a physical space, and which support different types of analysis thanks to the available well-defined formalizations [51]. These are way simpler representations of the environment, where spatial assets correspond to nodes and edges and where transformation rules can enforce advanced properties on the system. These kinds of analyzable models could be useful in the architectural domain [48].

For example, getting back to emergency planning, an expert might simulate the evolution of an extreme event and discover the flaws of their design. Moreover, CAD software could exploit them to support the designer by visualizing the effects of the simulation or maybe even suggest possible solutions to their problem.

Unfortunately, migrating information from the domain is not trivial, as architectural models usually include a wide variety of details that are not relevant for the analysis, and that might significantly increase the computational costs. Moreover, solutions suggested in the context of analyzable models might not be easy to interpret by domain experts, who neither understand the representation, nor the meaning of the changes recommended.

2.3 THE SYNCHRONIZATION CHALLENGE

Synchronization among domain-specific and analyzable models seems to be a reasonable solution to address the problem. One could ideally work on the domain tools and models he is confident with, and an automatic process might convert them, when needed, to perform more advanced levels of analysis. After, when the reasoner finds valuable suggestions, they could be propagated back and offered to the user when suitable. Figure 2.2 illustrates this process.

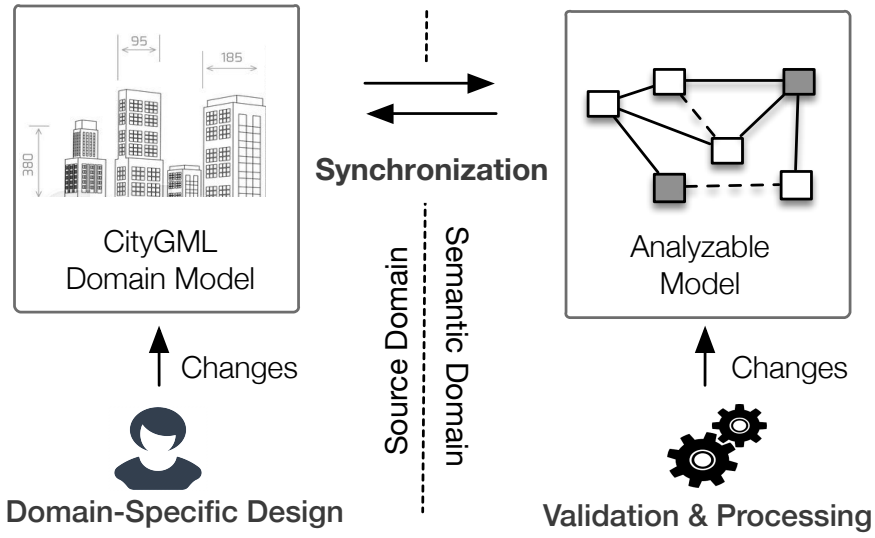


Figure 2.2: Synchronizing different representations of City Spaces.

While this idea might sound simple, synchronizing two sources of information at different levels of abstraction is not trivial at all. The informational asymmetry means that in many cases, a change in the more abstract representation might have multiple interpretations in the domain. What is perhaps even more problematic, is the fact that two independent procedures must be in place: one to migrate the information from the source model to the abstract view, and another to go the other way. This multiplicity generates multiple issues in terms of Software Engineering. First, it means that bugs may appear twice as frequently, resulting in longer times of development, more testing, and more documentation. Moreover, keeping the two

transformations consistent might become more costly over time: a change in the representation on one side would require to update both the programs to follow the new specification.

These traits might soon become impossible to overcome when dealing with multiple domain-specific goals that would demand the developing of a pair of transformations for each of them.

2.4 A BIDIRECTIONAL APPROACH

Although previous setbacks sound discouraging, we address the idea in an alternative way. To deal with the synchronization challenge systematically, we rely on the Bidirectional Transformations (BX) theory. The main idea at their core is to have a clear, formal definition of the *consistency relation* among the two models. Once that is established, various properties and theorems allow to assess the synchronization level, or enforce consistency. Through that, we developed an open development environment that should serve as the foundation for next-generation domain-specific tools, combining the usability and expressiveness of traditional architectural models with advanced graph-based reasoning facilities. We focus our attention on CityGML for domain-specific models, since they already support a wide variety of applications and can incorporate BIM [33].

We advocate that the combination of CityGML and *Cyber-Physical Spaces* can bring promising results to the cause and shed new light on the power of BX, which has been getting popularity in other applications, but has not been considered in this field yet.

3

PHYSICAL SPACES AND THEIR REPRESENTATIONS

The process of engineering space-intensive cyber-physical systems must necessarily start from a representation of the spatial environment. Among the many different models that are used nowadays in the domain of architecture and building engineering, we focused on the CityGML standard, which provides the right level of flexibility to represent a wide variety of scenarios. Conversely, the advances in IoT technologies and public data have lead to the development of new theories, like the one on cyber-physical spaces, to respond to the upcoming demand for new service. Among the different descriptions of cyber-physical spaces we focus on a machine-analyzable data representation, grounded on an extensive and solid mathematical theory [32]. Lastly, to bridge the gap between the two, we develop a consistency relation among objects of them, which we enforce by means of the bidirectional transformations technique.

In this chapter we get into the details of the models we mentioned earlier. We start from our source models in Section 3.1, then we move to the target/view models in Section 3.2. Finally, we devote the final Section to a detailed presentations of the goals we want to accomplish, in order to complete the background for a more formal analysis of the problem in the next chapters.

3.1 CITYGML DESCRIPTIONS AS SOURCE MODELS

3.1.1 *The advantages of CityGML*

Among the many spatial environment descriptions typically available in disciplines related to the construction and building industry, we focus on CityGML. As an industrial standard, CityGML is particularly well suited for our analysis for various reasons: firstly, it is being developed by an open consortium, which is continuously analizing use cases and application scenarios, providing a solid common language for practitioners, and field-specific software companies. Moreover, it is not only capable of representing city-wide spaces, but it also encompasses buildings (i.e. Building Information Models, BIM [33]), and provides a rich system for extending it with contents scoped to the specific purpose for which it is being used. Lastly, its' rise in popularity and the increased availability of public data, has lead to the release of many 3D city models of real cities, together with a set of tools to validate the representation, generate demo data and

compatibility bridges to import and export CityGML representation among most of the softwares related to the field.

Up to date, it has been successfully adopted with the purpose to analyze and take actions in a growing number of scenarios including urban planning, emergency management, traffic noise simulation, navigation systems, urban solar potential estimation, or visual communication [12, 42].

3.1.2 *An overview of the CityGML standard*

To get a view of how CityGML models work, we will briefly describe here its' basic properties. The key idea on which it founds is the combination of submodels with three specific objectives (i.e., space, purpose, and appearance) at five possible levels of details, which can even coexist for the same model whenever it is useful for applications [15, 28].

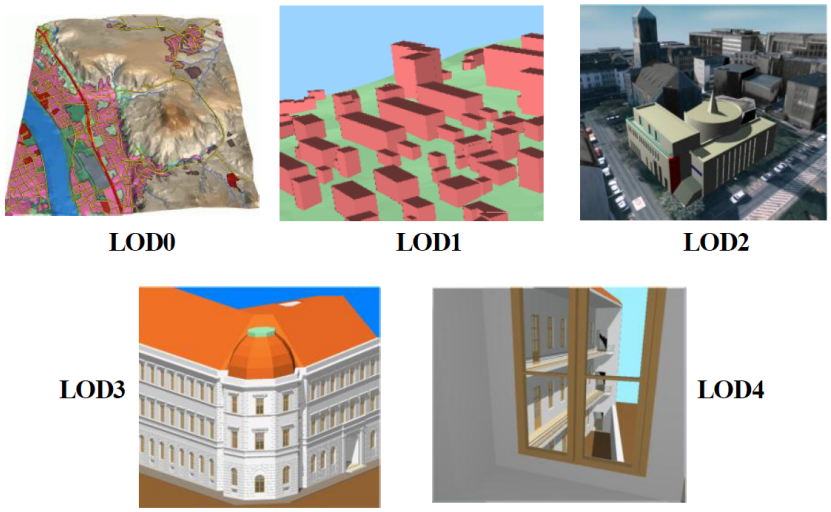


Figure 3.1: CityGML 2.0 levels of details as presented from the specification [15].

The *Spatial model* is the fragment of the specification responsible for the geometrical aspects of the model. It is grounded on the GML geographical language and provides the foundations for 3D representations of city objects; each CityGML object must have a representation in the spatial model since it is key for visualization and geographic applications. Figure 3.2 shows the main features.

The *Appearance module* is the component of the specification responsible for giving city objects the intended visual look. Every city object can have an appearance, even though it is not mandatory to have one. It plays a primary role in scenarios related to urban planning and gaming.

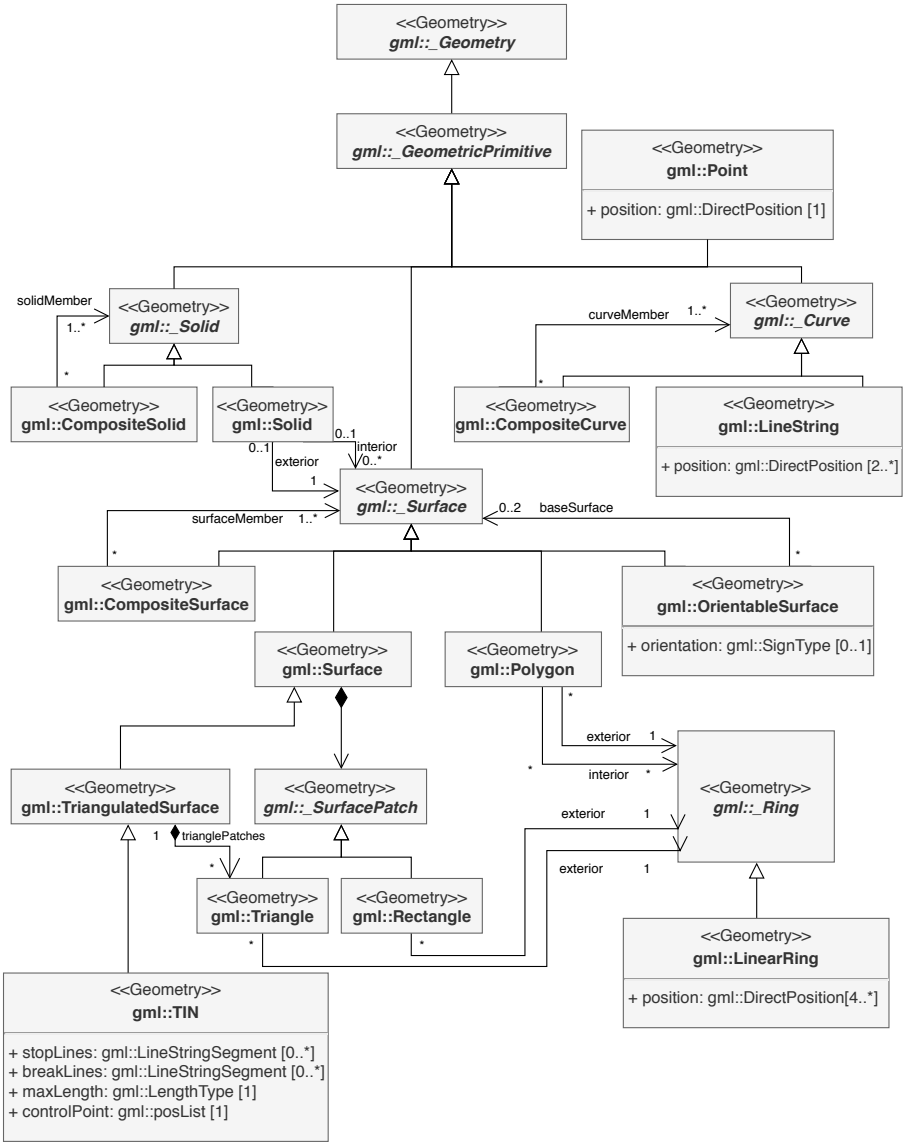


Figure 3.2: The main geometrical features of the CityGML spatial model.

Lastly, the *Thematic model* is the most significant part of the specification. It binds together the spatial description and the appearance and gives them the semantical meaning of city objects. It has a modularized approach, in which the Core has the fundamental features, which are then extended by modules scoped on specific aspects of the environment (like the Building module, the Transportation module, the Vegetation module, etc.).

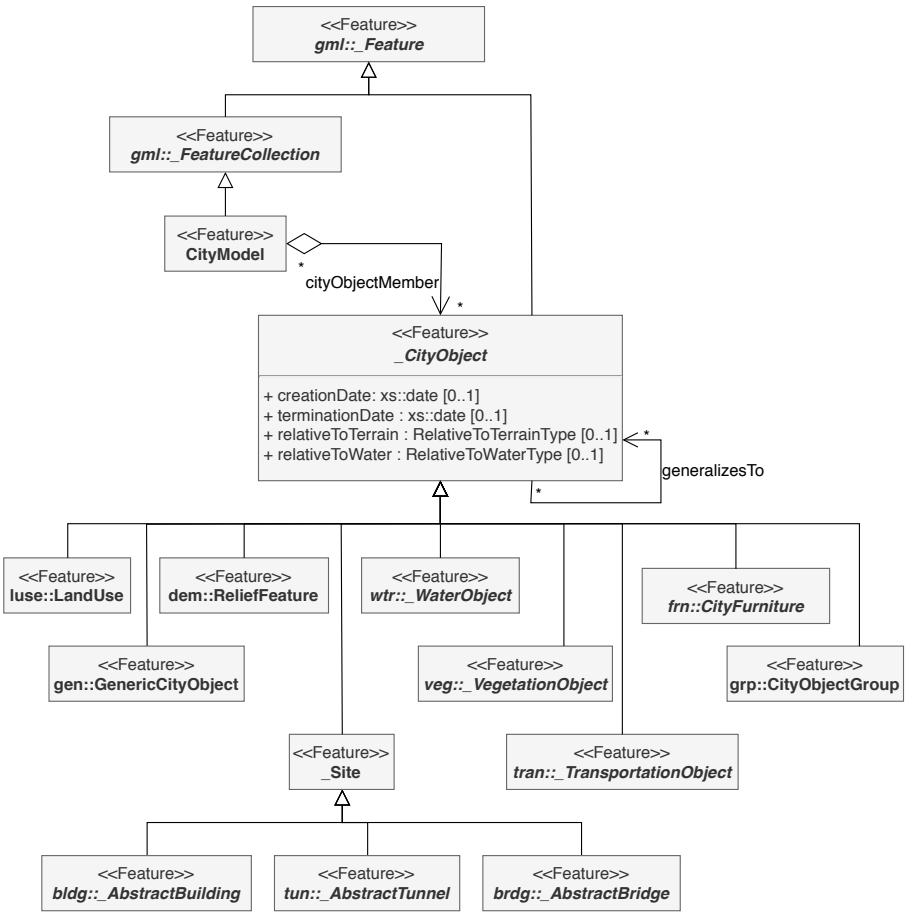


Figure 3.3: CityGML 2.0 Core features addressed by our framework.

3.1.3 *Extending the CityGML standard*

The final aspect of CityGML that we want to point out is the flexibility it introduces, by providing a way of defining an Application Domain Extension (ADE), in which application requirements related to the city models can be described, while the enriched models still comply to the specification [11]. ADEs are formally defined extensions, specified in XML Schema Definition (XSD) or Unified Modeling Language (UML), capable both of adding new properties to existing CityGML classes and of adding entirely new classes and data types. For example, an ADE

can be a set of extra attributes and elements nested into a standard CityGML model, to extend the capability of CityGML buildings to fully support Building Information Modeling descriptors. This also includes adding extra elements within the ADE, which reference standard CityGML objects and describe new relationships among them. These extensions can be arbitrary, ranging from geometrical aspects like shadow orientation to process-specific, like historical priority. More than 40 ADEs have been developed so far, with purposes including noise propagation, energy distribution, spatial topology and time variation among the others. Table 3.1 shows a summary of known ADEs defined in XSD. Refer to [11] for an extensive, yet non-exhaustive presentation.

	ADE	**Purpose**	**Origin**
1	Energy ADE	Application	Europe
2	Energy Efficiency ADE	Application	Italy
3	Noise ADE	Application	Germany
4	Road Traffic Noise ADE	Application	India
5	Robotics ADE	Application	Japan
6	UtilityNetworkADE	Application	Germany
7	CAFM ADE	Application	Germany
8	Immovable Property Taxation ADE	Application	Turkey
9	Cultural Heritage ADE	Application	Italy
10	Indoor ADE	Application	Korea
11	IMGeo ADE	Generic	Netherlands
12	INSPIRE ADE	Generic	Germany
13	ACRoof ADE	Generic	China
14	CityGML iTINs ADE	Generic	Netherlands
15	GeoBIM	Generic	Netherlands
16	3D Metadata ADE	Generic	Netherlands

Table 3.1: Known ADEs currently defined in XSD.

3.1.4 *CityGML role in our framework*

Despite being valuable sources of information, CityGML models' volume and domain-oriented design, make it challenging to consider them as a data source for complex analysis and operation, requiring huge application-specific pre-processing and post-processing. Our technical framework has been designed under the idea of automatically migrating both changes to the standard CityGML thematic features, shown in Figure 3.3, and a given ADE.

To the best of our knowledge, despite the many CityGML ADEs available neither tools nor data are readily accessible for any of them as of today and, therefore, a pre-processing step is still needed to prepare the source information describing application-specific relationships, by referencing objects of the original city model. In the next section, to simplify the discussion, we will assume the `key()` and `children()` functions are properly defined to providing a unique identifier of the CityGML feature and retrieving a list of sub-features respectively.

3.2 CYBER-PHYSICAL SPACES AS TARGET MODELS

3.2.1 *Key characteristics of cyber-physical spaces*

The analyzable models that we target are formally modeled topological structures specifically aimed at cyber-physical systems, termed CPSp [48, 49]. CPSps are composite environments characterized by the coexistence of physical and computational entities. They are a powerful tool for expressing requirements and dynamics of modern cyber-physical systems. The advantage of these structures relies in the way they have been established in the literature: not only they might have different representation, depending on the context and the goal of the applications, but in all those cases they make use of formally defined representations, which allow to guarantee or prove the properties of interest for the specific settings.

We opted for this generic graph-based target model because of (i) its flexibility and applicability to various types of analyses and (ii) its formal semantics, allowing for a precise definition of the correctness of a transformation.

3.2.2 *Cyber-physical spaces as bigraphs*

Among the different formal semantics commonly used to describe cyber-physical spaces, we focused on a process meta-calculus called bigraphs [32], which, in our view, provide a solid albeit easily understandable representation of them. The main idea of bigraphs is pretty simple: they can be described as the superimposition of two graphs (an hypergraph and a forest) defined over the same nodes; one describing some kind of containment relationship among nodes and the other describing any sort of links among them. Moreover, they provide a very convenient representation in terms of diagrams, like the one in Figure 3.4.

More formally, a bigraph is a 5-tuple:
$(V, E, ctrl, prnt, link) : \langle h, X \rangle \to \langle k, Y \rangle$

where V is a set of nodes, E is a set of edges, $ctrl$ is the control map that assigns controls to nodes, $prnt$ is the parent map that defines the

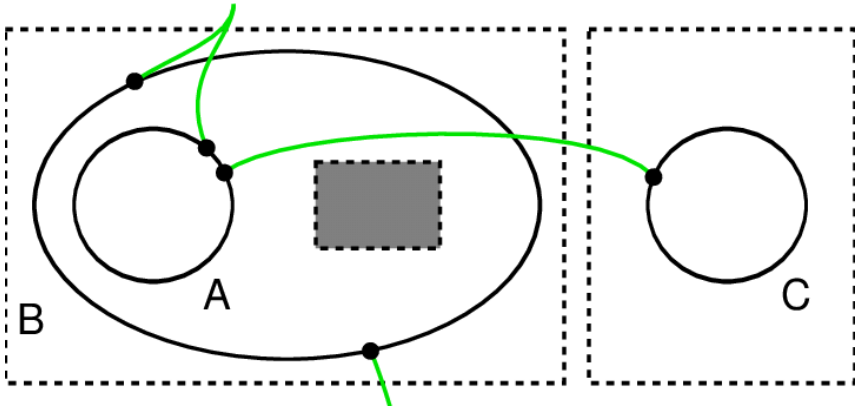


Figure 3.4: An example of a visual representation of a bigraph from [32]. Controls (A,B,C) help to identify nodes (represented here as circles). Dotted squares represent interfaces for connecting with other graphs and the green arcs represent the links.

containment of nodes, and *link* is the link map that defines the link structure. The notation $\langle h, X \rangle \to \langle k, Y \rangle$ indicates that the bigraph has h holes (sites) with a set of inner names X, and k regions with a set of outer names Y; the first couple defines the *inner* interface of the bigraph while the second one defines the *outer* interface.

For a more detailed look of their definitions, and proofs, and particularly for the category-theoretical analysis, the interested reader may refer to the vast body of literature on the topic [32].

3.2.3 *Bigraphs in our framework*

To simplify the analysis, we ignore some aspects of bigraphs, namely controls and interfaces (i.e. sites, names and regions), since they are not crucial to our context, despite they could be object of future work, in order to develop some refinement tools in designing constraints over city models. Scoped to our framework, bigraphs can be described in terms of the following components:

- A set of *labelled nodes* $\mathtt{v} \in \mathtt{V}$ which represent the elementary objects of the environment, labelled with a pair $(\mathtt{identifier}, \mathtt{type})$.
- A *place graph* as a tree: given the structure of CityGML models, we can simplify the discussion, considering that the containment relation develops from a single root representing the `CityModel` and, thus, the theoretical forest degenerates to a tree.
- A *link graph* is a hypergraph defined over the same set of nodes. Hyper-edges link any number of nodes; this graph represents generic links (i.e. many-to-many relationships) among nodes. Place and link graphs are orthogonal, and edges between nodes can cross locality boundaries.

In subsequent chapters of this work, we will assume that a `key(v)` function returning the label of a node is properly defined. In additon, we suppose that `findNode(k, S)` is a function that returns a node v from the set `S` labelled with `k`, while in a similar fashion, `findLinks(k, S)` will allow us to find the links connecting a given node. Ultimately, we will refer to `child(n)` for a node that has n as a parent in the containment relationship.

3.2.4 *Fitness of bigraphs as target analyzable models*

We advocate that bigraphs suit our analysis and synchronization goals particularly well; they allow us to achieve the level of expressiveness needed by key topological characteristics while keeping a high level of flexibility: the place graph defines a hierarchical structure, allowing us to model the locality in space of the city objects in terms of topological nesting, while the link graph can represent arbitrary connections among nodes (i.e. some other topological relation), enabling the representation of application-specific relations. Moreover, the key strength for analysis of bigraphs relies in the Bigraphical Reactive System (BRS), which extends the static bigraph description with formal semantics for expressing dynamic behaviour via reaction rules. This way, the user of the model can reason over possible future topological configurations of the cyber and physical spaces that are reachable from the current one, yielding a branching structure, just like in graph transformation systems. For more details about the connections between BRS and graph transformation systems, one can refer to [22].

3.3 ACHIEVING SYNCHRONIZATION

Before getting into the details of the bidirectional transformations we defined between the models presented in Section 3.1 and Section 3.2, we want to stress out the key features we would like to achieve, as well as the main problems that emerge even before addressing the issue.

3.3.1 *Design Goals*

In our view, model-based engineering of cyber-physical space-dependent systems should convey the following design principles, which underlie our design of a bidirectional transformation between the two models:

1. Interoperability with well-established domain-specific standards and data models, namely GIS and BIM as used in practice;
2. Provision of an actionable representation of the model in a non-domain-specific language that can enable complex analysis, in our case cyber-physical space reasoning;

3. Automatic composition of changed and unchanged parts of the model in a suitable way (i.e. well-behaved transformations), highly pertinent to both support design activities and runtime model operations;

4. Decoupling of independent levels of reasoning (e.g. topological from geometrical) whenever possible, since those can be considered as being on different levels of abstraction.

3.3.2 *Design Challenges*

When dealing with the problem of establishing a synchronization between an *abstract* model and a concrete, or *reified*, one, while the *abstraction* process can expose some issues, it is the *reification* one that brings the greatest difficulties. To get the intuition of this, consider that we have on one side a highly detailed model of a city, with buildings, roads, traffic lights, and each of these has details about its shape, its height, its position, etc. On the other side, however, we have a set of nodes and edges, only representing the minimal semantic meaning we are interested in (for example we have 5 nodes representing buildings, 2 representing roads, and a couple of edges connecting them to express the proximity of buildings to a given road). Now, a fully automated abstraction program can easily be developed, as it just has to throw away irrelevant details. On the contrary, reifying a node and its edges into a 3D model coherent with the city is not trivial at all. All sort of problems may arise (e.g. what color should it be? How tall? How many doors? etc.), some of which may even be meaningless in the specific context of the application. It is therefore very clear that the reification problem, also called *putback* process in bidirectional programming context, deserves a special attention, since it requires *new* information to be generated in order to fill missing details and produce a meaningful and consistent result in terms of practitioners' knowledge.

In the following, we illustrate how the above challenges may be tackled by designing and implementing a consistent and well-behaved bidirectional transformation (BX) between source city models and analyzable models which, by design, properly propagates changes when either one of the models is modified.

4 BIDIRECTIONAL TRANSFORMATIONS DEFINITION

To address the problem of migrating information from one representation to another, we need a clear idea of *what is what*, in the sense that we need to clearly define which parts of an object of the source representation have a corresponding equivalent in an object of the target representation. In the mathematical context, this kind of law is referred as an equivalence relation among two objects. Bidirectional Transformations have established as a theory willing to achieve exactly that goal: stating when (and to which extent) objects of the *source* are equivalent to objects of the *view*. The most intuitive name by which this equivalence problem is referred in the literature, takes the name of "*consistency relation*", among two (or more) sources of information [24].

This chapter focuses on this fundamental issue: we start by specifying the consistency relation among our models of interest in Section 4.1, then we move to defining the routines to enforce such specification in Section 4.2, and lastly we discuss the theoretical implications (and limitations) of our approach in Section 4.3. We defer implementation details of the procedures shown here to Chapter 5.

4.1 CONSISTENCY SPECIFICATION

4.1.1 *Consistent Model Transformations*

Bidirectional transformations BX is a development theory for maintaining the consistency relation between models. A general methodology to deal with that, when restricting the attention to pairs of models, is the one of asymmetric lenses [24]. Asymmetric lenses, in practice, consist of a pair of transformations *get* and *put* [20]. The *forward* transformation $get(s)$ is used to produce a target view v from a source s, while the *putback* transformation $put(s, v)$ is used to reflect updates from the view v to the source s. We call them asymmetric because the role of the two models is very different. We say that on one side,a model which has more information than the other is our *source*, while on the other side we have just a subset of the information, which we call *view*. A lot of the studies on Bidirectional Transformations have been centered on defining a clear set of properties that these two transformations should (or should not) have, in order to make sure the transformation satisfies some degree of correctness. The key property of our interest is the *well-behavedness*, which can be defined as follows.

Definition 4.1.1. An asymmetric lens (*get*,*put*) is said to be *well-behaved* if the following round-tripping laws hold:

$$\begin{aligned} put(s, get(s)) &= s \quad \textsc{GetPut} \\ get(put(s, v)) &= v \quad \textsc{PutGet} \end{aligned}$$

The idea of Definition 4.1.1 is not new in Computer Science and in fact it resembles, in a sense, the ideas of context-freeness in formal language theory, of stationarity in systems theory and of the Markov property in stochastic processes.

More precisely, the GetPut property requires that no change of the view shall be reflected to no change of the source, while the PutGet property requires all changes in the view must be completely reflected to the source so that the changed view can be recomputed -exactly identical- by applying the forward transformation to the updated source. As the reader might have noted, in both cases we require that neither the context nor the time of the transformation, will have any effect on the result of the transformation.

4.1.2 *Defining City-Graph Equivalence*

Put in context of the models we investigate, the consistency relation can be formally specified as follows.

Definition 4.1.2. Given s, s' elements of the CityGML model, given r relationships defined in the CityGML ADE of interest and v, v' nodes of the bigraph, we can say that s and v, where $key(s) = key(v)$, are *synchronized* ($s \rightleftharpoons v$) if and only if, the following conditions hold:

A.1 $instanceOf(s, \_CityObject) \wedge instanceOf(v, Node)$

A.2 $isContained(v, pv) \rightarrow childOf(s, ps) \wedge ps \rightleftharpoons pv$

A.3 $isLinked(v, v') \rightarrow holds(r, s, s') \wedge s' \rightleftharpoons v'$

Statement **A.1** prescribes the types of the objects of the transformation, this is checked by the predicates *istanceOf* predicate, where *_CityObject* refers exactly to the abstract class presented by CityGML specification (check Section 3.1 for details), while *Node* represents Bigraph nodes as described in Section 3.2. Statement **A.2** requires that to every node v which is situated in the place graph (*isContained*) of a node pv, corresponds a CityGML object s which is a direct child (*childOf*) of a parent node ps, which was previously synchronized to pv. Lastly, Statement **A.3** defines the connection between the link graph and the CityGML ADE: it states that every link (*isLinked*) from

v to a node v' that was previously synchronized with s', must have a corresponding relationship r that *holds* between s and s', supposing some application-specific meaning is somehow in place as well.

To help analyze the routines for checking consistency, we define now two auxiliary definitions which will prove useful later on.

Definition 4.1.3. A source model S is *place-consistent* with respect to a view model V if both **A.1** and **A.2** are satisfied.

In the same fashion, we can say that

Definition 4.1.4. A source model S is *link-consistent* with respect to a view model V if both **A.1** and **A.3** are satisfied.

And to summarize the Definitions 4.1.3, 4.1.4 we say that

Definition 4.1.5. A source model S is *consistent* with respect to a view model V if it is *place-consistent* and *link-consistent* at the same time (i.e. the models are synchronized).

By a closer look to the previous discussion, the reader might realize that the place-consistency has been fully formalized and, therefore, it can always be checked without ambiguity. On the other hand, link-consistency relies on the informal *holds* predicate, which cannot be in principle solved unambiguously, since it is application-specific. This not-completely formalized approach is not new in BX, since, in some cases, local correctness checks (sometimes also called black-box operations) are needed in order to achieve consistency [45]. More information about the implications of the level of formality of our approach will be provided in Section 4.3.

4.2 CONSISTENCY ENFORCEMENT

4.2.1 *Preliminary analysis*

We will now see how to make sure these conditions hold. To do that, Algorithms 1 and 2 give the high-level idea of the key things to verify from Definition 4.1.2. Details about unmentioned aspects need less theoretic considerations and will therefore be directly presented together with their implementation in Chapter 5.

The three conditions described are enforced by our framework in the same order as presented. To make sure Statement **A.1** holds, we simply rely on the Type checking system of the language we used. If the types of the objects provided are wrong, a run-time error prevents the execution of the transformation. If the types are right, we move on to checking (and repairing) the *place-consistency* and lastly we repair the *link-consistency*. We now present the algorithms to do that on the *putback* perspective which, given a CityGML source element $s \in S$ and a node $v \in V$ of the bigraph, return an updated version of the CItyGML source element s'.

4.2.2 *Enforcing Place-Consistency*

Algorithm 1 performs the first sweep of the synchronization logic, which starts from the root of the city model and from the outermost node of the bigraph, traverses the two structures at the same time and repairs the differences by adding source elements when they are missing in the source or removing them when not in the view. Then the procedure is mapped to the children. At the end of the execution, the source model elements will be at the position corresponding to the view nodes, which means the models will be *place-consistent* (i.e. conditions **A.1** and **A.2** must hold).

Algorithm 1 Containment Graph Syncing

```
function syncCont(s :: _CityObject, v :: Node)
  for all o ∈ children(s) do
    if key(o) ∉ children(v) then
        REMOVE(s, o)
    else
        syncCont(o, findNode(key(o), children(v)))
    end if
  end for
  for all n ∈ children(v) do
    if key(n) ∉ children(s) then
        ADD(s, n)
    end if
  end for
  return s
end function
```

4.2.3 *Enforcing Link-Consistency*

Algorithm 2 represents the second sweep of the synchronization. It starts from the assumption that the models are already *place-consistent*. From the root of the city model and the outermost node of the bigraph, it checks whether there is correspondence between CityGML ADE relationships and bigraphs links or not. When a mismatch is found, a repairing procedure is triggered which attempts to repair it based on the set of links that are present in the view. Note also that the fact that the same relationships exist in the source is not sufficient, as they must also be relating the same elements referenced by view nodes. The same logic is mapped to the children nodes, until the source model is link-consistent (i.e. **A.3** holds).

Algorithm 2 Link Graph Syncing

```
function syncLinks(s :: _CityObject, v :: Node)
  rs ≔ getRelsWith(s)
  ls ≔ getLinksWith(v)
  for all l ∈ ls do
    if key(l) ∈ rs then
        rel ≔ findRel(key(l), ls)
        for all n ∈ nodes(l) do
            if key(n) ∉ rel then
                UPDATE(n, ls)
                break
            end if
        end for
    else
        UPDATE(n, ls)
    end if
  end for
  for all r ∈ rs do
    if key(r) ∈ ls then
        link ≔ findLink(key(r), ls)
        for all o ∈ objects(r) do
            if key(o) ∉ link then
                UPDATE(n, ls)
                break
            end if
        end for
    end if
  end for
  repeat syncLink() on children
  return s
end function
```

Lastly, `ADD`, `REMOVE`, `UPDATE` procedures in Algorithms 1 and 2, represent a simplified version of the *Application Policy* actions, which play an important role in the transformation since they provide the application-specific meaning to the repairing procedures. We will now analyze them more in detail, but the goal here was to just highlight the idea behind the transformation. For implementation details and in-depth looks at the interfaces Application Programming Interface (API), the interested reader can refer to Chapter 5 and the accompanying artifact.

4.3 DEALING WITH DOMAIN-SPECIFICS

4.3.1 *Consistency Design Issues*

We designed Algorithms 1,2 to check that all the elements of the two models are in the right place and that they satisfy the consistency conditions of Section 4.1. However, two important aspects have been disregarded and need more considerations. Firstly, in condition **A.3** we used the predicate *hold* which has not been formalized explicitly. Moreover, `ADD`, `REMOVE`, `UPDATE` operations have been used in Section 4.2 without any description about how they work.

About the *holds* predicate, the reason of this informal approach is actually quite simple: the problem is that stating that a CityGML ADE relationship between two CityGML has a twofold meaning. From the syntactic point of view, it means that an XML node of the right type, referencing the right elements is present; this can be formalized and can be (and in fact it) easily checked automatically. From the semantical point of view, however, since CityGML ADEs' role is usually to provide connections with external data or information, there will be application-specific requirements which are, in general, not known. Concerning the `ADD`, `REMOVE`, `UPDATE` operations, we stumble in a related issue: when reifying view changes in the source, we must generate some extra information in order to end up with a correct model of the source space. This information, will likely affect domain experts' decisions regarding other aspects of their design, and it is therefore advisable that the reification strategy for new or removed objects can be influenced by developers interacting with our framework, depending both on the purpose of the specific object and on application scope and requirements.

4.3.2 *Application Policy*

We envisioned an *Application Policy* which is the component ultimately appointed for verifying that task. We assume that depending on the development context and application goals, developers might need different policies, in order to exploit various insight they might have

about the objective of the transformation. We defined a small set of clearly scoped interfaces of the Application Policy, which we called *actions*. Each *action* can access some restricted information that can be useful in order to achieve its goal. It is important to note that the interface of the action is strict, which means they are required to produce an output that does not break the previous assumptions on the type and on the structure.

The following actions have been defined:

- `ADD(s :: _CityObject, v :: Node) :: _CityObject`

  which is bound to generate missing objects of the source. To that extent, it has access to all the information available from the parent of the target object. It is also allowed to change its actual representation (this is needed in some applications such as keeping spatio-semantic coherence) and it must return a new child having the key and type provided by the respective bigraph node.

- `REMOVE(s :: _CityObject, v :: Node) :: _CityObject,`

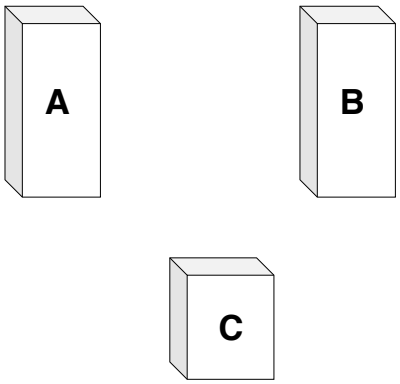
  which symmetrically to `ADD` has the purpose of removing extra objects from the source. It has access to the same information with the same constraints, albeit in this case it only returns the updated representation of the parent.

- `UPDATE(s :: _CityObject, v :: Node) :: _CityObject`

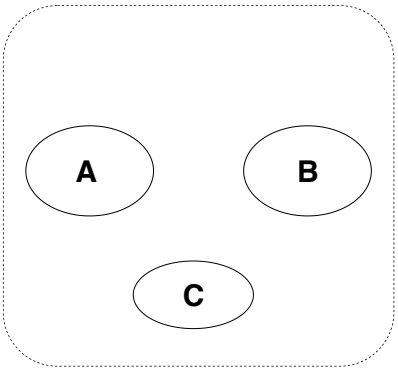
  is the most general action, responsible for both updating ADE relationships and potentially changing the representation of the current object. The problem of correctly reflecting a set of links may be very hard to solve in general. For this reason, our framework makes two simplifying hypotheses. Firstly, we assume that a change in a relationship (or the definition of a new one) can be fully expressed in terms of separated updates to the objects corresponding to the different nodes of a link. Moreover, we assume that the information required to address this task is limited to the subgraph of nodes and links related to the current one.

We end this chapter with some consideration about the `UPDATE` action and the reasons that support its design. The general idea of *Application Policy* actions is to reformulate a problem originating in the original domain space (i.e. a change to do at the city) into a more compact form where only a subspace of interest is taken into account (only the building raising the problem and the ones *"nearby"* it). This sub-problem is not necessarily simpler than the one in the original space, but it is defined on much less elements (depending on the number of links of the current node).

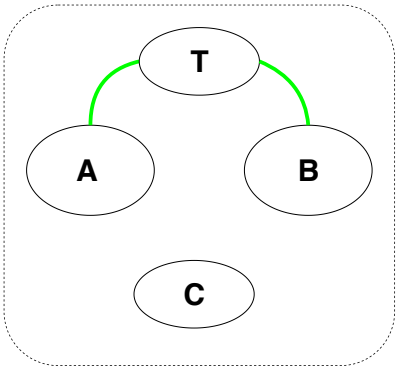
To get a concrete understanding of the issue, imagine a setting like the one in Figure 4.1: let's say we want to add a tunnel that connects buildings A and B. Figure 4.1c shows the bigraph expressing this requirement. To reflect the changes back to the city model, let's say we start iterate the putback procedure starting from the building A; there are clearly many alternative solutions, some of them are shown in Figure 4.2.



(a) A simple city with three buildings.



(b) The corresponding view generated.



(c) The updated view.

Figure 4.1: A typical problematic case. There are three buildings in our city (a), and we start to analyze them without any extra relationships - no links in view (b). Then we decide we want to connect A and B with a tunnel T (c).

By looking at the picture, one can see that there could be theoretically multiple (actually infinite) solutions to the problem. Our `UPDATE` action would only allow solution (a), because solutions (b) and (c) would require to change the shape or the position of nodes different from the current one, which are two illegal operations for our Application Policy. In the most degenerate case, a solution may even result in a completely different configuration of the city space, where for example the position of every object in the city is changed in order to satisfy a requirement. In such cases it would become meaningless to synchronize the two models because, as it is even harder to say

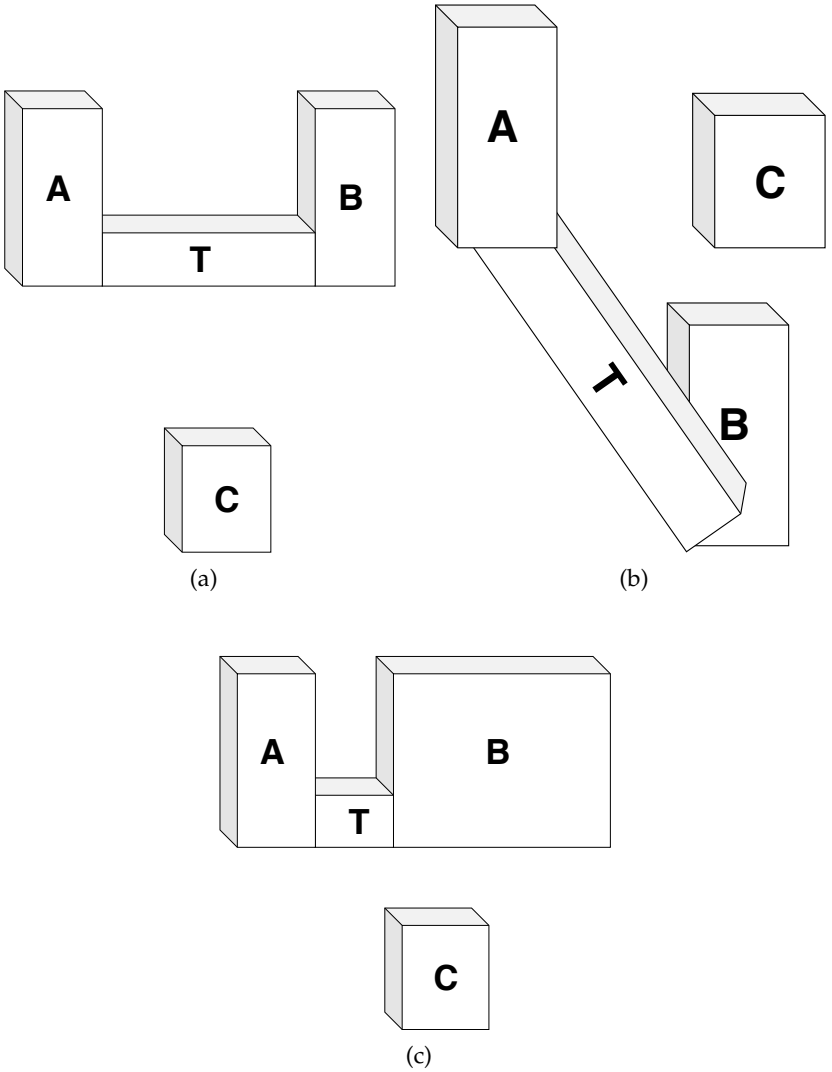


Figure 4.2: Despite all these solutions are formally correct, only solution (a) is reasonable, since it does not require to move other buildings or to change their shapes.

that we are talking about the same model as the beginning. It must be noted that transformations like case (b) and (c) might indeed be useful in some cases. However, we did not find, to date, any suitable case scenarios for them and the limiting factor of this assumption is still a matter of active investigation.

5

TOPOCITY BX FRAMEWORK

After having presented the models of our interest in Chapter 3 and the laws over which we defined a consistency relation among them in Chapter 4, it is time now to see the details of Topocity, the BX framework we implemented to reflect the transformation we defined earlier.

We thought a powerful choice for the language of the framework was the Haskell language. There are two main reasons supporting this choice: firstly, the functional approach seemed well-suited to deal with transformation, which is indeed defined in terms of functions. Moreover, it felt more natural to integrate with the BiGUL [27] library, which is a putback-based BX language developed as an Haskell Domain-specific Language (DSL). The primary strength of BiGUL is that, in contrast with other BX techniques, it is designed to automatically derive the *get* direction of the transformation, given the *put* code. This means that developers have to implement just the backward/putback transformation from the view to the source, and the forward one comes for free. In this Chapter, we will analyze the different aspects of our framework, starting from a broad picture of both internal and external components of Topocity in Section 5.1, then we will have a look at some key points of the implementation in Section 5.2 and lastly, we will see how to use the framework in practice in Section 5.3. Note that Topocity is freely available online and is still under active development [1].

5.1 TOPOCITY ARCHITECTURE

Topocity's main components are shown in Figure 5.1; its modular design allows for external component development and integration.

The functioning of Topocity revolves around two models, a source CityGML description as input, and a bigraph view representing the Cyber-physical Space as output. We will now present the different components of Topocity, by breaking it down to the functional layers that are traversed in getting from one model to the other, as show in Figure 5.1. It should already be apparent, by looking at the picture, that the modular design allows for easy changes to support external components and integrations.

- *HXT* is the most popular library for handling XML files in Haskell. Its abstract approach allows to choose from different

1 For more up to date information about Topocity, refer to https://topo.city

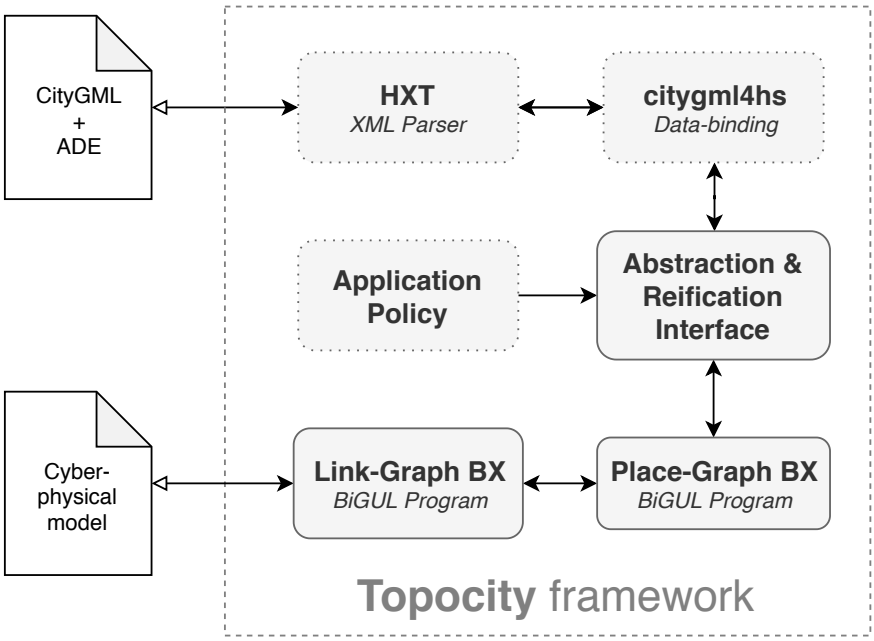


Figure 5.1: Combined view of architecture and dataflow of TOPOCITY. Dotted boxes represent external components.

parsers depending on the context and to define data-binders from/to XML nodes. We used it to to perform basic I/O loading, storing and parsing of the CityGML files, as well as for the underlying engine of the data-binding library.

- *citygml4hs* is an independent project maintained by the authors of this work. The idea is to replicate the citygml4j library released by some of the people working at the CityGML specification. Its goal is to provide a full semantic typed data structure, along with basic helper functions like the *key* method described in previous chapters. While this component is not crucial in terms of the strict requirements of the transformation, it plays an essential role in terms of non-functional requirements, since it provides a more actionable representation easily exploitable by third-party software.

- The *Abstraction & Reification interface* is the software component delegated to deliver a common representation of citygml4hs types. The underlying idea is that instead of building a separated BX for each type of the source, it makes algebraic transformations to generate a more abstract representation starting from the source, which can be easily converted back to the original format.

- *Place-Graph BX* is the first sweep of the synchronization process. It implements Algorithm 1 in BiGUL, i.e. it synchronizes the nodes and the place graph with the source.

- *Link-Graph BX* is the second sweep of the synchronization process, and it makes use of BiGUL primitives as well to implement Algorithm 2. In contract with the previous algorithm, it synchronizes only the link graph with the source. However, this process could be more complex than the previous one, as a graph might be a lot convoluted, and therefore it might have to reiterate many times over the same node, in order to check all the subgraphs containing that node.
- *Application Policy* is the primary container of framework API actions. The main ideas have already been presented in Section 4.3, the interested reader might refer to the project website for more information.

5.2 KEY IMPLEMENTATION POINTS

5.2.1 *Data Types*

Before looking at the code of the transformation, it is interesting to discuss a little about the data types over which it is defined. For starters, the source data structure is supplied by the *citygml4hs* library, in particular the CityGML `_CityObject` type is defined as follows:

```
data CityObjectMember      =      Site Site
                           |      Veg VegetationObject
                           |      Gen GenericCityObject
                           |      Wtr WaterObject
                           |      Tran TransportationObject
                           |      Dem ReliefFeature
                           ...
deriving (Read, Show, Eq,  Data,  Generic, Identifiable)
```

Listing 5.1: `citygml4hs` core implementation of `_CityObject`

Notice that, not only it uses a different type for each kind of city objects (allowing for strategies where only some types may be filtered). The important addition of the library is the `Identifiable` class, which provides a unique ID for the object (if possible) by considering the identifier hierarchy defined in the CityGML specification.

However, to reduce the complexity of the data structure to be transformed, the Abstraction interface transforms them into the following, more practical data structure:

```
type AbsCity      =  (NTree AbsCityNode, [AbsRelation])
type AbsCityNode  =  (UID, (Type, Data))
type AbsRelation  =  (UID, (Type, [AbsCityNode]))
```

Listing 5.2: ADT of city data

`UID`, `Type` and `Data` are simply some special strings representing the identifier, the object type and the internal data of each object respectively, while `NTree` is just an n-ary ordered tree data structure, commonly used in Haskell. As it should be apparent from the types, the idea of this intermediate data structure is just to highlight the predominant features of interest, in a structure of pairs of the kind $(head, tail)$, which is a quite convenient data structure for subsequent processing.

Finally, the output data structure representing the Bigraph is shown now. Note that this is has been conventionally decided by the authors, since, at the time of writing, no standard representation has been already defined for this special kind of graphs.

```
type BiGraph      =  (PlaceGraph, LinkGraph)
type LinkGraph    =  [BiGraphEdge]
type PlaceGraph   =  NTree BiGraphNode

type BiGraphNode  =  (UID, Type)
type BiGraphEdge  =  (UID, (Type, [BiGraphNode]))
```

Listing 5.3: Bigraph ADT

The similarities between this and the previous definition should be quite evident. This structure is equivalent to the previous one, except for having dropped the `Data` field. That is in fact the primary difference between the two structures, since the bigraph is just some sort of *projection* that only selects a subset of the information. In principle, however, other difference could be introduced, for example city object types and bigraph nodes' types might be using different datatypes.

5.2.2 *BiGUL Programs*

We have seen that the source and the view can be represented in a similar way without any irreversible (i.e. lossy) transformation. Our objective now becomes to migrate the information from one part to the other while keeping the correspondence as defined in Sections 4.1,4.2.

Concerning Condition **A.1**, we defined the following equivalence relation:

```
equiv             ::  AbsCityTree -> PlaceGraph -> Bool
equiv a b         =   check1 (tKey a == tKey b)

tKey              ::  NTree (UID, a) -> UID
tKey (NTree d _)  =   key d
```

Listing 5.4: Equivalence relation between `s` and `v`

The `check1` function checks whether a predicate defined on two trees is true up to the first level of the tree, which in our case means that it checks whether the keys of both the root and the children of the two trees are the same. In this way we are sure that when the equivalence holds, we are at least talking about the same object.

A *Place-Graph BX*

This equivalence is directly exploited by the *Place-Graph BX*, which is encoded in this way:

```
syncTree :: BiGUL AbsCityTree PlaceGraph
syncTree = Case
   [ $(adaptive [|\s v -> not (equiv s v)|])
      ==> \s (NTree _ vcs) -> syncChildren s vcs
   , $(normal [| equiv |] [| noCond |])
      ==> $(update [p| NTree (i, (t, _)) cs |]
                   [p| NTree (i, t) cs |]
                   [d| i = Replace; t = Replace; cs = align |]
          )
   ]
```

Listing 5.5: *Place graph* BX code

The last piece of code may not be easy to understand even for experienced Haskell programmers. That is because it exploits the Template Haskell notation used by the BiGUL language. We will not analyze all the elements of the code here, but we will just give the general understanding. The `Case[a]` BiGUL operator resembles the `switch/case` construct of procedural languages: it takes a list of pairs of the kind $(pattern, operation)$. The idea is that if the function parameters match the pattern (defined by `$(•)`), then the operation after the `==>` is performed. Note the `normal` keyword: it is the equivalent of a procedural `case` having a `break` at the end, while `adaptive` corresponds to a `case` after which, if matched, the program continues checking against subsequent patterns in the list.

Put it simply, the idea of the `adaptive` branch is to verify that both the current element and the children have the correct ID. If it is not the case, then it means that there might be new or missing nodes in the view and therefore, the `syncChildren` function aligns the two lists by exploiting Application Policy actions to create/remove nodes in the source. Conversely, the `normal` branch is activated each time both the source tree children and the view tree children have the same id. When this happens, the source type is updated to the one of the view and the syncTree procedure is mapped to the children (which means each of them will execute it on its children, and so on recursively).

B *Link-Graph BX*

After having repaired the models in Listing 5.5 so that Conditions **A.1**,**A.2** hold, it is time to synchronize the last part of the bigraph (i.e. the links). The procedure is not dissimilar from the previous one:

```
syncGraph :: BiGUL AbsTopology AbsHypergraph
syncGraph = Case
    -- If the graphs are different, update source by using policy p
    [ $(adaptive [|\(NTree (_, ls) _) (NTree (_, ls') )
                  -> not (ls `equivLink` ls') |])
        ==> \s (NTree (_, vls) _) -> p s vls

    -- else, replace the data and map the algorithm to children
    , $(normal [|\(NTree (_, ls) _) (NTree (_, ls') _)
                  -> ls `equivLink` ls' |]
               [| noCond |])
        ==> $(update [p| NTree ( (i, (t, _)) , ls) c |]
                     [p| NTree ( (i, t ) , ls) c     |]
                     [d| i = Replace; t = Replace;
                     ls = align2; c = (align p)      |]
             )
    ]
```

Listing 5.6: *Link graph* BX code

To understand how `syncGraph` works we must first clarify why it is working on different data types than the ones presented in Listings 5.2,5.3. The reason is just efficiency: `AbsTopology` and `AbsHypergraph` are just alternative representations of `[AbsRelation]` and `LinkGraph` respectively, which are indexed on nodes, instead of relations/edges. The reason for this choice is the assumption that it is more convenient to loop over nodes instead of edges, since, after all, edges might not have a physical representation, but can indeed affect the way the nodes are represented as city objects.

The logic of the code is structurally equivalent to the one of Listing 5.5. The only difference is that we now are not comparing nodes but links defined on them, and take action when a mismatch is found. The policy action, here represented as `p`, corresponds to the `UPDATE` action presented in Section 4.3. The important thing to note is that it is the most powerful action, since it is not activated on a single object, but it takes as input a sub-graph. This graph is composed of the current node and the set of links (with corresponding nodes) defined on it, and has the goal of providing a representation of the corresponding city objects that satisfies the set of contraints the links define.

The procedure terminates with a CityGML model that is syntactically consistent with the current bigraph and semantically meaningful if a proper Application Policy is in place, so that also Condition A.3 is true. For more information of BiGUL specific operators like `Case` or `Replace`, and how it generated the *get* code from the *put* one, the interested reader might refer to [27] or related work.

5.3 PRACTICAL EXPLOITATION

Now that the most interesting details of the implementation have been unveiled, we show how to use the framework in practice. Note that, since the library `citygml4hs` is still at an early stage of development, source data might need to be pre-processed in order to be compliant with the data format supported by the library. We also report some publicly available data repositories so that users can choose the city model they are most interested in.

5.3.1 *Usage Guidelines*

Before running TOPOCITY, there are two software prerequisite:

- *Git*: a git client to pull the repository publicly available at `https://github.com/ennioVisco/topocity`.
- *Haskell Stack*, the cross-platform Haskell environment for compiling and running the framework.
- A CityGML source data model.
- A suitable *Application Policy*. Note that a trivial one is provided by the framework by default, even though it should not be used if not for basic demos.

To execute the transformation, one can follow these simple steps:

1. Loading the source model (which is the pair of a CityGML and CityGML ADE description) by calling e.g., `load(city.gml, ade.gml)`.
2. Generating a CPSp target model (i.e. perform the *get* transformation) by simply calling `get(source)`.
3. Generating an updated source model (i.e. perform the putback transformation) by calling `put(source, view)`.
4. Storing the new source model in a file by calling `store(filename.gml)`.

5.3.2 *Gathering Data*

At the time of writing, it is not easy to retrieve real data of cities in the CityGML format (note that for building models, Autodesk softwares alredy provide the possibility to export drawing objects as CityGML models). For this reason, we want to give some pointers of publicly available resources, so that the interested readers can have an effective basis to ground their work.

The most relevant online resources are:

- TU Delft has been developing CityGML models of some Dutch cities, available at https://3d.bk.tudelft.nl/opendata/3dfier/.
- TU Munchen has been developing a highly-detailed model of the city of New York, available at https://www.lrg.tum.de/gis/projekte/new-york-city-3d/.
- North Rhine-Westphalia has the biggest and most recent source to date of city models, for the major cities of the region, available at: https://www.opengeodata.nrw.de/produkte/geobasis/3d-gm/.

Note that CityGML is rising in popularity and, therefore, new data might be available from other sources. A frequently updated source is the CityGML Wiki [14].

6

EVALUATION: USE CASES

After having presented the most relevant details of the transformation, we believe it is time to get to the original problems we introduced in Chapter 2 and see some possible effects of the transformation on real-world case studies taken from the literature of the respective domains. The semi-formal approach we introduced can be beneficial, in a variety of scenarios, for engineering city-based systems where complex requirements and objectives can be pursued automatically. Among the case studies, two seemed particularly well-suited to show the advantages of our framework, and for this reason, we divide the rest of this chapter in the following way:

- *Facilitating Systems Design* – We devote Section 6.1 to exploring the impact on the design cycle. To this end, we consider the case of a civil engineering, willing to optimize the positioning of the construction facilities of a construction site, uses a TOPOCITY based software tool to automatically determine the best position for a tower crane. This problem falls in the category of construction site layout planning & optimization.

- *Facilitating Systems Operation* – The focus of Section 6.2, instead, is on considering how TOPOCITY can affect run-time decisions. For this goal, we consider the case where the source model is frequently updated through a monitoring process; the information these updates provide might be instrumental in performing a *planning* activity. However, planning decisions take place at a more abstract representation of the space, but the plan execution must be considered in terms of the real physical coordinates of the original model. A software based on TOPOCITY makes sure the gap between the abstract and the full representation is bridged, by consistently propagating changes on both directions. This problem is frequently addressed in the context of emergency response planning, in the context of smart cities.

Before moving on to the case studies, it is important to clarify the assumptions we made, to design a transformation that was also meaningful for people having domain-specific attention. Firstly, it must be noted that despite the CityGML models were real, they had no CityGML ADE available. For that reason, we sketched by hand some relationships ad-hoc for the goal of the case study, but they were by no means considerable a sufficient representation of real application data integrated in the model. Moreover, a proper use of TOPOCITY requires to have a suitable Application Policy in place that and takes decisions,

based on the context and on the goal of the specific application. In our case studies, our Application Policy is a trivial one, which just places a stub of a building at a fixed position; this is way incomplete and in no way usable in a realistic context. Lastly, the things that happen in the view-domain are clearly out of the scope of this work; for this reason we assume changes on the view are determined by some kind of *"automatic reasoner"*, but they are actually hand-made in our cases, just for the sake of showing the transformation on realistic changes.

6.1 TOWER CRANE POSITIONING

Proper optimization of construction sites layout is key to efficient construction activities. Before construction starts, site layout planning provides the necessary equipment and temporary facilities for the construction process, including allocation and dimensioning of elements like tower cranes, containers or storage areas. Decisions taken during this planning phase have direct impact on cost development and occupational safety on site during construction. Positioning of tower cranes is an important exemplar [3, 26]. Recent literature has provided techniques to automate the solution of this task, where two critical issues have been identified: (i) the lack of a simple but formal language capable of expressing rules, standards and best practices to check a building model [43], and (ii) the absence of tools able to perform this kind of operations by exploiting BIM/GIS descriptions like CityGML models, so that meaningful solutions can be found before implementation takes place [26]. In the following, we demonstrate how a flexible solution can be designed in which our framework plays a central role.

We consider an hypothetical construction site to be placed in a district of the city of Remscheid, North Rhine-Westphalia, Germany. For the real CityGML models we rely on North Rhine-Westphalia open data [34] – the linking structure related to tower crane positioning, is designed ad-hoc, since this step could be easily generalized and reproduced by modern user-guided CAD software [41]. Figure 6.1 shows the most relevant part of the model generated by our framework; an extra object and extra links are shown, corresponding to the changes made to the cyber-physical space in order to elicit the topological requirements for the new tower crane. Advanced analysis and model processing to generate such changes can take into account topological information in the analyzable model, such as proximity of construction site elements or complex relationships in the space layout, positioning the crane in a manner that satisfies some occupational safety or optimal placement requirements. As we are concerned with model transformations only, we consider such reasoning facilities as out of scope for this paper.

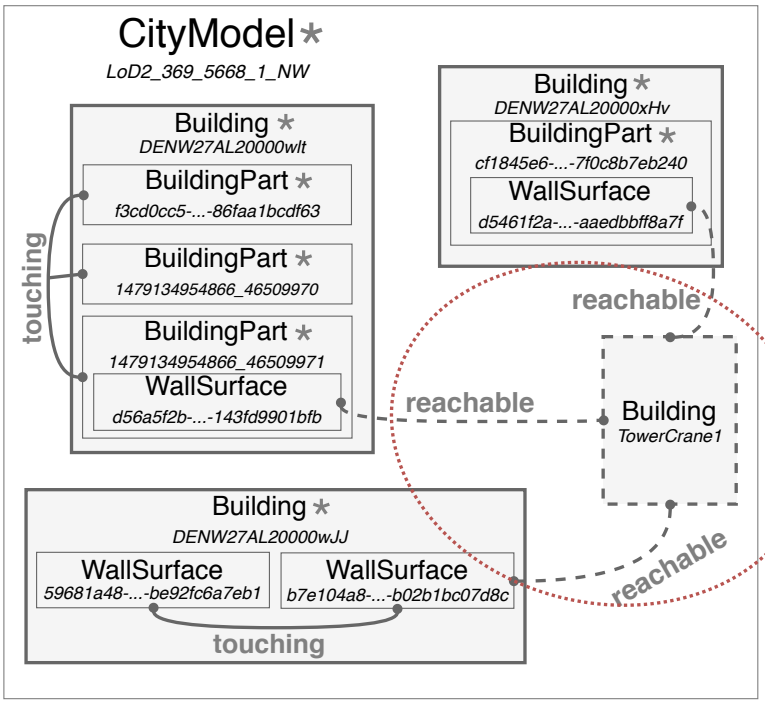
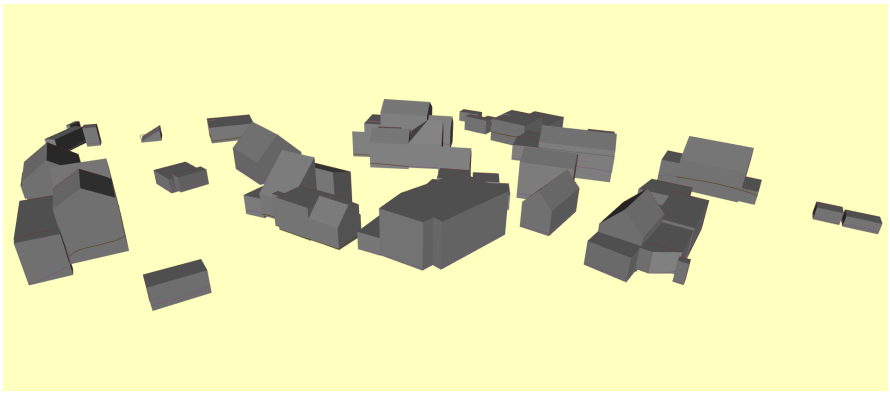


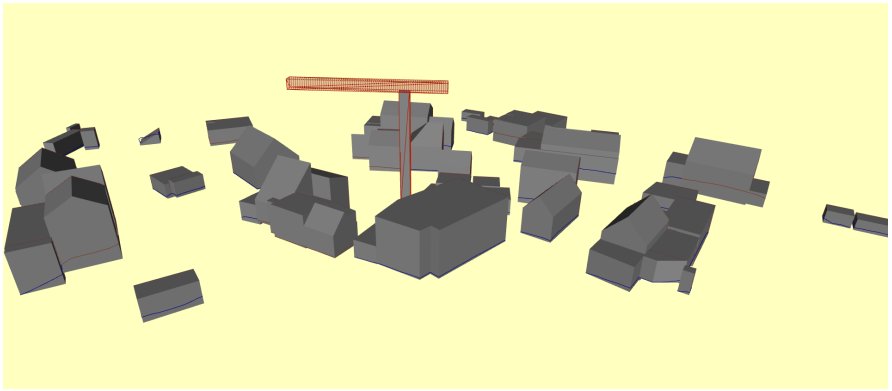
Figure 6.1: Fragment of the view model derived from the CityGML description of a district in Remscheid. Nodes are ID-Type pairs as they appear in the real CityGML model. Presence of other, not shown, elements of the model is indicated by *.

Once the target model is updated reflecting some reasoning (e.g., identifying the optimal position of the crane), changes have to be reflected back to the original model. To this end, TOPOCITY takes care of identifying changed objects and prompts the Application Policy to provide the 3D shape of the tower crane and spatial coordinates. For our case study, this was a fixed position, but a policy can specify arbitrary alternatives, from random to user-defined positioning, depending on the kind of links defined. Once those are given, TOPOCITY identifies the place in the original source hierarchy to arrange the new objects and reifies the model back again to the CityGML description.

Figure 6.2 shows a fragment of the original model and the final result as visualized CityGML descriptions. Note how certain reachability links between edges of three buildings are additionally defined, supposing these are buildings of interest for the construction site (Figure 6.1).



(a) Area without a tower crane (before).



(b) The crane is placed automatically via a *put* to the source model (after), reflecting its addition on the view model.

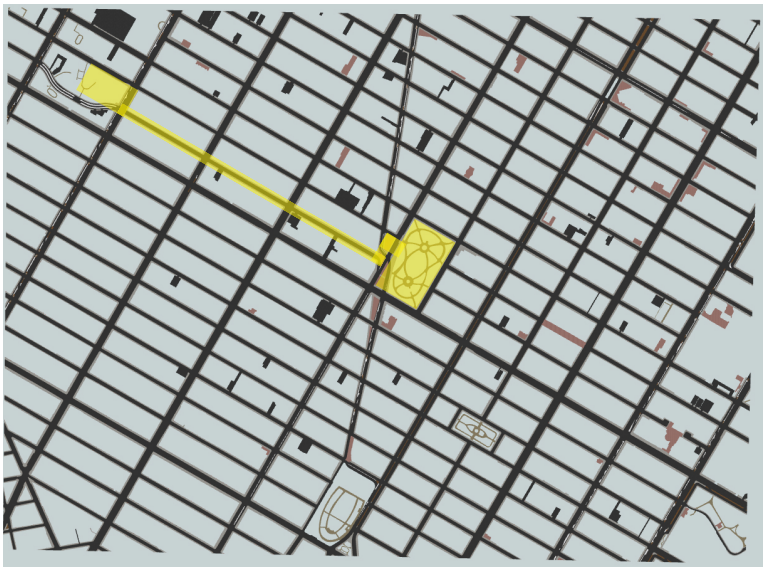
Figure 6.2: Placement of a crane entity on the derived, analyzable model (Fig. 6.1) entails its automatic reflection on the source city model (Fig. 6.2a), resulting in Fig. 6.2c.

6.2 EMERGENCY RESPONSE

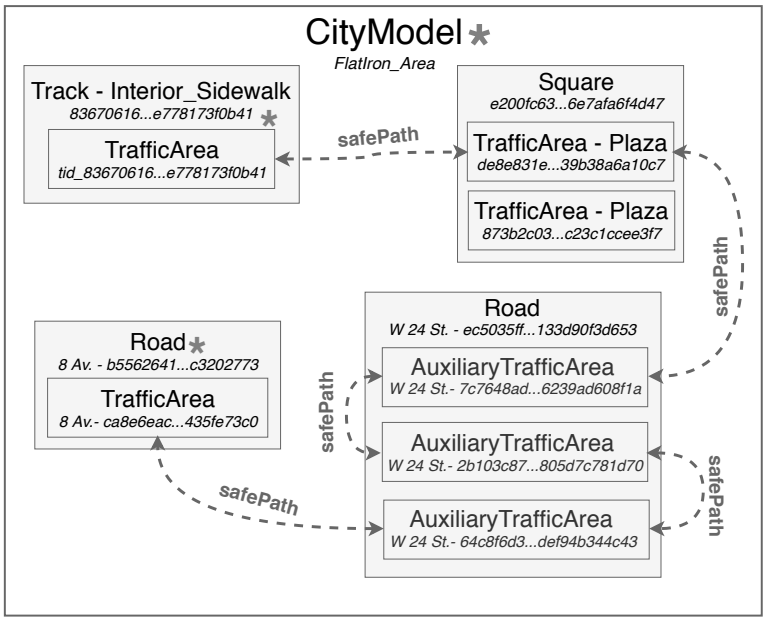
Technology adoption for fast emergency response in urban environments is gaining increasing attention: technological advances may in fact provide new human-computer interaction capabilities, allowing for effective real-time response. Consider the classical setting [30] where a disaster scenario is replicated in the Flatiron Building area of New York [1], with several relief entities (e.g. rescue teams, ambulances or Unmanned aerial vehicles – UAVs) dispatched throughout the area to locate and rescue victims [17, 49].

The agents have initial knowledge of the environment, given by the original model of the city. However, in such a scenario, we expect the model to be updated regularly, as soon as new information is acquired by monitoring processes. Agents must dynamically adjust search operations and rescue priorities through some criteria such as the likelihood of finding victims in an area or current disaster propagation. In order to perform such tasks, which largely amount to *planning* and *surveillance* [19], an actionable representation of the city can be a hypergraph in which nodes represent city objects, while links represent safe connections between multiple nodes. This typically occurs within a Monitor-Analyze-Plan-Execute loop, as this is an instance of a self-adaptive system. Agents monitor the area and update the model with the information they collect about safety of streets and buildings, while others escort civilians from the disaster area to hospitals. Path planning takes place based on analyzed monitored information upon the model, with the purpose of e.g., maximizing the number of victims rescued. Both the planning and monitoring facilities are not relevant for the synchronization of the models and they are therefore out of the scope of our research.

In our approach, we define and extract a CityGML ADE from the city model and populate it with real-time information, with the goal of making the *safe distance* relation between city objects explicit. TOPOCITY provides the hypergraph exploited by the agents, which is updated at runtime as the monitoring process generates new information. Figure 6.3 shows the aerial view of the Flat Iron Street area of New York as described by the CityGML model (6.3a), and the corresponding analyzable view (6.3b). A viable safe path for the city area is shown, both in the original model and in the analyzable one.



(a) The area nearby Flat Iron Street considered for our analysis.



(b) Fragment of the corresponding view generated.

Figure 6.3: Runtime safe path analysis models. The source (a) is transformed into the analyzable model (b). The highlighted area in (a) represents the safe path illustrated in (b). Nodes are ID-Type pairs as they appear in the available CityGML model of New York; the presence of other elements in parts of the model (not shown) is indicated by *.

6.3 DISCUSSION

The previous case studies address synchronization in two classical problems settings, from different application domains. In both cases, new kinds of analysis are enabled via the use of TOPOCITY. While the development process is somehow similar, the differences in the requirements and in the application goals, highlight some of the aspects one should consider when working with our tool.

6.3.1 *Cases Pecularities*

A prominent trait that differentiates the two cases relies in the level of details they target. In the case of Section 6.1, the goal of the application is described in quite abstract terms: designing a construction site like that one does not require to take into account the streets or any variation of the environment. For that reason, the basic LOD1 CityGML we used already has all the relevant information to solve the problem, which, on the other end, can become a very complex nonlinear optimization task. This fits well for a design-stage process, where computational time can be traded with better solutions with almost no costs. Conversely, in Section 6.2 we see the other side of the spectrum. In this case, we are dealing with a runtime evaluation, where having a more detailed representation of the environment can provide useful insights about future threats and possible routes to follow. We exploited a LOD2 CityGML model in this scenario, which has a richer set of features, regarding streets, squares, etc. That said, in contrast with the previous case, we now are dealing with a real-time updating environment, where any second of computation might have a high cost for the people to be rescued. Luckily, the problem to solve is simpler than the previous one, as the transformation here does not change the structure of the model, it just adds extra information to highlight the preferred path. If the procedure is executed on devices with very contrained resources, one could minimize the transformation and approximate the analysis of the view, in order to obtain faster decisions, albeit sub-optimal.

6.3.2 *Application Policy Realizability*

Another important question that one could ask is whether it is reasonable to assume that the two Application Policies used in the case studies are realizable in practice. The solution to the first case falls in the category of Geometric Constraint Satisfaction (GCS) problems, where links can be interpreted as constraints defined over the shapes of the buildings of the city. This kind of problems are out of the scope of our analysis, but there are various techniques to obtain interesting solutions in reasonable amounts of time [23]. Concerning the second

case, the Application Policy would instead be very trivial, since it would just check and add extra features concerning the level of safety of an area, or wether or not that building/object is part of a safe path. On the other hand, other forms of optimization could be designed for the transformation, to achieve even less computational penalty for non-changing parts of the city model.

6.3.3 *Preliminary Performance Evaluation*

We defer a complete performance evaluation of the transformations developed to future work, since our proof-of-concept tooling is still in prototypical state. However, some comments can be made about the execution time based on the models we used. Despite the complexity of the algorithm heavily depends on the number of links, we can say that, in cases where the number of links is negligible with respect to the number of nodes (this is not unlikely in cases where only a small area of the city is considered), the transformation is very fast. In our cases, the execution time of the actual transformation was only $2(\pm 1)\%$ of the total time, where the rest of the time was used in performing IO operations, parsing and data-binding. While these stages are necessary to successfully synchronize the two models, there are a handful of possible improvements that could be used, particularly for runtime evaluations like in the second case, to dramatically cut the overall computational cost.

6.3.4 *Policy Flexibility Limitations*

A significant flexibility constraint that could affect Application Policies, particularly for the first case study, has been briefly presented in Section 4.3. As anticipated there, links can be a very powerful medium for expressing arbitrarily complex configurations: in some convoluted scenarios, a putback to the original model may not be feasible or even worse, it may result in changes affecting a vast number of features, essentially resulting in a different model. To avoid this kind of problem, it is essential to contrain the kind of changes the Application Policy is allowed to make, limiting the changeable objects like described in Section 4.3. Moreover, it must be noted that, to date, the set of allowed Application Policy is quite limited. That is because they are described in terms of *pure* functions of the elements of the two models, $f(s, v)$. The *purity* constraint means they cannot have *side-effects* on the environment (like reading files or asking for user input), which restricts the Application Policies to parametric Haskell programs. We believe our solution addresses a relatively general set of meaningful applications, but further research on application scenarios may result in more precise understanding of practical limitations.

An important aspect in practice, is the level of automation desired – ideally, one would expect to be able to choose an Application Policy that meets certain needs, plug it in our framework and use the combinations of these programs with no extra effort. While we aim at that, our current experience has shown that, when dealing with custom ADE objects and relationships, one might still need to write some (very simple) custom bridging code to enable the transformation. Techniques of general programming might mitigate this problem significantly, and are certainly an area of future research.

The two cases considered for our evaluation purposes are model problems obtained from domain-specific literature, highlighting the use of bidirectional transformations within our framework for model-based engineering of space-dependent systems. We believe that the strength of our approach is twofold: firstly, adaptability is exhibited, since integrating disparate application-related sources of information still result in the same analyzable model; secondly, providing an automatic way to obtain an abstract model where verification can be performed, can lead to the development of more sophisticated analysis-based workflows.

6.3.5 *More on the case studies*

We conclude this chapter by observing that [43] already solves the tower crane problem of Section 6-A by developing a plugin for Autodesk Revit –an established tool in building and urban design. However, as pointed out by the authors, only a small set of pre-defined simple rules are allowed, implemented ad-hoc for this purpose. In addition, [26] shows that GIS-BIM models (like CityGML) have enough information for treating the problem in terms of geometrical and topological analysis. Our approach, on the contrary, is general enough to allow for complex rules and user-defined customization if a proper Application Policy is set in place.

7 RELATED WORK

While there are alternative approaches to the design of city spaces in general, and there are, as well, many studies of bidirectional transformation on other contexts, the approach to cyber-physical systems, based on bidirectional model transformations, is unprecedented and quite unique. Our goal has been, not only, to offer assurances on the quality of the transformation in terms of correctness and well-behavedness, but also, to comply with modern industrial standards and analysis technique as much as possible. Therefore, it seems natural to arrange the related work in a progressive way.

We start our brief presentation in Section 7.1 by considering the state of the art in model-based analysis of physical spaces, which is useful to position our work in terms of practical implications and support. Then, in Section 7.2 we consider the broader context of cyber-physical systems, together with engineering approaches based on bigraphs and alternative representations. Finally, we do a quick review of the alternative transformation techniques, as well as the theoretical foundations of consistency relations and enforcement in Section 7.3.

7.1 CITYGML & BIM ANALYSIS

Interest on model-based analysis of cities has been consistently growing in recent years. Historically, the models are classified in two categories: the Geographical Information Systems (GIS) and the BIM. CityGML is reportedly the first attempt to integrate both kinds of information. The adoption of CityGML for building modeling purposes has been studied extensively lately [35, 46, 58] and the integration of classical BIM features has been a leading design goal [13] in defining CityGML 3.0, to be released in 2019 [1]. In addition, city-based analysis is being developed in all kinds of application scenarios; most notably, recent efforts have been on traffic noise analysis [29], photovoltaic potentiality analysis [8], urban emission measurements analysis [6] and ubiquitous robot networks management [47]. Official city datasets are increasing, with recent public effort from Turkey [9], Singapore [44] and Germany [34] among all.

1 CityGML 3.0 Official Development Repository – https://github.com/opengeospatial/CityGML-3.0

7.2 CYBER-PHYSICAL SYSTEMS

Different forms of graphs as formal models of static representations of buildings or cities have been proposed in diverse fields such as architectural informatics [31] or computer graphics [56], with different objectives. Several approaches target case-based reasoning [2] in the architectural domain. However, actionable and analyzable models are necessary for advanced design and operation of overall space-dependent systems [48]. In [31], a topology of spatial configurations is extracted from building information models as well as handwritten architectural sketches [7] and represented as graphs. Focusing on security reasoning while aiming at early design phases, Porter et al. [39] propose a method and heuristics to discover security threats on building specifications via simulation. Analyses such as similarity checking are performed based on graph matching techniques [16]. Forms of graphs representing topology of space are highly useful. To this end, our target analyzable models are graph-based and readily analyzable with a variety of approaches. The notion of a cyber-physical space refers to a composite model able to capture complex relations of human, cyber and physical entities, which may span physical or computational barriers [52]. Such a model may be obtained from a physical model and enriched with formally-specified dynamics capturing possible ways it can change [50]. Spatio-temporal model checking of evolving cyber-physical spaces can then be considered [49], since topology can provide a system with awareness of multiple characteristics [36].

7.3 BIDIRECTIONAL TRANSFORMATIONS

Consistency between models has been of interest for many years, historically originating from the view-update problem in database research [10, 40]. It has become relevant within BX and, in most recent approaches, it has emerged as a means for automatic generation of lenses [45]. BiGUL is a formally verified putback-based bidirectional programming language [27]. Alternative relevant approaches based on different types of lenses are QVT [45], Triple Graph Grammars [25] and Edit Lenses [55].

8

CONCLUSIONS AND FUTURE WORK

Cities are growing to unprecedented sizes, bringing completely novel challenges whether we need to design new buildings or to restore run-down districts. Inspired by this quest, we examined the broad area of model-based engineering from space-based models to graph-based ones. We recognized that human experts usually favor space-based models, while machines are more effective with graph-based ones. It is, therefore, of great scientific interest, to bridge the gap between the two so that techniques on graph-based systems can indeed inform experts' choices. To that end, a reasonable starting point was to fix a reference model type for both categories: CityGML for space-based and Bigraphs for graph-based seemed suitable choices. However, to fulfill the goal, choosing the models is not sufficient. Before passing the information between the two, we must have a clear idea of when the two are consistent with each other. Once we set a proper definition, some procedures for repairing the consistency became of primary interest. We designed a strategy in various stages, that also considered application-specific goals, with adaptability and reusability in mind. We realized these ideas in a development framework. We presented some details of the implementation and discussed the limitations. Finally, we concluded our work by considering two real case studies from domain literature: one for the design process and the other for run-time analysis. After showing the potential of our approach, we concluded with a review of alternative ideas. The essential advantage of our method is that we get the best of both worlds: developers can analyze graph-based models, while domain experts can see suggestions they can easily understand. Despite that, it should not be considered the silver bullet for all synchronization problems. The class of problems for which our solution is advantageous is primarily the one where the consistency relation between the models rarely changes. Changing that, even slightly, might result in a massive rewriting of code, to restore the transformation, which is a common problem of solutions based on BX. It is, therefore, plausible that applications having in the consistency between the models a relevant source of change cannot benefit significantly by our work.

8.1 FUTURE WORK

The system we developed is just the first of a long chain of steps needed before having a real impact on daily domain-specific decisions. We end our presentation by highlighting some research directions that

we believe are intriguing and of influential relevance to achieve such a revolution.

DOMAIN-SPECIFIC ADVANCEMENTS: A fundamental limitation that hinders the flourishing of automated tools in the context of City Spaces is the lack of data. This limitation seems to be mainly a matter of time since the cause has been gathering a lot of public investment lately. Efforts like the ones of the World Council on City Data (WCCD), the United Smart Cities (USC), and the standardizing communities [5] must be followed to exploit the new results they achieve.

APPLICATION POLICY LIMITATIONS: The primary area of our interest involves the flexibility of Application Policy: the current implementation requires it to be a *pure* function with no side effects. This comes not only from a limitation of BiGUL, but also from the general awkwardness when dealing with side effects in Haskell. A very clear, and rather promising, analysis of these problems in Haskell comes from [37], and perhaps connecting with current research on monadic bidirectional transformations [4] can bring new light and allow for way more powerful (and simple) policies, dependant on user input and external information.

RELAXING EFFECTS OF CONSISTENCY CHANGES: Our technique takes shape starting from a precise definition of *what is what* in the two models; it is therefore not surprising that, if this relationship changes over time, the logic (and hence the code) to be reworked can be quite significant. Alternative methods from the Bidirectional Transformations community might prove more effective in solving this issue. While this may not be as interesting for the kinds of problems we addressed, it might as well result in a broadening of the potential impact of Bidirectional Transformations over Cyber-Physical Systems.

COMPUTATIONAL COMPLEXITY: Be it for design or runtime decisions, a clear understanding of the costs of the transformation is a mandatory condition for the broad applicability of the approach. While we did some performance analysis, we are in a context where worst-case considerations are not sufficiently informative to predict the time needed by the program. That is because the number of links in the graph, and the depth of analysis of the Application Policy might impact the computational time in non-linear ways. Working at a methodology for assessing the performances on relevant cases is a necessary direction for future work.

APPLICATIONS CONCERNING ADAPTABILITY: With the case study of section 6.2, we barely scratched the surface of what is possible

to accommodate self-adaptive behaviors. Future smart cities might benefit significantly from adaptive devices. For that reason, future work should address adaptability issues and requirements. Recent work on the topic is the one from the Shonan Meeting [57], which proposes some interesting ideas.

BIBLIOGRAPHY

[1] *3D City Model of New York City - TUM*. https://www.gis.bgu.tum.de/en/projects/new-york-city-3d/. 2015. (Visited on 01/26/2019).

[2] Agnar Aamodt and Enric Plaza. "Case-based reasoning: Foundational issues, methodological variations, and system approaches." In: *AI communications* 7.1 (1994), pp. 39–59.

[3] Mohammed Adel Abdelmegid, Khaled Mohamed Shawki, and Hesham Abdel-Khalek. "GA optimization model for solving tower crane location problem in construction sites." In: *Alexandria Engineering Journal* 54.3 (2015), pp. 519 –526. ISSN: 1110-0168. DOI: https://doi.org/10.1016/j.aej.2015.05.011. URL: http://www.sciencedirect.com/science/article/pii/S1110016815000836.

[4] Faris Abou-Saleh, James Cheney, Jeremy Gibbons, James McKinna, and Perdita Stevens. "Introduction to Bidirectional Transformations." In: *Bidirectional Transformations: International Summer School, Oxford, UK, July 25-29, 2016, Tutorial Lectures*. Cham: Springer International Publishing, 2018, pp. 1–28. ISBN: 978-3-319-79108-1. DOI: 10.1007/978-3-319-79108-1_1. URL: https://doi.org/10.1007/978-3-319-79108-1_1.

[5] Giorgio Agugiaro, Joachim Benner, Piergiorgio Cipriano, and Romain Nouvel. "The Energy Application Domain Extension for CityGML: enhancing interoperability for urban energy simulations." In: *Open Geospatial Data, Software and Standards* 3.1 (2018), p. 2. ISSN: 2363-7501. DOI: 10.1186/s40965-018-0042-y. URL: https://doi.org/10.1186/s40965-018-0042-y.

[6] Dirk Ahlers, Frank Alexander Kraemer, Anders Eivind Braten, Xiufeng Liu, Fredrik Anthonisen, Patrick Driscoll, and John Krogstie. "Analysis and Visualization of Urban Emission Measurements in Smart Cities." In: *EDBT*. 2018.

[7] Sheraz Ahmed, Markus Weber, Marcus Liwicki, Christoph Langenhan, Andreas Dengel, and Frank Petzold. "Automatic analysis and sketch-based retrieval of architectural floor plans." In: *Pattern Recognition Letters* 35 (2014), pp. 91–100.

[8] Nazmul Alam and Volker Coors. "Detecting shadow for direct radiation using CityGML models for photovoltaic potentiality analysis." In: 2013.

[9] S. Ates Aydar, Jantien E. Stoter, Hugo Ledoux, E. Demir Ozbek, and Tahsin Yomralioğlu. "Establishing a National 3 D Geodata Model for Building Data Compliant to Citygml : Case of Turkey." In: 2016.

[10] F. Bancilhon and N. Spyratos. "Update Semantics of Relational Views." In: *ACM Trans. Database Syst.* 6.4 (Dec. 1981), pp. 557–575. ISSN: 0362-5915. DOI: 10.1145/319628.319634. URL: http://doi.acm.org/10.1145/319628.319634.

[11] Filip Biljecki, Kavisha Kumar, and Claus Nagel. "CityGML Application Domain Extension (ADE): overview of developments." In: *Open Geospatial Data, Software and Standards* 3.1 (Aug. 2018), p. 13. ISSN: 2363-7501. DOI: 10.1186/s40965-018-0055-6. URL: https://doi.org/10.1186/s40965-018-0055-6.

[12] Filip Biljecki, Jantien Stoter, Hugo Ledoux, Sisi Zlatanova, and Arzu Çöltekin. "Applications of 3D City Models: State of the Art Review." In: *ISPRS International Journal of Geo-Information* 4.4 (2015), pp. 2842–2889. ISSN: 2220-9964. DOI: 10.3390/ijgi4042842. URL: http://www.mdpi.com/2220-9964/4/4/2842.

[13] *CityGML 3.0 Requirements - Munich 2013*. http://en.wiki.modeling.sig3d.org/index.php/Workshop_Munich_2013. 2013. (Visited on 01/24/2019).

[14] *CityGML Open Data Initiatives*. http://www.citygmlwiki.org/index.php?title=Open_Data_Initiatives_in_Germany. 2017. (Visited on 11/25/2019).

[15] Open Geospatial Consortium. *City Geography Markup Language (CityGML) Encoding Standard, version: 2.0.0*. http://www.opengis.net/spec/citygml/2.0. 2012. (Visited on 11/17/2014).

[16] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. "Thirty years of graph matching in pattern recognition." In: *International journal of pattern recognition and artificial intelligence* 18.03 (2004), pp. 265–298.

[17] Wesley DeBusk. "Unmanned Aerial Vehicle Systems for Disaster Relief: Tornado Alley." In: Infotech@Aerospace Conferences. 0. American Institute of Aeronautics and Astronautics, Apr. 2010. Chap. Unmanned Aerial Vehicle Systems for Disaster Relief: Tornado Alley. DOI: 10.2514/6.2010-3506. URL: https://doi.org/10.2514/6.2010-3506.

[18] Chuck Eastman, Charles M Eastman, Paul Teicholz, and Rafael Sacks. *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*. J.W & S, 2011.

[19] Christopher M. Eaton, Edwin K. P. Chong, and Anthony A. Maciejewski. "Multiple-Scenario Unmanned Aerial System Control: A Systems Engineering Approach and Review of Existing Control Methods." In: *Aerospace* 3.1 (2016). ISSN: 2226-4310. DOI: 10.3390/aerospace3010001. URL: http://www.mdpi.com/2226-4310/3/1/1.

[20] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. "Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem." In: *ACM Trans. Program. Lang. Syst.* 29.3 (2007), p. 17. DOI: 10.1145/1232420.1232424. URL: http://doi.acm.org/10.1145/1232420.1232424.

[21] *Galería de Clásicos de Arquitectura: Villa San Luis / CORMU - 26*. https://www.plataformaarquitectura.cl/cl/761203/clasicos-de-arquitectura-barrio-san-luis-cormu. 2015. (Visited on 11/25/2019).

[22] Amal Gassara, Ismael Bouassida Rodriguez, Mohamed Jmaiel, and Kalil Drira. "Encoding Bigraphical Reactive Systems into Graph Transformation Systems." In: *Electronic Notes in Discrete Mathematics* 55 (2016). 14th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW16), pp. 207 – 210. ISSN: 1571-0653. DOI: https://doi.org/10.1016/j.endm.2016.10.051. URL: http://www.sciencedirect.com/science/article/pii/S1571065316302074.

[23] Jian-Xin Ge, Shang-Ching Chou, and Xiao-Shan Gao. "Geometric constraint satisfaction using optimization methods." In: *Computer-Aided Design* 31.14 (1999), pp. 867 –879. ISSN: 0010-4485. DOI: https://doi.org/10.1016/S0010-4485(99)00074-3. URL: http://www.sciencedirect.com/science/article/pii/S0010448599000743.

[24] Jeremy Gibbons and Perdita Stevens, eds. *Bidirectional Transformations*. Springer International Publishing, 2018. DOI: 10.1007/978-3-319-79108-1. URL: https://doi.org/10.1007/978-3-319-79108-1.

[25] Frank Hermann, Hartmut Ehrig, Fernando Orejas, Krzysztof Czarnecki, Zinovy Diskin, and Yingfei Xiong. "Correctness of Model Synchronization Based on Triple Graph Grammars." In: *MoDELS*. 2011.

[26] Javier Irizary and Ebrahim Karan. "Optimizing location of tower cranes on construction sites through GIS and BIM integration." In: *Electronic Journal of Information Technology in Construction* 17 (Sept. 2012), pp. 351–366.

[27] Hsiang-Shang Ko, Tao Zan, and Zhenjiang Hu. "BiGUL: a formally verified core language for putback-based bidirectional programming." In: *PEPM*. 2016.

[28] Thomas Kolbe, Gerhard Gröger, and Lutz Plümer. "CityGML: Interoperable access to 3D city models." In: *Geo-information for disaster management*. Springer, 2005.

[29] Amol Konde and Sameer Saran. "Web enabled spatio-temporal semantic analysis of traffic noise using CityGML." In: 2017.

[30] Mei-Po Kwan and Jiyeong Lee. "Emergency response after 9/11: the potential of real-time 3D GIS for quick emergency response in micro-spatial environments." In: *Computers, Environment and Urban Systems* 29.2 (2005), pp. 93 –113. ISSN: 0198-9715. DOI: https://doi.org/10.1016/j.compenvurbsys.2003.08.002. URL: http://www.sciencedirect.com/science/article/pii/S0198971503000796.

[31] Christoph Langenhan, Markus Weber, Marcus Liwicki, Frank Petzold, and Andreas Dengel. "Graph-based retrieval of building information models for supporting the early design stages." In: *Advanced Engineering Informatics* 27.4 (2013), pp. 413–426.

[32] Robin Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.

[33] G.A. van Nederveen and F.P. Tolman. "Modelling multiple views on buildings." In: *Automation in Construction* 1.3 (1992), pp. 215 –224. ISSN: 0926-5805. DOI: https://doi.org/10.1016/0926-5805(92)90014-B. URL: http://www.sciencedirect.com/science/article/pii/092658059290014B.

[34] *Nordrhein-Westfalen Open Geographic Data*. https://www.opengeodata.nrw.de/produkte/geobasis/3d-gm/. 2017. (Visited on 01/24/2019).

[35] Ken Arroyo Ohori, Abdoulaye A. Diakité, Thomas Krijnen, Hugo Ledoux, and Jantien E. Stoter. "Processing BIM and GIS Models in Practice: Experiences and Recommendations from a GeoBIM Project in The Netherlands." In: *ISPRS Int. J. Geo-Information* 7 (2018), p. 311.

[36] Liliana Pasquale, Carlo Ghezzi, Claudio Menghi, Christos Tsigkanos, and Bashar Nuseibeh. "Topology Aware Adaptive Security." In: *Proc. of the 9th Int. Symp. on Software Engineering for Adaptive and Self-Managing Systems*. 2014, pp. 43–48.

[37] Simon Peyton Jones. "Tackling the awkward squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell." In: *Engineering theories of software construction*. IOS Press, Jan. 2001, pp. 47–96. ISBN: ISBN 1 58603 1724. URL: https://www.microsoft.com/en-us/research/publication/tackling-awkward-squad-monadic-inputoutput-concurrency-exceptions-foreign-language-calls-haskell/.

[38] Karl R. Popper. *Objective Knowledge: An Evolutionary Approach*. Oxford University Press, 1975.

[39] Stuart Porter, Terence Tan, Tele Tan, and Geoff West. "Breaking into BIM: Performing static and dynamic security analysis with the aid of BIM." In: *Automation in Construction* 40 (2014), pp. 84–95.

[40] Terrence W. Pratt. "Pair grammars, graph languages and string-to-graph translations." In: *Journal of Computer and System Sciences* 5.6 (1971), pp. 560 –595. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(71)80016-8. URL: http://www.sciencedirect.com/science/article/pii/S0022000071800168.

[41] *Revit products 2018 documentation - Constraints definition feature*. https://knowledge.autodesk.com/support/revit-products/learn-explore/caas/CloudHelp/cloudhelp/2018/ENU/Revit-Model/files/GUID-4AD7D371-F757-4BFF-9F3C-8321A77D3A02-htm.html. 2018. (Visited on 01/24/2019).

[42] Sameer Saran, Kapil Oberai, Parag Wate, Amol Konde, Arnab Dutta, Kavisha Kumar, and A. Senthil Kumar. "Utilities of Virtual 3D City Models Based on CityGML: Various Use Cases." In: *Journal of the Indian Society of Remote Sensing* 46.6 (June 2018), pp. 957–972. ISSN: 0974-3006. DOI: 10.1007/s12524-018-0755-5. URL: https://doi.org/10.1007/s12524-018-0755-5.

[43] Kevin Schwabe, Markus König, and Jochen Teizer. "BIM Applications of Rule-Based Checking in Construction Site Layout Planning Tasks." In: July 2016. DOI: 10.22260/ISARC2016/0026.

[44] K. H. Soon and V. H. S. Khoo. "CITYGML MODELLING FOR SINGAPORE 3D NATIONAL MAPPING." In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-4/W7 (2017), pp. 37–42. DOI: 10.5194/isprs-archives-XLII-4-W7-37-2017. URL: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-4-W7/37/2017/.

[45] Perdita Stevens. "Bidirectional model transformations in QVT: semantic issues and open questions." In: *Software & Systems Modeling* 9.1 (2010), p. 7.

[46] Rudi Stouffs, Helga Tauscher, and Filip Biljecki. "Achieving Complete and Near-Lossless Conversion from IFC to CityGML." In: *ISPRS Int. J. Geo-Information* 7 (2018), p. 355.

[47] Yaemi Teramoto, Akiko Sato, Kishiko Maruyama, and Hitoshi Tomita. "Map Representation for Ubiquitous Network Robot Services." In: *Proceedings of the Fourth ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness*. ISA '12. Redondo Beach, California: ACM, 2012, pp. 29–32. ISBN: 978-1-4503-1697-2.

DOI: 10.1145/2442616.2442623. URL: http://doi.acm.org/10.1145/2442616.2442623.

[48] Christos Tsigkanos, Timo Kehrer, and Carlo Ghezzi. "Architecting dynamic cyber-physical spaces." In: *Computing* 98.10 (2016), pp. 1011–1040.

[49] Christos Tsigkanos, Timo Kehrer, and Carlo Ghezzi. "Modeling and verification of evolving cyber-physical spaces." In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, 2017*. 2017, pp. 38–48.

[50] Christos Tsigkanos, Timo Kehrer, Carlo Ghezzi, Liliana Pasquale, and Bashar Nuseibeh. "Adding static and dynamic semantics to building information models." In: *Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems*. ACM. 2016, pp. 1–7.

[51] Christos Tsigkanos, Nianyu Li, Zhi Jin, Zhenjiang Hu, and Carlo Ghezzi. "On early statistical requirements validation of cyber-physical space systems." In: *Proceedings of the 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems, ICSE 2018, Gothenburg, Sweden, May 27, 2018*. 2018, pp. 13–18.

[52] Christos Tsigkanos, Liliana Pasquale, Carlo Ghezzi, and Bashar Nuseibeh. "On the Interplay Between Cyber and Physical Spaces for Adaptive Security." In: *IEEE Trans. Dependable Sec. Comput.* 15.3 (2018), pp. 466–480.

[53] *United Nations – World Urbanization Prospects*. https://population.un.org/wup/. 2018. (Visited on 11/25/2019).

[54] Ennio Visconti, Christos Tsigkanos, Zhenjiang Hu, and Carlo Ghezzi. "Model-Driven Design of City Spaces via Bidirectional Transformations." In: *Proceedings of the ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems*. MODELS '19. Munich, Germany, 2019.

[55] Daniel Wagner and Nate Foster. "Symmetric Edit Lenses : A New Foundation for Bidirectional Languages." In: 2014.

[56] Raoul Wessel, Ina Blümel, and Reinhard Klein. "The Room Connectivity Graph: Shape Retrieval in the Architectural Domain." In: *The 16-th Intl Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision*. 2008. ISBN: 978-80-86943-15-2.

[57] Yijun Yu, Arosha K. Bandara, Shinichi Honiden, Zhenjiang Hu, Tetsuo Tamai, Hausi A. Müller, John Mylopoulos, and Bashar Nuseibeh, eds. *Engineering Adaptive Software Systems - Communications of NII Shonan Meetings*. Springer, 2019. ISBN: 978-981-13-2184-9. DOI: 10.1007/978-981-13-2185-6. URL: https://doi.org/10.1007/978-981-13-2185-6.

[58] Junxiang Zhu, Graeme Wright, Jun Wang, and Xiangyu Wang. "A Critical Review of the Integration of Geographic Information System and Building Information Modelling at the Data Level." In: *ISPRS Int. J. Geo-Information* 7 (2018), p. 66.

DECLARATION

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or processional qualification except as specified.

Milano, Anno Accademico 2018 – 2019

Ennio Visconti

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić, with the exception of the cover, which has been designed specifically for this work by Ennio Visconti. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Thank you very much for your feedback and contribution.

Final Version as of December 2, 2019 (classicthesis v4.6).