2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)

Inferring Analyzable Models from Trajectories of Spatially-Distributed Internet of Things

Christos Tsigkanos
Distributed Systems Group
TU Wien
Austria

Laura Nenzi
Department of Mathematics and Geoscience
University of Trieste
Italy

Michele Loreti
School of Science and Technology
University of Camerino
Italy

Martin Garriga
Universidad Nacional del Comahue
Neuquen, Patagonia
Argentina

Schahram Dustdar
Distributed Systems Group
TU Wien
Austria

Carlo Ghezzi
Dipartimento di Elettronica,
Informazione e Bioingegneria
Politecnico di Milano, Italy

***Abstract*—Internet of things systems are increasingly common nowadays. They feature spatially-distributed, mobile entities with an arising collective behavior. Such entities bear radionavigation sensors that produce positioning information, then used by the (software-enabled) device to produce positioning information over time, referred to as trajectories. However, software applications built on top of this require composite models of space to be in place; such models can provide adaptive behaviors by observing, evaluating, and reacting to a constantly changing spatial environment. This is typically achieved by monitoring for changes, analyzing requirements violations and then planning and executing adequate countermeasures. We are concerned with the fact that model representations of space are highly pertinent to requirements reasoning of internet of things systems, and such spatial models can be very useful for engineering adaptation. To this end, we provide and implement a technique to infer analyzable models from general trajectories of spatially-distributed systems, which may be used for engineering analysis or planning facilities for the overall self-adaptive systems. Moreover, we illustrate how such spatial models are used for evaluation of requirements predicating about the structure of space, the spatial distribution of devices, temporal as well as quantitative aspects through formal spatio-temporal verification.**

I. Introduction

The world is evolving and integrating increasingly, thanks to novel types of pervasive systems, technologies and paradigms such as the Internet of Things (IoT). Such systems often feature physically distributed entities roaming inside the physical space [1], exhibiting an emergent collective behavior. This comes along with new types of requirements and the need for their dependable satisfaction over an ever changing context, while scale and complexity renders this even more challenging.

IoT systems operating within a dynamic spatial environment are commonplace nowadays; think of a bike-sharing fleet within a smart city optimizing spatial distribution with respect to user demand, or crowdsourcing of environmental measurements across wide urban spaces. The mobile entities comprising the IoT system are usually equipped with satellite-based radionavigation sensors such as GPS or Galileo [2], providing geolocation information to the software-enabled devices. Radionavigation sensors typically produce readings of geo-spatial positioning including longitude and latitude in the earth spheroid. Those can be in sequences of coordinate points over time, what is commonly referred to as a *trajectory* [3]. Trajectories are used for various reasons, including situations where map data are not available, in a form that is not analyzable or where there are legal issues in using richer spatial representations. Moreover, trajectories are used in place of map data in machine learning [4] and data mining [5] domains. Advanced techniques have been developed, tackling adjacent problems of low sampling, uncertainty and noise [6] of trajectory data [7], their context [8], [9], or forecasting [10].

From a software engineering perspective, spatially-distributed IoT systems live within a dynamic spatial environment populated with devices, changing context and/or localized resources. This implies having analyzable models in place, used to observe, evaluate and react to a constantly changing space [11]. All of this must take place at system *runtime*, ensuring that the changing spatial distribution – for example due to actions performed by active agents, or by the external environment – does not lead to requirements violations [12]. Typically, this can be achieved through an autonomic, self-adaptive approach – e.g., a MAPE loop [13]: (M)onitoring the spatial environment for changes, (A)nalyzing possible requirements violations, (P)lanning required countermeasures (e.g., moving a device from one point of space to another) and then (E)xecuting such actions and updating the shared model of space for the next loop.

We are not concerned with geo-spatial, geo-desic, geo-database or trajectory-semantic [14]–[16] aspects here, but with model representations for software-enabled systems of IoT in general, and their suitability for engineering adaptation. This entails multiple aspects within MAPE activities: (i) monitoring raw readings and constructing analyzable models, (ii) performing requirements analysis upon them (iii) choice and (iv) execution of appropriate counteraction measures that may be triggered to satisfy requirements.

In particular, we consider adaptive systems which operate in

978-1-7281-3368-3/19/$31.00 ©2019 IEEE
DOI 10.1109/SEAMS.2019.00021

a discrete space. Models of such space arise from topological relations in the spatial environment, where the information abstraction of location is inherently important [12]. Previous work [1] has considered spatio-temporal verification and has advocated that spatial topology can provide a system with awareness of multiple contextual characteristics critical for requirements satisfaction, ranging from safety and security [17], [18] to performance [19], [20]. With the present paper and artifact, we address the need of the community for actionable, real system models where self-adaptation research can be developed and evaluated. Our analyzable models are constructed from general trajectories observed from entities within spatially-distributed IoT, which may be used for engineering analysis or planning facilities for self-adaptive systems. We illustrate how such models may be used specifically for evaluation of requirements predicating on the global distribution of IoT entities through formal spatio-temporal verification. Specifically, our artifact's contributions are as follows.

- We demonstrate a technique and provide tooling facilities to generate a graph-based model capturing accessibility in space from an observed set of trajectories and known points in geographical space, sourced from the domain.
- We utilize the *Spatio-Temporal Reach and Escape Logic* (STREL) [21] to specify and verify complex requirements over the graph-based model, which predicate about the structure of space, the spatial distribution of IoT devices, temporal as well as quantitative aspects.

The rest of the paper is structured as follows. Section II gives a birds-eye view of the proposed approach within self-adaptive systems as well as illustrating a running example. Section III describes analyzable models inference from IoT trajectories. After illustrating model population at runtime, Section V showcases spatio-temporal verification with STREL on obtained models. Section VI concludes the paper.

II. Models from Points for Self-Adaptive Systems

IoT devices are often equipped with satellite-based radionavigation sensors, producing readings of geo-spatial positioning. The induced IoT systems are thus situated within a dynamic spatial environment, and composite models are often used for engineering self-adaptation; observing, evaluating and reacting to the constantly changing space.

Figure 1 illustrates our approach in context of the MAPE loop. IoT entities roaming inside a spatial environment produce trajectories. Trajectories, combined with knowledge of already known landmarks in space are used to infer analyzable models, which include both a model of space (as inferred through observed trajectories) and sequences of presences of IoT entities in landmarks based on trajectories (as inferred discrete traces). The inferred model of space can then be brought at runtime, populated with contextual information and be used for analysis within a MAPE loop, a significant case of which we illustrate in Section V. Inferred traces can be used for simulation or testing. Planning countermeasures or executing actions is also possible based on verification results.

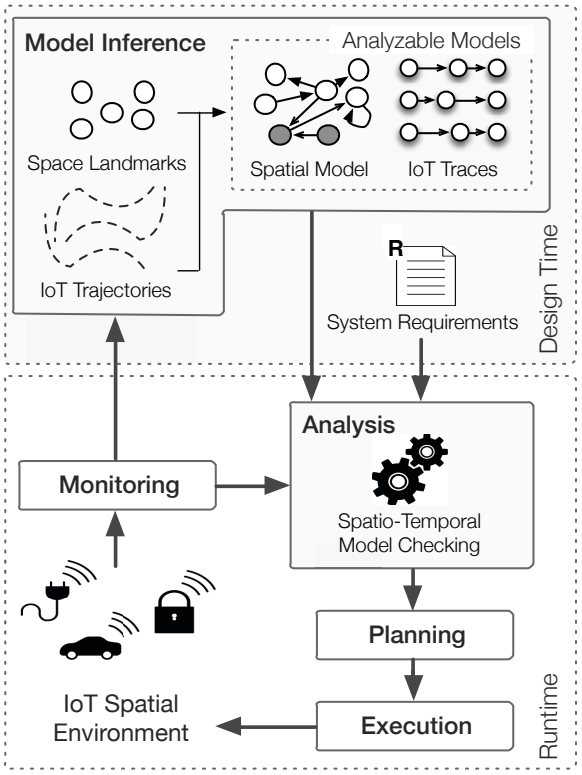

Fig. 1. Analyzable Models for Adaptation within Spatially-Distributed IoT.

Our resulting spatial models are graph-based and are inferred from past observation of trajectories of IoT devices in geographical space [3]. Trajectories are often available as a domain model, as public datasets or obtained e.g. by monitoring a spatially-distributed system usage for a limited amount of time [22]. We infer (i.e. upwards arrow from Monitoring in Figure 1) from a set of trajectories an analyzable model essentially in two steps:

1) Obtain landmarks in the city. These are nodes in a graph, and each consists of a name, various attributes and coordinates of the point in geographical space, sourced from some widely available repository (e.g. Open Street Map [23] or SPOI [24]).
2) Map trajectories of active entities over landmarks. If, based on a predefined distance, a trajectory passes through two points, these two points are considered *accessible* from one another, and an edge between the respective nodes is created.

When the system is operational, requirements validation may need to take place. The model is populated (i.e. rightwards arrow from Monitoring in Figure 1) by current environmental information, such as IoT device positions or sensor readings. Subsequently, verification facilities evaluate requirements.

Motivating Example. Consider a fleet of taxis roaming in the physical space of a city. The taxis, equipped with geolocation sensors (e.g. GPS) maintain a connection to the network, rendering the system an instance of the IoT. As taxis

move within the city, they produce and transmit sequences of their positions in the spatial plane. In our scenario, we consider three representative requirements which concern the system's operation. As typical QoS goals of a spatially-distributed application, requirements predicate about the global spatial distribution of taxis, the structure of the space and the time taxis take to traverse the city:

(R1) "Each hospital has a taxi less than 10 minutes away";
(R2) "Taxis should be at landmarks where one can reach the main square through bus or metro stops in less than 10 minutes";
(R3) "If there is a landmark where more than 200 people are located, then a taxi will be there within 20 minutes";

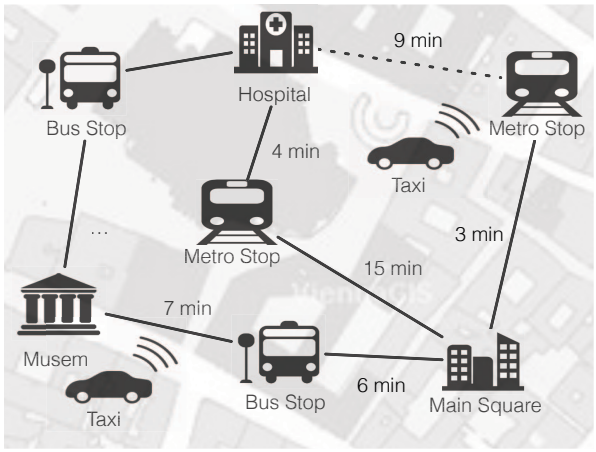

Fig. 2. Taxis as IoT devices near landmarks in a city.

The three exemplar properties represent different kinds of non-trivial analyses that may be sought while the system is in operation. Firstly, a model kept at runtime can be used to account for positions of taxis in the spatial plane, where an intuitive requirement about taxi availability can be stated (R1). Secondly, complex spatial relationships can also be stated (R2) – observe that the requirement does not specify which, or how many bus or metro stops need to be traversed when reaching the main square. Finally, other sensing data can be integrated upon the model, such as measurements of the number of people in locations, leading to a more complex requirement predicating about multiple aspects (R3). Furthermore, R3 represents a characteristic property requiring a verification procedure able to consider not only the current state of the spatial plane but also the progression of the system to the that state. Results of verification of R1-R3 may be used for counteraction planning, depending on system-wide goals.

Inherent in the formulation of the scenario above is a description of how various points in the city are connected, meaning that one can use a taxi to go from a point to another. As movements of taxis take time to traverse points in the city, time should also be considered. Such an *accessibility* model, capturing also traversal time, can be built by monitoring taxi over some time. The result is a graph structure, which has nodes corresponding to various points of the city and edges connecting those which are directly reachable. Edges can be labelled with entity traversal time, or other aspects such as energy cost, distances etc. For our example city, such a graph is depicted in Figure 2, where various city landmarks such as a park, a museum and transportation stops are connected, and edges capture some traversal time.

III. Model Inference from IoT Trajectories

In this section, we describe how a discrete, analyzable model can be inferred from observed trajectories of IoT entities, which constitute the *source* model. This model inference process makes use of a set of known landmarks in space, attributes of which are coordinates. Trajectories are assumed to have been monitored from some past operation of the spatialy-distributed IoT system. Back to our running example, if an taxi emits a sequence of geolocation points starting from the landmark *museum* (Figure 2), going through a *bus stop* and arrives near the subsequent *main square*, then the three are intuitively (step-wise) connected, since accessibility was demonstrated by at least one IoT device trajectory. Thus, the first step in inferring a model consists of registering the landmarks in space which a single IoT entity passes through. What we seek is a discrete sequence of landmarks (referred to as a *trace*), at each of which the IoT entity was found at most some defined distance, at some point in time.

Algorithm 1 for inferring a discrete trace from a trajectory has as input a set of landmarks, a search range parameter and an IoT trajectory. A landmark consists of some characteristic name and a pair of coordinates, while a trajectory is a set of time-stamped coordinates where the IoT entity appeared. For our running example, we consider a single type of IoT entity – a taxi – but more can be handled similarly. The predefined search range aims to mitigate sensing errors in the data, as well as resolution within the scale of which landmarks are defined. The algorithm proceeds to identify landmarks which are *near* (i.e. within a measure of distance in an earth spheroid) points of the trajectory. If one is found, it is registered as a time-stamped point in a trace. This trace describes the progression of the entity through landmarks, consisting the output of the algorithm. Algorithm 1 is in a simple form for clarity – a procedure calculating traces based on temporal windows upon multiple IoT entity trajectories can be similarly derived.

Subsequently, we can consider a set of IoT device traces over the set of known landmarks in space. By considering the set of observed traces, we can construct an accessibility relation in geographical space; the relation connects the (known) landmarks that the IoT collective has visited in the past and records the mean time IoT devices have taken to traverse the landmarks. Essentially, if a trace shows that an IoT device went from a landmark to another, these two landmarks must be connected, and one is *accessible* from one another; the time the device took to traverse them is also recorded. The accessibility relation between landmarks may induce a *spatial model* [21], that, together with a set of traces, can be used for formal reasoning with a spatio-temporal logic.

Algorithm 1 Inferring a Discrete Trace from a Trajectory.

inferTrace : id, trajectory, landmarks, range → trace

Require:
TR – trajectory: $p_0 \mid ... \mid p_n$
p_n – trajectory point: $\{id, datetime, longitude, latitude\}$
L – landmarks: $\{name, longitude, latitude...\}$

Ensure:
$trace$ – $\{id \in l_i\} \cdot \{id \in l_j\}$ ...

1: `map_landmarks()` – in spheroid geometry
2: $trace$ = []
3: **for all** $p \in TR$ **do**
4: **for all** $l \in L$ **do**
5: **if** `distance_spheroid`$(p, l) < range$ **then**
6: $trace = trace \cdot \{id \in l_{name}\ at\ datetime\}$
7: **end if**
8: **end for**
9: **end for**

Algorithm 2 Inferring a Model from Observed Traces.

inferModel : traces → weighted spatial model

Require:
$traces$ – $trace_0 \mid ... \mid trace_n$ ...
$trace$ – $\{id \in l_i\} \cdot \{id \in l_j\}$ ...
L – landmarks: $\{name, longitude, latitude...\}$

Ensure:
$M = (L, \mathcal{W})$ – spatial model
$\mathbf{W} \subseteq L \times \mathbb{R} \times L$ – mean temporal proximity of landmarks
SU – $location_0 \mid ... \mid location_n$ ...

1: **for all** $t \in traces$ **do**
2: **for all** $e \in trace$ **do**
3: **if** $e.l.name \notin SU$ **then**
4: $SU \leftarrow SU \cup e.l.name$
5: **end if**
6: δ = `next`$(e).datetime - e.datetime$ – timedelta
7: **if not** $\mathbf{W}(l.p,$ `next`$(e).l)$ **then**
8: $\mathbf{W} \leftarrow \mathbf{W} \cup (\ell_1, \delta, \ell_2)$
9: **else**
10: $\mathbf{W}(l.p,$ `next`$(e).l)$ = `running_mean`(δ)
11: **end if**
12: **end for**
13: **end for**

We define a *spatial model* $\mathcal{S}$ as a pair $\langle L, \mathbf{W} \rangle$ where L is a set of *locations* (the landmarks), also named *space universe* and $\mathbf{W} \subseteq L \times \mathbb{R} \times L$ is a *proximity function* associating at most one label $w \in \mathbb{R}$ with each distinct pair $\ell_1, \ell_2 \in L$. The meaning of the weight w depends on the type of analysis. As we observed, w can be e.g. the euclidean distance between landmarks, hops for connected nodes, or for our example purposes, the observed traversal time by observing past trajectories. A pair $(\ell_1, w, \ell_2) \in \mathbf{W}$ iff ℓ_1, ℓ_2 satisfy the accessibility relation. In the following, we will equivalently write $(\ell_1, w, \ell_2) \in \mathbf{W}$ as $\mathbf{W}(\ell_1, \ell_2) = w$.

Given a set of discrete traces capturing the progression of an entity over landmarks in space and a set of landmarks, Algorithm 2 computes a spatial model reflecting the *accessibility* of landmarks based on given observed traces. The model consists of a graph for which the set of edges forms an accessibility relation (i.e., through W), as well as the mean traversal time annotated upon its edges – nodes of the graph are the landmarks the IoT devices have appeared in, in some previously observed trace.

Note how one can additionally capture arbitrary propositions on nodes as desired, such as e.g. initial positions or counts of devices based on the traces observed. Similarly, time or distance may be introduced to the spatial model by time-stamping points of a trajectory when mapping over landmarks and taking into account trajectory traversal time – for our example, the average time traversed is used, but other signals may be adopted (e.g. geographical or logical distance, or energy cost). The additional dimension can enable quantitative reasoning. In Figure 3, the relation of *accessibility* between locations (i.e. landmarks) gives rise to a spatial model. Thereupon, further propositions associated with each model point may capture a landmark name or presence of a taxi.

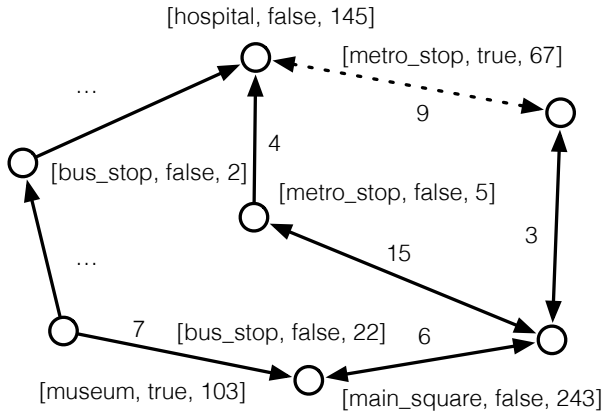

Fig. 3. Propositions over connected points form a weighted spatial model for the example city, here shown populated with presences of taxis and people.

In practice. The above procedures are implemented in a set of python scripts interacting with a geospatially-enabled mongoDB instance which resolves spatial queries over an earth spheroid geometry. To use the supporting tooling[1] in practice, one i) obtains a trajectory dataset (e.g. [25], [26]) ii) obtains a set of geospatial landmarks (e.g. Open Street Map [23] or SPOI [24]), iii) configures parameters for the spatial model inference (such as the landmarks' search range), and iv) invokes the tool. Helper procedures are available, e.g. for calculating traces based on time windows or simulating movement of entities within models. Resulting models can be readily verified with a variety of tools operating on the spatial models presented (e.g. [27], [28]), or transformed to other formalisms taking advantage of the generality of the underlying graph structure.

The artifact is available [29], reflecting the procedure described in Section III. Additionally, analyzable spatial models inferred from the Microsoft T-Drive dataset [25], [26] are provided for reference, comprising of trajectories of 10,357 taxis in Beijing spanning one week. Landmarks of Beijing are sourced from the OpenStreetMap repository [23]. The inferred spatial model of Beijing has 487,133 weighted edges over

[1] dsg.tuwien.ac.at/staff/ctsigkanos/amelia/.

8,798 nodes. Weights on edges reflect the mean time observed taxis –as IoT devices– took to traverse pairs of accessible points within the source dataset.

IV. Spatial Model at System Runtime

The model defined in the previous section can be constructed from trajectories obtained from some past operation of the spatialy-distributed IoT system. The model captures the inferred structure of the space, the various landmarks that have been observed as well as the weights computed. This constitutes the model inference stage (upper part of Figure 1).

At system runtime, requirements validation may need to take place. To this end, an analyzable model needs to be kept alive at runtime, populated and updated with contextual or environmental information. Monitoring may record information that is unknown at design time, such as the current position of various IoT entities, or other sensor readings or metrics. For our example, we assume that sensors dispersed in the city record the number of people in the various landmarks. Accordingly, the model is updated through Monitoring. In Figure 3, the number of people in a landmark is associated with the respective location – for example, 103 people are located in the museum.

Monitoring facilities are responsible for detecting changes and updating the model throughout the system's operation. Subsequently, verification facilities evaluate system requirements (e.g. R1-R3 of Section II). When the model is updated, analysis can be triggered again. The rate where this occurs depends on the kind of system considered as well as deployment particulars. In the following section, we discuss a significant case of analysis, representing formal spatio-temporal verification of the exemplar requirements R1-R3. Furthermore, we outline how such verification can be deployed on the cloud, serving requests for verifying requirements on behalf of IoT devices at system runtime.

V. Spatio-Temporal Analysis

As we observed, at system runtime, requirements validation may need to take place over a model populated with contextual information through monitoring. In our running example, formal verification of R1-R3 may be triggered upon the populated model. Specifically for evaluation of R3, we further require knowledge of the progression of the system state, as a sequence – how the runtime values change in each time step, yielding a runtime, global trace. Such a trace is necessary for the spatio-temporal verification inherent to check satisfaction of R3 with respect to a monitored trace at runtime.

Given the spatial model $\langle L, \mathbf{W} \rangle$, we define a *runtime spatio-temporal trace* as the function $\vec{x} : L \times \mathbb{T} \to D^n$ that associates each location $\ell \in L$ and time step $t \in \mathbb{T}$, a vector of values $\vec{x}(\ell, t) = (\nu_1, \ldots, \nu_n)$, referred to as *signal values*. This means that the trace describes the evolution in time of a number of variables in each location of the spatial model. In our running example, $\vec{x}(\ell, t) = (\nu_{type}, \nu_{IoT}, \nu_{people})$, where ν_{type} records the type of node ℓ (landmarks such as museums, metro stops or parks), ν_{IoT} is a Boolean describing the presence or not of a taxi in ℓ and ν_{People} counts the number of people in that landmark.

Given a spatial model and a spatio-temporal trace over that model we can use a spatio-temporal model checker [21], [30], [31] to specify and monitor interesting properties of the system. A spatio-temporal model checker is a tool that uses formal languages, called spatio-temporal logics, to specify the dynamics of a system and then apply automatic procedures (monitoring algorithms) to verify if the system satisfies or not such properties. These tools extend classical *temporal model-checking* [32] to explicitly take *space* into account. Spatio-temporal logics combine *atomic propositions* via a set of operators: the standard Boolean operators (e.g. $\vee$, $\neg$, $\rightarrow$,...), spatial operators that specify configurations of space, and temporal operators, to reason about temporal evolution. When a spatio-temporal model checker is used, *requirements* are specified via a spatio-temporal logic. A model checking procedure can then be used to identify the set of locations where the requirements are violated. The system satisfies the requirements when no location violates them.

In the following, we use the *Spatio-Temporal Reach and Escape Logic* (STREL) [21], and we describe the relevant features of the logic through a number of examples needed to specify the requirements described in Section II. For a complete formal treatment and semantics of the logic, the interested reader is referred to [21]. The monitoring tool is available at [33], where some examples of properties verification over our model are also provided. In STREL, *atomic propositions* consist of *Boolean expressions* interpreted over signal-values. For instance, location ℓ satisfies the *atomic proposition* $taxi$ whenever there is a taxi at ℓ. Hence, given the spatio-temporal signal $\vec{x}$, formula $taxi$ is satisfied at any location ℓ and for any t such that $\vec{x}(\ell, t) = (\nu_{type}, \nu_{IoT}, \nu_{people})$ and $\nu_{IoT} = \top$. Instead, location ℓ satisfies the *atomic proposition* $Label_i$ whenever the label of location ℓ is equal to $Label_i$, e.g. formula $hospital$ is satisfied at any location ℓ and for any t such that $\vec{x}(\ell, t) = (\nu_{type}, \nu_{IoT}, \nu_{people})$ and $\nu_{type} = hospital$.

The first property (R1) can be described by combining the atomic propositions $Taxi$ and $Hospital$ with the implication ($\rightarrow$) and the spatial operator *somewhere* ($\diamondsuit_f \phi$):

$$\phi_{availability} = hospital \rightarrow \diamondsuit_{w<10}\ taxi.$$

A location ℓ satisfies $\diamondsuit_{w<10}\ taxi$ whenever at a distance $w < 10$ there exists another location ℓ' satisfying $taxi$. In this case the distance between ℓ and ℓ' is computed as the sum of the weights of the edges in the shortest path from ℓ to ℓ'. We will see below that different kind of *bounds* can be combined to compute the distance among locations. The meaning of the whole property $\phi_{availability}$ is that the somewhere property has been true in all the locations labeled as hospitals. Notice how this property is satisfied, since the (single) hospital in Figure 3 has a taxi at a metro stop 9 minutes away.

To describe spatial properties, STREL provides a *reachability* operator ($\phi_1 \mathcal{R}_f \phi_2$) that can be used to specify that a location satisfying ϕ_2 can be reached within

a given distance bound while only locations satisfying ϕ_1 are traversed. For instance, the formula $((bus\_stop \vee metro\_stop)\mathcal{R}_{(w\leq 10)}main\_square)$ indicates that we can reach a *main square* within 10 minutes while traversing only *bus or metro stops* ($bus\_stop \vee metro\_stop$).

The second property (R2) can be specified by the formula:

$$\phi_{route} = taxi \rightarrow taxi\ \mathcal{R}_{\substack{(hop\leq 1)\\ \wedge(w\leq 3)}} \Big((bus\_stop \vee metro\_stop)\mathcal{R}_{(w\leq 10)}main\_square\Big).$$

The meaning of the property is that if in a location there is a taxi than it implies ($taxi \rightarrow$) that from this location we can reach a bus or a metro stop in "less than 3 minutes" ($w \leq 3$) and "traversing a single edge" ($hop \leq 1$)[2] and that from the *bus* or *metro stop* we can reach the *main square* in less than 10 minutes ($\mathcal{R}_{m\leq 10}main\_square$). Notice that the taxi at the metro stop at the right upper corner of Figure 3 is close to the main square, but the property is violated due to the taxi at the museum, which after traversal of the bus stop, the specified time is exceeded.

Both requirements (R1) and (R2) can be specified by using only spatial formulas. To express requirement (R3) *temporal formulas* are also needed. Indeed, STREL provides standard operators borrowed from classical *linear-time temporal logics* [32]. Here, to specify (R3) we need the *eventually operator* $\mathrm{F}_{[t_1,t_2]}\phi$ stating that the property ϕ will eventually satisfied at a time $t \in [t_1, t_2]$. Requirement (R3) can be specified by the formula:

$$\phi_{people} = (\nu_{People} > 200) \rightarrow \mathrm{F}_{[0,20]}\ taxi.$$

The formula above states that if at time t there are more than 200 people ($\nu_{people} > 200$) in a landmark then *eventually* in a time between t and $t + 20$ minutes a taxi will be at that landmark ($\mathrm{F}_{[0,20]}taxi$). In practice, property verification of the above spatio-temporal properties occurs through the STREL verification procedure, reflected in the STREL model checker.

Note how verification results (e.g. of properties R1-R3) upon some state of the system at runtime can be used to bootstrap planning facilities of the system. This entails devising some counteraction in response to a violation of a property, towards satisfying system objectives. Afterwards, counteractions may be enacted by the system as part of an execution phase (i.e. as per the MAPE loop of Figure 1). Planning actions may for instance instruct taxis to move to specific locations, optimizing their distribution in space according to QoS business goals, or reward/penalize them depending on their alignment with system-wide goals.

Situating Analysis on the Cloud. As IoT devices making up the spatially-distributed system roam inside the space and updating the model kept at runtime as well as other contextual information, verification may need to be invoked for every device. This may need to occur because verification results are needed for the system's business logic or e.g. for deciding some action as part of an IoT device's local planning stage. Since devices are internet connected (i.e. an instance of the IoT), supporting the computationally-intensive model checking operations –often not possible on resource-constrained devices– can be done through cloud facilities.

To this end, the model and reasoning machinery can be materialized as two separate microservices within a cloud-IoT architecture, evaluating requirements such as the R1-R3 previously presented for every IoT device that comprises the system. Stateless components reflecting solely computation (i.e., the STREL procedure for evaluating spatial properties R1-R3), and mostly immutable data (i.e., the graph inferred as per Section III) can form a model checker microservice, which receives the current system state and a property, and verifies if such property holds. Stateful components can form a location-cache service, which provides an interface to the current global state of the space, i.e., the location of the devices at a given point in time. In this manner, those services can be scaled individually, thus able to easier meet demands of unknown request loads at systems' operation, by instantiating them to the cloud – e.g., by means of virtual machines, containers, or serverless functions. The decision among the various deployments should take into account actual needs of the system at runtime regarding scalability, latency and cost [34].

VI. Conclusions and Future Work

Mobile entities comprising IoT systems are usually equipped with radionavigation, providing geolocation information to their software-enabled components. As engineering self-adaptive systems requires composite models of space to be in place, used to observe, evaluate and react to a constantly changing space, in this paper and accompanying artifact we illustrated a technique to infer analyzable models from general trajectories of IoT. Such models can be useful for engineering adaptation and the multiple aspects of MAPE activities. Furthermore, we illustrated how such spatial models may be used specifically for evaluation of complex requirements, which predicate about the structure of space, the spatial distribution of IoT devices, temporal as well as quantitative aspects.

Regarding future work, we aim to investigate spatial model inference at runtime as well as model-specific aspects. The former refers to challenge the assumption that the model can be constructed, stored and evaluated in full at design time based only on past observations. Moreover, providing facilities to construct and perform formal analysis on partial models can be highly beneficial. Model-specific aspects such as introducing uncertainty and probabilities (e.g., of IoT device presences) are highly applicable. We further plan to extend spatial model inference to include aspects pertinent to quantitative [28] and metric [35] verification.

Acknowledgments

Research partially supported by the Research Cluster SmartCT at TU Wien, the Austrian National Research Networks RiSE/ShiNE (S11405) and ADynNet (P28182) of the Austrian Science Fund (FWF).

[2]The bound hop is computed by counting the number of edges in a path.

REFERENCES

[1] 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*, pages 38–48, 2017.

[2] European Space Agency. Galileo begins serving the globe, 2016. Retrieved from: http://www.esa.int/Our_Activities/Navigation.

[3] Yu Zheng and Xiaofang Zhou. *Computing with spatial trajectories*. Springer Science & Business Media, 2011.

[4] Nam Thanh Nguyen, Dinh Q Phung, Svetha Venkatesh, and Hung Bui. Learning and detecting activities from movement trajectories using the hierarchical hidden markov model. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 955–960. IEEE, 2005.

[5] Mikołaj Morzy. Mining frequent trajectories of moving objects for location prediction. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 667–680. Springer, 2007.

[6] Maria Luisa Damiani, Fatima Hachem, Hamza Issa, Nathan Ranc, Paul Moorcroft, and Francesca Cagnacci. Cluster-based trajectory segmentation with local noise. *Data Min. Knowl. Discov.*, 32(4):1017–1055, 2018.

[7] Jingyu Chen, Ping Chen, Qiuyan Huo, and Xuezhou Xu. Clustering network-constrained uncertain trajectories. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, volume 3, pages 1657–1662. IEEE, 2011.

[8] Monica Wachowicz, Rebecca Ong, and Chiara Renso. Tailoring trajectories and their moving patterns to contexts. In *Geographic Information Science at the Heart of Europe*, pages 285–303. Springer, 2013.

[9] Maria Luisa Damiani and Fatima Hachem. Segmentation techniques for the summarization of individual mobility data. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, 7(6), 2017.

[10] Zhixian Yan. Traj-arima: A spatial-time series model for network-constrained trajectory. In *Proceedings of the Third International Workshop on Computational Transportation Science*, pages 11–16. ACM, 2010.

[11] Zhixian Yan, Dipanjan Chakraborty, Christine Parent, Stefano Spaccapietra, and Karl Aberer. Semantic trajectories: Mobility data computation and annotation. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 4(3):49, 2013.

[12] Christos Tsigkanos, Timo Kehrer, and Carlo Ghezzi. Architecting dynamic cyber-physical spaces. *Computing*, 98(10):1011–1040, 2016.

[13] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[14] Christine Parent, Stefano Spaccapietra, Chiara Renso, Gennady Andrienko, Natalia Andrienko, Vania Bogorny, Maria Luisa Damiani, Aris Gkoulalas-Divanis, Jose Macedo, Nikos Pelekis, Yannis Theodoridis, and Zhixian Yan. Semantic trajectories modeling and analysis. *ACM Comput. Surv.*, 45(4):42:1–42:32, August 2013.

[15] Maria Luisa Damiani, Hamza Issa, Giuseppe Fotino, Marco Heurich, and Francesca Cagnacci. Introducing 'presence' and 'stationarity index' to study partial migration patterns: an application of a spatio-temporal clustering technique. *International Journal of Geographical Information Science*, 30(5):907–928, 2016.

[16] Maria Luisa Damiani, Hamza Issa, Ralf Hartmut Güting, and Fabio Valdés. Symbolic trajectories and application challenges. *SIGSPATIAL Special*, 7(1):51–58, 2015.

[17] Christos Tsigkanos, Liliana Pasquale, Carlo Ghezzi, and Bashar Nuseibeh. On the interplay between cyber and physical spaces for adaptive security. *IEEE Trans. Dependable Sec. Comput.*, 15(3):466–480, 2018.

[18] Christos Tsigkanos, Liliana Pasquale, Claudio Menghi, Carlo Ghezzi, and Bashar Nuseibeh. Engineering Topology Aware Adaptive Security: Preventing Requirements Violations at Runtime. In *Proc. of the 22nd Int. Requirements Engineering Conf.*, pages 203–212, 2014.

[19] 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*, pages 1–7. ACM, 2016.

[20] 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*, pages 13–18, 2018.

[21] Ezio Bartocci, Luca Bortolussi, Michele Loreti, and Laura Nenzi. Monitoring mobile and spatially distributed cyber-physical systems. In *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2017, Vienna, Austria, September 29 - October 02, 2017*, pages 146–155, 2017.

[22] Natalia Adrienko and Gennady Adrienko. Spatial generalization and aggregation of massive movement data. *IEEE Transactions on visualization and computer graphics*, 17(2):205–219, 2011.

[23] Open Street Map. http://wiki.openstreetmap.org/, 2017.

[24] Otakar Cerba, Raitis Berzins, Karel Charvat, and Tomas Mildorf. Smart poi: Open and linked spatial data. In *EGU General Assembly Conference Abstracts*, volume 18, page 12272, 2016.

[25] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems*, pages 99–108. ACM, 2010.

[26] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 316–324. ACM, 2011.

[27] Vincenzo Ciancia, Diego Latella, Michele Loreti, and Mieke Massink. Specifying and verifying properties of space. In *Theoretical Computer Science*, pages 222–235. Springer, 2014.

[28] Laura Nenzi, Luca Bortolussi, Vincenzo Ciancia, Michele Loreti, and Mieke Massink. Qualitative and quantitative monitoring of spatio-temporal properties. In *Runtime Verification*, pages 21–37. Springer, 2015.

[29] Accompanying datasets, models and open source implementation. http://dsg.tuwien.ac.at/staff/ctsigkanos/amelia, 2018.

[30] Laura Nenzi, Luca Bortolussi, Vincenzo Ciancia, Michele Loreti, and Mieke Massink. Qualitative and quantitative monitoring of spatio-temporal properties with SSTL. *Logical Methods in Computer Science*, 14(4), 2018.

[31] Vincenzo Ciancia, Diego Latella, Mieke Massink, Rytis Paškauskas, and Andrea Vandin. A tool-chain for statistical spatio-temporal model checking of bike sharing systems. In *International Symposium on Leveraging Applications of Formal Methods*, pages 657–673. Springer, 2016.

[32] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, Cambridge, Mass., 2008.

[33] MoonLight: a light-weight framework for runtime monitoring. https://github.com/Quanticol/MoonLight, 2019.

[34] Luciano Baresi, Danilo. Mendonca, Martin Garriga, Sam Guinea, and Giovanni Quattrocchi. A unified model for the mobile-edge-cloud continuum. *ACM Transactions on Internet Technology*, 2018. In Press.

[35] Haiying Sun, Jing Liu, Xiaohong Chen, and Dehui Du. Specifying cyber physical system safety properties with metric temporal spatial logic. In *2015 Asia-Pacific Software Engineering Conf.*, pages 254–260. IEEE, 2015.