Digital Twins and Semantic Data Fusion for Security in a Healthcare Environment[⋆]

Barry Norton[1] and Mathias Jes Normann[1]

Milestone Systems A/S, Denmark bno@milestone.dk

Abstract. While the concept of digital twin, and the use of semantic technologies, is already established in the field of IoT, and while the co-existence of such other sensor devices with IP-enabled video cameras is common, this combination, in so-called Video IoT, has yet to see widespread adoption of these technologies. In this paper we discuss our 'in use' context in managing all of these devices, and our experience in building a prototypical implementation of digital twins using semantic technologies.

Keywords: Video · IoT · Digital twin · Semantic technologies.

1 Introduction

The introduction of the IP-enabled digital video camera, commonly called IP camera, by Axis Communications in 1996 [1] enabled video management systems (VMSs) to be built in software. VMSs allow a number of core functions: live viewing of video streams from multiple camera sources; the application of a policy on when to record video; and, the viewing of previously-recorded video streams. Modern VMSs expand beyond these core capabilities, as illustrated in Figure 1 [4].

In particular, modern VMSs provide multiple user interfaces which may be 'desktop' (installed), which allows capabilities like driving a video wall, or Web or mobile based — native mobile clients often providing more advanced user interactions, including comprehensive provision for searching over existing video recordings. Furthermore, modern IP cameras allow a high degree of software-based configuration, both with analogues to traditional cameras — i.e. aperture, shutter speed, etc. — and digital specific video streams characteristics — such as resolution and codec / compression targets. In fact a single camera might produce multiple streams with different settings for this latter class. Furthermore, for an entire class of cameras, commonly called 'PTZ cameras', there is continuous operational control of motorised pan and tilt, as well as zoom. These settings motivate having a database of device management settings, as well as a media database, containing the actual recordings.

[⋆] Supported by the EU Horizon 2020 project 'SAFECARE: Integrated cyber-physical security for health services', grant agreement number 787002.

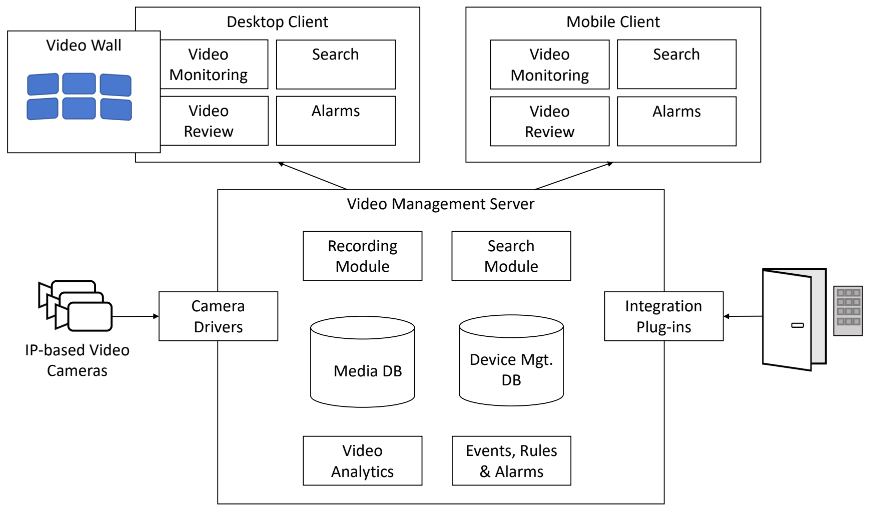


Fig. 1. Generic VMS Architecture

The next common feature of modern VMSs is some provision for rules and events via which they are triggered. An in-built source of events might be motion detection within a video stream, so that the rule can trigger recording of the stream; in this way storage is not wasted on video in which no motion occurs. Some more sophisticated level of video analytics might also provide events that trigger rules: for instance, license plate recognition might trigger the recording of vehicles. External devices might also trigger events via integration plug-ins.

One of the most successful commercial VMS offerings is XProtect [7], produced by Milestone Systems. XProtect is an open platform, which makes available a number of APIs for extension. In the following we will use XProtect as an example, and concentrate on video analytics and access control integration.

XProtect has adopted the open source framework GStreamer [2] to allow extension with video analytics. In this way, integrators can run inference on video streams using off-the-shelf models for deep learning-based object detection, such as YOLOv3 [6], and stream the results to object tracking solutions. Such a solution creates first a classification of objects — for instance detecting people and vehicles of different types — together with bounding boxes within video frames, but also tracks of the movement of objects across video frames.

One of the key device integrations in XProtect is with access control systems, which control physical access to areas of buildings. Figure 2 illustrates in more detail how an access control system may be used in a door. What might colloquially be called a 'lock', or more properly a 'strike', controls electronically when a door can be opened. A door controller unit contains the logic to control the strike, and this is connected to a reader, which may simply be an RFID card reader or PIN pad or, commonly, a combination thereof. When a card is used to identify a permitted person, the door controller can operate the strike to allow the door to be opened.

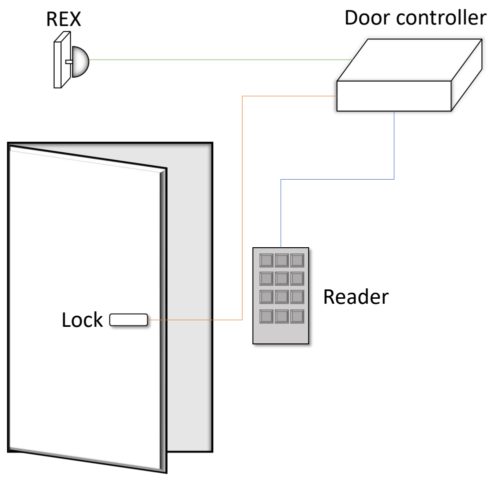


Fig. 2. Access Control System

In some configurations, a reader is used on both sides of the door. In others, a simple REX (request-exit) button is used, in which the operator does not need to identify themselves with a card or PIN, as exit is allowed for all. In more sophisticated set-ups the door might be equipped to automatically open, or ‘speed gates’ might be employed, which have both physical and digital means to prevent — or at least provide an alarm in the case of — ‘tailgating’, that is a second unauthorised person gaining access by following closely behind an authorised user.

In this paper, we will sketch a combination of the above technologies in which access control on a simple door is combined with video analytics, specifically object detection and tracking, to provide a means for tailgating detection. Further, a knowledge graph that contains information on the layout of a hospital building and the devices deployed in it, based on a set of ontologies that also allow the representation of events based thereon in dynamic scenarios, will be shown to allow inference of the vulnerabilities when such situations occur. An integration with fire sensors will also be included, which show escalation of the scenario of attack.

These scenarios are defined by the collaborative project SAFECARE[1], specifically by three hospital consortium partners, the Assistance Publique — Hôpitaux de Marseille (coordinator of the project), the Academisch Medisch Centrum in Amsterdam and ASLTO5 (Azienda Sanitaria Locale - Chieri - Carmagnola - Moncalieri - Nichelino) — hospitals in the Piedmont region of Italy. While trials of the project’s results are planned with all three hospital partners, an interim test platform will be housed at the Instituto Superior de Engenharia do Porto, as illustrated in Figure 3.

[1] https://www.safecare-project.eu/

Test Platform:
UNREAL ENGINE
CENTRAL HOSPITAL
SAFECARE
Integrated cyber-physical security for health services
Pilot Sites:
AP-HM
Assistance Publique Hôpitaux de Marseille
A.S.L. TO5
amC
Academisch Medisch Centrum
Universiteit van Amsterdam
Marseille
Turin
Amsterdam

Fig. 3. SAFECARE Test Sites

This part of the test platform for the SAFECARE project is built in Unreal Engine [9]; although this is primarily a game engine, it has a long history of use in building simulation [3] and in machine learning [5]. Within an off-the-shelf hospital model, we have added models for virtual cameras, virtual fire detectors (smoke and heat type) and virtual access control, as shown in Figure 4. Furthermore, we have written generic and reusable extensions to Unreal such that these device models appear to the XProtect VMS as real devices, as shown on the left-hand side of Figure 5. In the case of cameras these provide live video feeds from the point of view of the model instances. In the case of access control and fire sensors these are true digital twins to hardware devices and respond, within the virtual world, to the events produced by the real world counterparts.

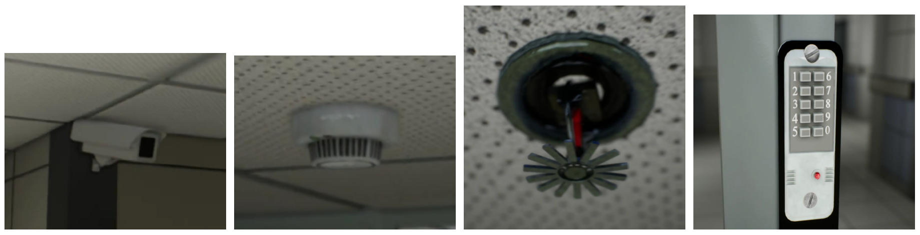


Fig. 4. Virtual Devices (Camera, Smoke / Heat Detector, Access Control Reader)

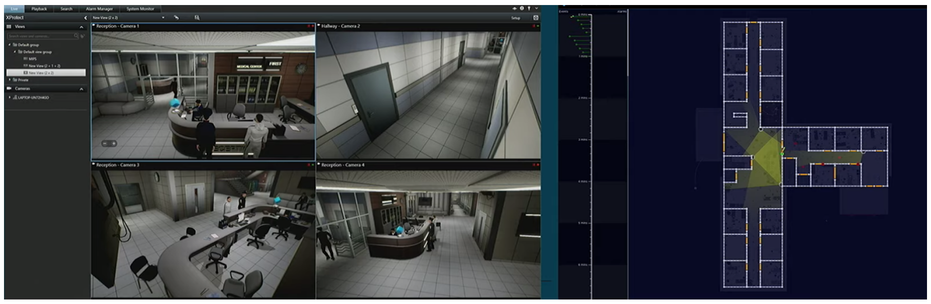


Fig. 5. Demonstration Systems (XProtect Integration and Novel Map UI

The right-hand of Figure 5 shows the novel map-based interface built for demonstration of this work.

In the remainder of this paper we will first introduce the ontologies used to build this system, and then sketch the cloud-based architecture via which the system was created.

2 Ontologies

In order to locate devices within a building, simulated or real, it is natural that we introduce a building ontology, which is shown in Figure 6. While this ontology introduces concepts of *Building* and *Floor*, further structure is left to the concept *Location*, which is defined in the Grid Ontology, shown in Figure 7; this concept is subclassed to define *Zone*s and *Intersection*s. In a building context, Zones describe areas such as rooms and hallways, while Intersections are those junctures between Zones, which may be controlled for instance by physical access control.

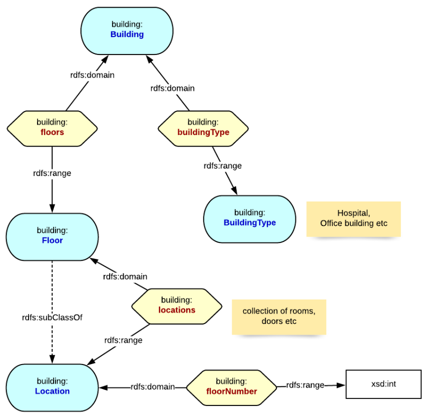


Fig. 6. Building Ontology

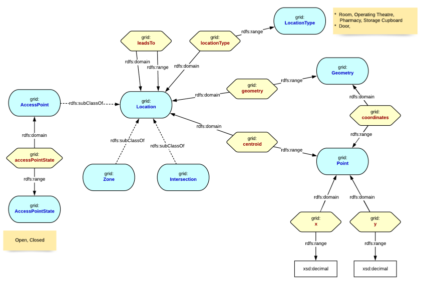


Fig. 7. Grid Ontology

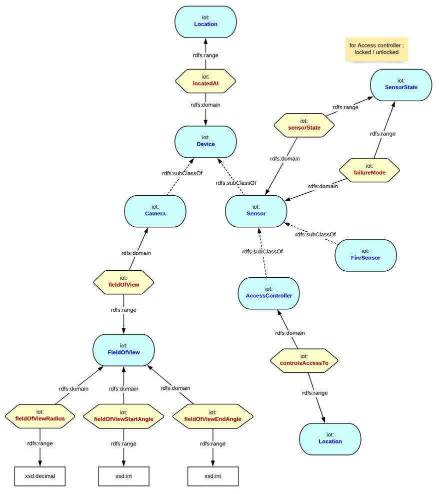


Fig. 8. IoT Ontology

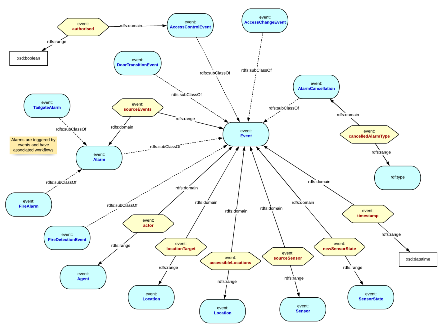


Fig. 9. Events Ontology

It is worthy of note that the Grid Ontology was not defined for this building-based use case, but originated in the description of traffic networks, where Intersections are literal traffic intersections, and these are controlled by traffic lights.

In a similar way, the IoT Ontology shown in Figure 8 is intended to represent a whole array of sensor-type devices, which in another context might be traffic sensors like inductive loops. Two characteristics of the IoT ontology are of particular relevance. First, those subset of devices modelled by the *Camera* class have a *FieldOfView* property, which is illustrated by four yellow triangles on the map in Figure 5, representing the field captured by the cameras in the XProtect Smart Client on the left. Second, while all devices have a *locatedAt* relationship with locations, *AccessController*s have a second relationship named *controlsAccessTo*; this is shown for select parts on the floor plan in Figure 10.

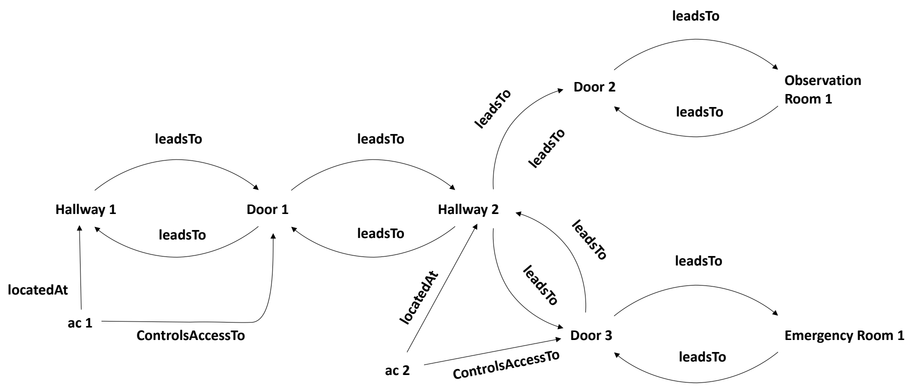


Fig. 10. Graph Representing Segment of Floor Plan

Finally, the Events Ontology, shown in Figure 9 is used to model the dynamics of this system, as follows:

– *AccessControlEvent* is used to represent the operation of the access control reader — this carries a property to convey whether the authorisation was successful;
– *AccessChangeEvent* is used to represent the operation of the door sensor associated with the access controller, or of the gate in the case of speed gates;
– *DoorTransitionEvent* is used to represent the event that a human actually transits through a door, and is produced using video analytics (though in the current prototype this is simulated by having an extension to Unreal Engine that produces this event when a person model passes through a door);
– *FireDetectionEvent* is used to represent the operation of the fire detection sensors;
– *Alarm*s are produced by event processing logic, described in the following section, as relate the events that led to the alarm.

All instances of these event types carry a timestamp, and can be related to locations, sensors and actors.

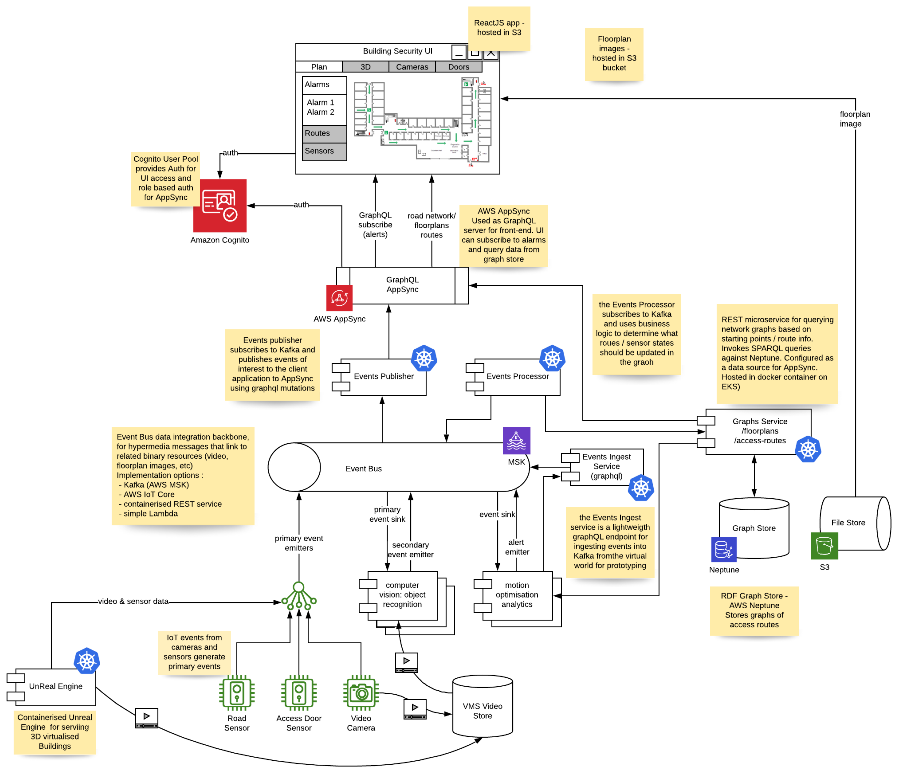
3 Architecture

The software architecture used for this system in shown in Figure 11. This is cloud-based, built on Amazon Web Services, and uses the following services in particular:

– AWS EKS — the Elastic Kubernetes Service — allows the orchestration of containers of custom code, for instance deployed using Docker containers;
– AWS MSK — the Managed Streaming in Kafka service — allows horizontally scalable publish-subscribe messaging of payloads such as JSON messages, via Apache Kafka;
– AWS AppSync — a managed service based on Apache Apollo that allows the formation of GraphQL-based APIs, including the ability to subscribe to receive notifications according to a graph pattern;
– AWS Neptune — a managed graph database service, which supports the W3C RDF and SPARQL standards.

Using these technologies together with the ontologies described in the previous section, we can construct this system according to a uniform graph-based data model. In particular, JSON-LD payloads are used on Kafka via which the sensors, in the prototype proxied via the virtual hospital in Unreal Engine, transmit their changes. Clients can both subscribe to hear particular messages, for instance filtering by sensor or event type, using GraphQL, a combination of technologies that is explored for instance in [8], and on which the W3C has just initiated a working group[2].

[2] https://www.w3.org/community/graphql-rdf/



ReactJS app - hosted in S3

Floorplan images - hosted in S3 bucket

Cognito User Pool provides Auth for UI access and role based auth for AppSync

AWS AppSync Used as GraphQL server for front-end. UI can subscribe to alarms and query data from graph store

the Events Processor subscribes to Kafka and uses business logic to determine what roues / sensor states should be updated in the graoh

REST microservice for querying network graphs based on starting points / route info. Invokes SPARQL queries against Neptune. Configured as a data source for AppSync. Hosted in docker container on EKS)

Events publisher subscribes to Kafka and publishes events of interest to the client application to AppSync using graphql mutations

Event Bus data integration backbone, for hypermedia messages that link to related binary resources (video, floorplan images, etc)
Implementation options :
- Kafka (AWS MSK)
- AWS IoT Core
- containerised REST service
- simple Lambda

the Events Ingest service is a lightweigth graphQL endpoint for ingesting events into Kafka fromthe virtual world for prototyping

RDF Graph Store - AWS Neptune Stores graphs of access routes

IoT events from cameras and sensors generate primary events

Containerised Unreal Engine for serviing 3D virtualised Buildings

Fig. 11. Architecture

The complex event processing involved in inferring tailgating from constituent events is illustrated in Figure 12. In particular the sequence of a single authorisation in the form of an AccessControlEvent, followed by two DoorTransition-Events, without an interleaved AccessControlEvent leads to Events Processing to raise a TailgatingAlarm and publishing this back to the Kafka topic.

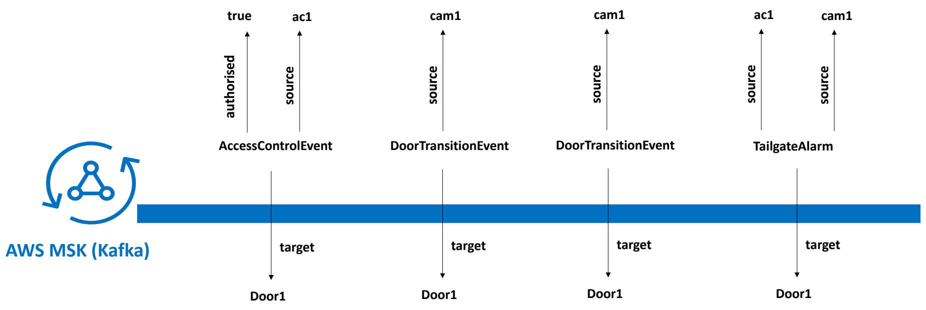


Fig. 12. Events Making Up a Tailgating Instance

By querying the floor plan graph, according to the dynamic information on state, we can consequently determine a reachability graph for the tailgater. Similarly, since the state of doors change in the event of a fire alarm, allowing free access due to the potential need to escape fires, this reachability result can be updated in the event of fire, as shown (reachability indicated in red) in Figure 13.

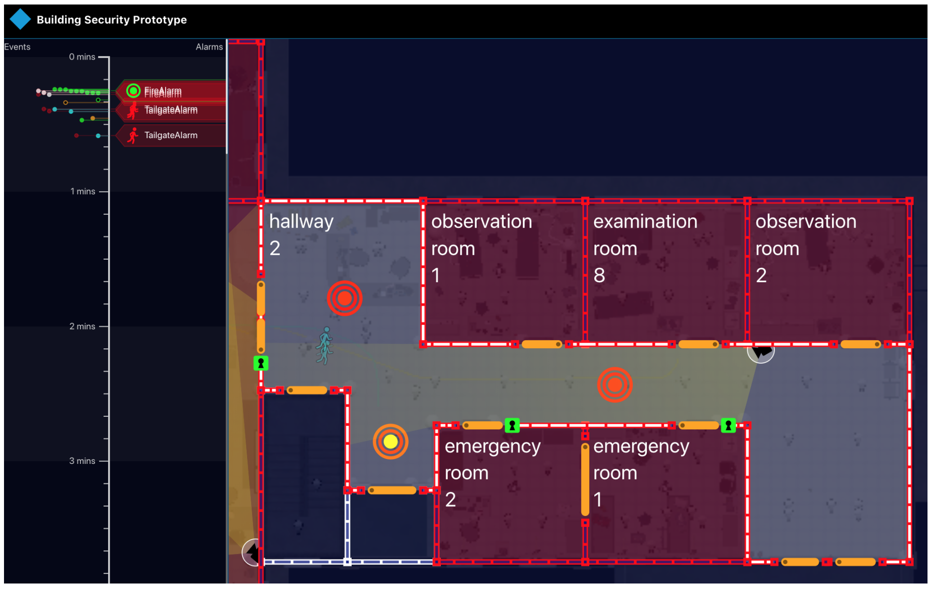


Fig. 13. Reachability Following Tailgaiting in the Event of Fire

4 Conclusion and Further Work

We have shown how an ontology-based approach allows both static information on the situation of security devices within a hospital environment, and the dynamics of these devices, to be modelled. We have shown further that an architecture based on modern, horizontally-scalable cloud technologies can be used to realise a useful implementation. We have illustrated the use of digital twins for both access control devices and fire sensors for simulation within this framework.

Work is already underway in ontology-based representation of the full set of camera, and other device, configuration based on the XProtect Management Server. In future work we will make the virtual cameras true digital twins, where these can be used as proxies for real hardware cameras.

The approach here would be greatly aided by the convergence of GraphQL and RDF/SPARQL, and we aim to contribute to this effort. Similarly the use of Apache Kafka together with JSON-LD, we feel, is an important enabling approach for scaling the promising approach of digital twins in an IoT context where network connectivity is neither constant nor reliable.

Note

The demonstration is available at https://youtu.be/UYH_YI_31v4?t=906, but the intention is also to give this demonstration at ESWC2020, both during the workshop and at the demo session, for which a shorter paper has been submitted.

Acknowledgements

The authors would like to acknowledge the work of Dennis Jørgensen at Milestone Systems in assembling the complete virtual hospital model and developing its virtual devices, as well as John Madsen and Tong Su at Milestone Systems for their work on ontologies and graph data. We would also like to thank the staff of Data Language for their collaboration on this project, and particularly their insights into the use of GraphQL and cloud architecture.

The virtual hospital made use of:

- 'Modular Hospital', a 3D model developed by Aleksandr Zhdanov, and purchased via Unreal Marketplace.
- 3D models for cameras, fire sensors and access control panels purchased via SketchFab, with the following credits:
  - https://sketchfab.com/Terizmeh
  - https://sketchfab.com/gleb_tihon
  - https://sketchfab.com/michael.diaz029

References

1. Axis Communications AB: Moments that made us (2020), https://www.axis.com/about-axis/history, last accessed 26 February 2020
2. Lima, G.A.F., Santos, R.C.M., Azevedo, R.G.D.A.: Programming multimedia applications in GStreamer. In: de Jesus Lima Gomes, F., de Andrade Lira Rabelo, R., de Salles Soares Neto, C., Willrich, R., Teixeira, C.A.C., de Almeida, J.M., de Carvalho, W.V. (eds.) WebMedia. pp. 19–20. ACM (2016)
3. Mól, A.C.A., Jorge, C.A.F., Couto, P.M.: Using a game engine for VR simulations in evacuation planning. IEEE Computer Graphics and Applications **28**(3), 6–12 (2008)
4. Normann, M., Suciu, G., Mantzana, V., Gkotsis, I., Sachian, M.A., Petrescu, G., Ijaz, H., Norton, B.: Security systems in the healthcare sector. In: Integrated Security of Critical Infrastructures (to appear). Now Publishers (2020)
5. Qiu, W., Yuille, A.L.: Unrealcv: Connecting computer vision to unreal engine. CoRR **abs/1609.01326** (2016)
6. Redmon, J., Farhadi, A.: YOLOv3: An incremental improvement. CoRR **abs/1804.02767** (2018), http://arxiv.org/abs/1804.02767
7. Security Sales & Integration: Milestone systems ranked top global VMS provider for 10th straight year (2020), https://www.securitysales.com/surveillance/milestone-systems-top-vms-provider/, last accessed 26 February 2020
8. Taelman, R., Vander Sande, M., Verborgh, R.: GraphQL-LD: Linked Data querying with GraphQL. In: Proceedings of the 17th International Semantic Web Conference: Posters and Demos (Oct 2018), https://comunica.github.io/Article-ISWC2018-Demo-GraphQlLD/
9. Tavakkoli, A.: Game Development and Simulation with Unreal Technology. A. K. Peters, Ltd., USA, 1st edn. (2017)