Global Dispatcher Interface - Initial Prototype Design

Jonathan Beebe

Jonathan Beebe Ltd., 2 Heap Bridge, Bury, BL9 7HR UK

Keywords: Global dispatcher, Standard Elevator Information Schema, group control, prototype, REST, API, UML.

Abstract. This paper presents an overview of the design and development of a prototype Global Dispatcher Interface (GDI) for the control of a group of lifts. The role of the dispatcher is to assign passenger calls to the optimal lift in a group, as decided by a dispatcher algorithm. The GDI is independent of the underlying algorithm, which may be distributed remotely, and provides a standard means through which all interactions with the dispatcher may occur. To warrant the "Global" appellation the GDI must support any of the currently available, as well as anticipated, call station modes, types and configurations of cars, topology of control equipment and buildings. The analysis and design process follows a recognised Systems Development Life-Cycle, centred on Use Cases in a UML model. Significant diagrams from the model are presented and discussed to illustrate the evolution of the prototype design. The requirements, resulting from analysis of the Use Cases, identify that the GDI design must be compatible with a publish-and-subscribe architecture and a RESTful interface is selected for this purpose. Where possible, the prototype design uses open standards with an emphasis on demonstrating those aspects that are specific to lift system dispatcher operation, while attempting to demonstrate independence from implementation details such as programming language, network protocols, etc. The Standard Elevator Information Schema is particularly relevant and fulfils these objectives. The working prototype, which operates in conjunction with simulated lifts and passengers, is presented as a validation of the design.

ABBREVIATIONS USED

API (Application Programming Interface) - a type of software interface, offering a service to other pieces of software.

CoAP (Constrained Application Protocol) - see [22]

GDI (Global Dispatcher Interface) - an interface that separates the lifts and other equipment of a group from the dispatcher in order to present the dispatcher in standard way - the subject of this paper.

JSON (JavaScript Object Notation) - an open standard file format and data interchange format that uses human-readable text. (see - [23])

REST (REpresentaional State Transfer) - A software architectural style (see [17])

SDLC (Systems Development Life-Cycle) - see [4]

SEIS (Standard Elevator Information Schema) - see [13]

UML (Unified Modelling Language) - a general-purpose modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system. (see [10])

URI (Uniform Resource Identifier) - a unique sequence of characters that identifies a logical or physical resource used by Internet web technologies. (see [24])

1 INTRODUCTION

While definitions of standard group control algorithms have been documented [1], the reality of group control to date is that manufacturers have created proprietary designs that are inextricably linked to their own lift equipment. The result is that it is not possible accurately to compare or predict the effects on performance of different control policies during the design phase of a building, or in advance of a refurbishment of the lifts. The benefit of a standard interface is that it would make it possible to supply the dispatching capability in a component form that could be "plugged" into any group of lifts that conforms to the interface.

Secondly, a dispatcher design which has been configured and validated using simulation can be transferred directly into a physical installation with confidence, if both the simulator and the real lifts use the same standard interface.

As lifts become better integrated with the other services of so-called "smart buildings" and with the introduction of applications that allow passengers to register requests for lift travel via a variety of channels [2] including personal mobile devices, an interface that allows simplified and standardised secure access to the group call assignment mechanism becomes increasingly desirable.

Furthermore, performance and status monitoring capabilities, possibly added as a later enhancement to an initial lift installation, would benefit from a standard interface, which would be consistent across a number of buildings in, for example, a property portfolio.

A previous paper [3] analysed the requirements for a Global Dispatcher Interface (GDI) via which a group of lifts could be controlled. Current trends and possible future developments in lift group controller technology were reviewed so that the identified requirements are sufficiently broad and flexible to avoid the analysis becoming prematurely outdated. The paper presented a structured statement of the requirements that a GDI must satisfy, followed by an analysis of those requirements using the Requirements Capture and subsequent Analysis phases of a light-weight Systems Development Life-Cycle (SDLC) [4]. The outputs of these initial phases of the process are

- Passenger (user perspective) use cases
- Dispatcher (system perspective) use cases
- Requirements catalogue
- Domain object catalogue (defining roles and responsibilities)

These outputs take the form of a UML Model plus supporting report documentation generated from the model and, because of their number and complexity, are published separately from the paper, which can only present the key features and conclusions. The report documents can be found at the project website [5]. The model has been developed and is maintained via a specialized tool [6] which supports the entire SDLC.

A second paper [7] continues the SDLC process with a discussion of the design and development of a functioning prototype. By definition, a software prototype [8] is not intended to be deployed in a live situation serving real users (i.e. passengers, maintainers, managers, etc.), rather it is intended to demonstrate the viability of delivering a variety of key functional capabilities, while other characteristics may be only partially implemented or completely omitted. Further prototypes may be produced, in order to explore other aspects of the GDI. After each prototype evaluation, the software should be archived and at the conclusion of the final prototype the design and development phases should be completely reiterated, but from that point onwards with the additional requirements of security, performance, robustness and cost fully accounted for in the GDI design.

In addition to the GDI itself, the prototype demonstration system consists of:

- a configurable simulator (a commercial product [9]) of passengers and of lift car activity.
- new gateway software that has been developed to provide a more realistic representation of lift and call registration activity, which in real life (as opposed to in a simulation program) are enacted as independent asynchronous activities.

An important point, presented in [3], relates to the preferred use of open standards to provide the generic hardware and software, which are of themselves not specific to lift systems. Thus the discussion can concentrate on those considerations which are specific to lift systems. In response, the GDI prototype sets out to demonstrate the delivery of dispatching functionality supported by an infrastructure built of as many interoperating, heterogeneous and open technology standards (e.g. programming language, network protocol, etc.) as it is practicable to include.

2 GDI ANALYSIS

The SDLC describes the process by which development should proceed, evolving the requirements use cases into a set of more detailed System Use Cases. During the system analysis phase, a detailed description text of each of the system use cases (the use case "story" or "flow") provides the basis for developing a detailed diagrammatic definition of the sequences of interactions that must occur between the collaborating domain objects. Two principal use cases are identified - Assign Call and Cancel Call - and it is important to understand that in a busy lift system, multiple instances of both these use cases will exist concurrently and all at different stages of completion.

The domain objects have names like "Car" and "Landing Call Device" but represent very general concepts rather than specific items of physical hardware - a level of detail that will not be developed until a later stage. Sequence diagrams elaborate the messages passed between the objects as the use case proceeds. A separate sequence diagram [10] is developed for each significant alternative route ("scenario") [11] through the use case (often the result of different outcomes from an If-Then-Else like decision). For example:

> *"When the assigned car arrives at the call origin floor, if the dispatcher has not been informed of the passenger's destination floor the passenger's call is then deleted from the list of current calls. However, if the destination floor is known then the call is retained but its status is changed to "Answered".*

While there is insufficient space to include all of these sequence diagrams, they are available at [5] and an example (Figure 1) is included for illustration. It is then through the elaboration of sequence diagrams that the design phase of the SDLC can be commenced. During the design phase further sequence diagrams are produced, but now showing the collaborations between the software components which will be implemented, rather than abstract domain objects. This paper is concerned with the design of the Global Dispatcher Interface only but the diagrams also consider the operation of the dispatcher itself to ensure that all of the dispatcher requirements are supported by the interface. The full set of design sequence diagrams is maintained at [12].

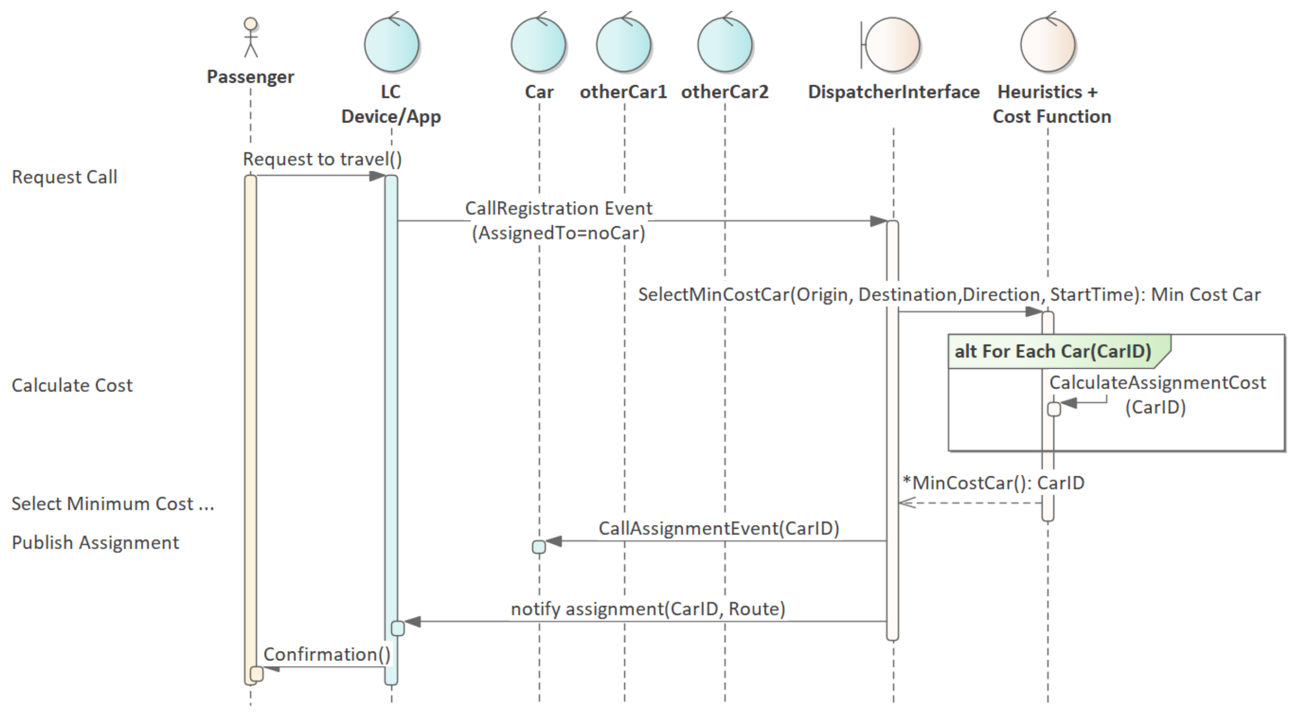
Passenger
LC Device/App
Car
otherCar1
otherCar2
DispatcherInterface
Heuristics + Cost Function
Request Call
Request to travel()
CallRegistration Event (AssignedTo=noCar)
SelectMinCostCar(Origin, Destination,Direction, StartTime): Min Cost Car
alt For Each Car(CarID)
Calculate Cost
CalculateAssignmentCost (CarID)
*MinCostCar(): CarID
Select Minimum Cost ...
Publish Assignment
CallAssignmentEvent(CarID)
notify assignment(CarID, Route)
Confirmation()

Figure 1 Analysis Use Case Sequence Diagram - Assign Direction Call

3 GDI DESIGN

The outputs of the design phase of the SDLC are:

- Sequence Diagrams
- Class library definitions

These will be the necessary inputs for the subsequent software development phase – in this case development of the prototype.

3.1 Design Requirements

During the elaboration of the sequence diagrams some further design requirements are identified.

3.1.1 Standard Elevator Information Schema (SEIS)

The system use cases are described in terms of messages representing *events* that occur within a lift system. These are changes in the information which describes the state of the lift system and are key to ensuring the interoperability of all systems which communicate via the GDI. It is therefore of critical importance that the communicated information content and inter-relationships are specified in a formal and standardised manner. Such a formal specification is provided by a schema, and the necessary schema already exists - the Standard Elevator Information Schema (SEIS) [13][1].

[1]Where this paper makes references to the data types of the SEIS schema, these are indicated by text in CamelCase, which (for the digital format of this document) includes a hyperlink to the definition on the website where the schema is published [13]

3.1.2 Publish and Subscribe Architecture

It is clear from the analysis sequence diagrams that the assignment resulting from a passenger's request to travel must be communicated not only to the passenger via the source of the request (call device/application) but also, most importantly, to the assigned car (and possibly to all other cars as well). Until the assignment is made, the cars are unaware that a call has been registered. So a mechanism is required that will notify the car(s) without them having to continually poll the dispatcher service "just in case". This mechanism is provided by the Publish-And-Subscribe [14] messaging pattern. With this pattern, any number of active elements (cars, call devices, etc.) may request the GDI to publish a list of observable information sources against which they may submit a "subscribe" request. The GDI will subsequently send a message to all subscribers each time an event occurs associated with the information being observed (described by the Observer software design pattern [15]).

3.1.3 Dispatcher Interface as a "Notice-Board"

It has been shown that the system use case sequence diagrams comprise messages being passed to and from the GDI describing information events (conforming to SEIS) and since these events are updates, it is implied that the GDI must maintain a record of the current state (an information model), which is then to be updated. The GDI ensures that the information model is always kept up to date and acts like a central notice-board where the elements of the lift system simply post their current status, under specific subject headings (defined by SEIS). Coupled with the publish-and-subscribe architecture, it becomes like a social-media notice-board where subjects of interest can be "followed" (i.e. subscribed to). This is a very important property of the GDI since none of the lift system elements is assuming to understand or maintain expectations of the operation (or even the existence) of any other element which may receive its messages. In software engineering this characteristic is referred to as 'separation of concerns' [16].

The resulting interface is compatible with any dispatcher algorithm technique from dynamic sectoring to neural networks based on cost functions, and therefore the inevitable debate is avoided about which parameters must be passed in any call to the interface.

This mode of interaction allows an enormous amount of flexibility in the configuration and component architecture of lift systems which may use the GDI. Figure 2 illustrates some of the many possible configuration options.

3.2 Component Architecture Flexibility

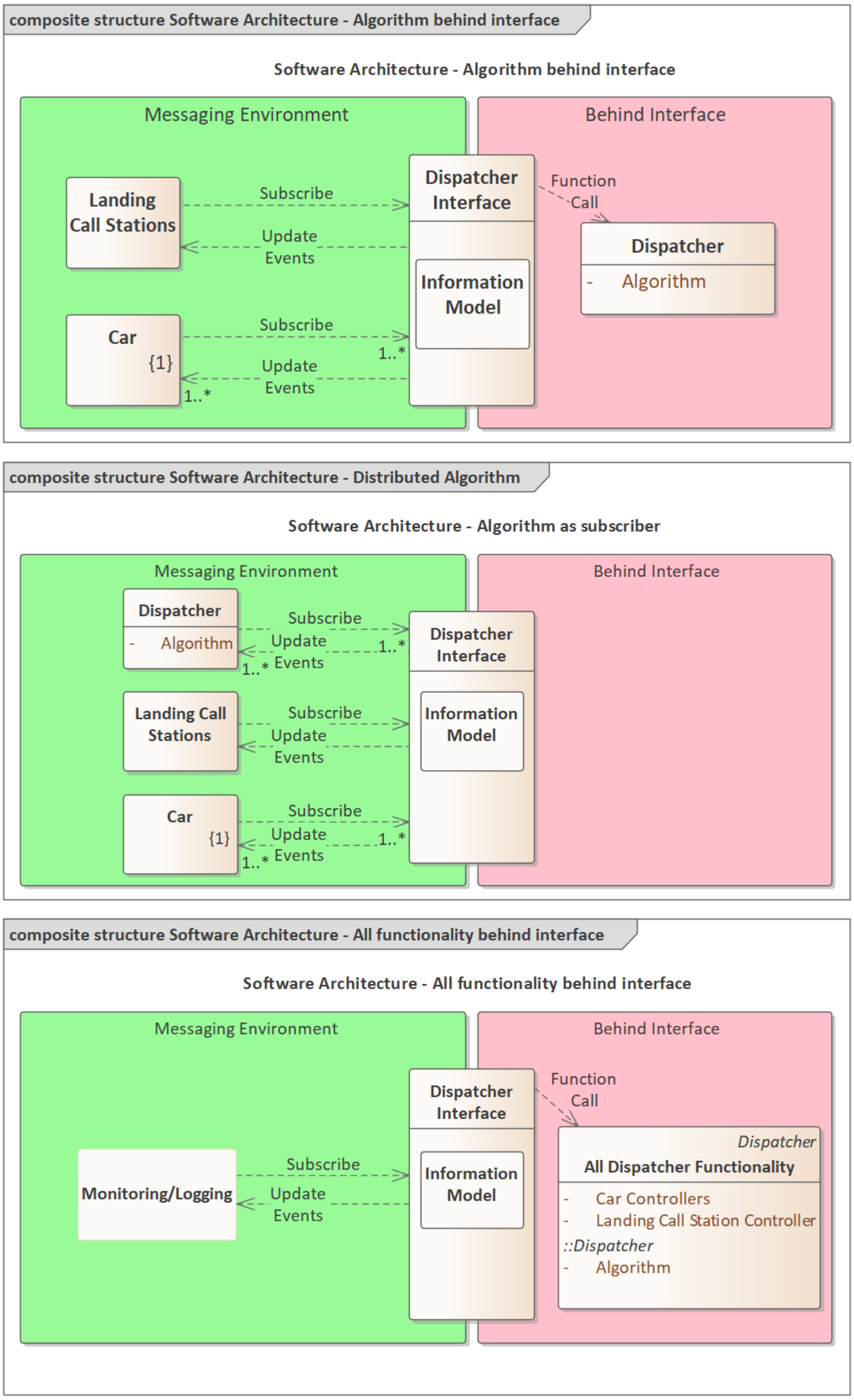

Figure 2 Some GDI architecture options

3.2.1 Configuration option - Algorithm Behind Interface

The use case sequence diagram (Figure 1) shows the cars and landing call devices sending messages to the dispatcher interface with the dispatcher algorithm located "behind" the interface, implying a simple function call from the algorithm to update the information maintained by the dispatcher interface with the assignment result.

3.2.2 Configuration option - Algorithm as Subscriber

In some configurations it may be more appropriate for the algorithm to be implemented simply as another subscriber to the interface (a.k.a. Notice-Board) leaving nothing "behind" the interface.

3.2.3 Configuration option - All Functionality Behind Interface

On the other hand, in some circumstances it may be preferred to have all elements of the lift system control software implemented as a single software component that sits "behind" the interface. In this case the interface would operate simply as a reporting mechanism via which data logging and status monitoring equipment could be connected.

3.3 RESTful Interface

It has already been noted that messages to and from the GDI represent information events that are defined in terms of SEIS. Each message is either placing new information into the information model or reading the current state from the information model. Updates of the information model might simply change the value of an existing element in a list of the information model, e.g. Car4 Direction is now UP (the action of this message is called "PUT"). Alternatively, the event message may create a new element in a list, e.g. a landing call has been registered (the action of this message is called "POST"). In this case a unique identifier is attached to the element so that it can be referenced in future, for example when information is read from the information model (the action of this message is called "GET"). We can conclude therefore that messages do not make calls to the specific functions of the dispatcher, nor any other aspect of operation of passenger lifts. Instead, the same set of standard generic functions (called "methods")

- POST
- PUT
- GET

can be requested from each observable node of the information model.

These functions might be implemented as a 'RESTful' [17] interface which may be implemented in a variety of available programming technologies and languages. The elements of the information model are "resources" in REST terminology. Furthermore, we see that resources which have a multiplicity greater than 1 (i.e. lists) must support queries using the properties and referenced links of the node.

A valuable characteristic of REST is its independence of any network topology (e.g. proxies, gateways, firewalls, etc.), so it is scalable. If required, a single instance of the GDI might therefore support a number of groups of lifts, located in the same building or campus or might even be made available via the Internet as a cloud service. Conversely, one large group of lifts serving many floors will have a heavy computational burden and so might require multiple instances of the dispatcher to be accessed transparently through what appears to be the same GDI.

The GDI prototype complies with all 6 REST constraints – see [17]. Additionally, access permissions to each published node (resource) of the SEIS information model must be considered:

3.3.1 Access Rights

The GDI should implement an overall security policy to restrict access (including subscription) to authorised clients only. Access rights will be established during execution of the Registration use case, but this is not discussed further in the current paper.

3.3.2 Create/Update access restricted to "owned" resources

The ability of a client to create and update resources via the GDI is limited to those resources that are "owned" by the client. Thus a car may update any attributes of its own CarDynamicData or CarStaticData but not those of another car.

Landing call devices may create (POST) a new LandingCall in the list but updating (PUSH) the resulting LandingCall is owned by the dispatcher.

It is a matter of internal design of the dispatcher whether LandingCalls are deleted or retained when their Status becomes Cancelled and should be considered as part of the greater discussion of data logging and retention [2].

4 PROTOTYPE DEMONSTRATOR

The purpose of the prototype is to demonstrate and validate the ability of the GDI design to support the functionality that is particular to the task of lift system dispatching. To achieve this objective, it is necessary to build a complete and realistic environment in which the dispatcher interface can operate. Such an environment needs to have access either to several operational lift installations or to a variety of configurations of simulated groups. The different configurations must allow the prototype to be driven by both direction and destination passenger call stations and to demonstrate different numbers of cars and patterns of passenger demand and floors served. The environment selected for the prototype is illustrated in Figure 3. Implementation and test of the prototype are discussed in greater detail in [7].

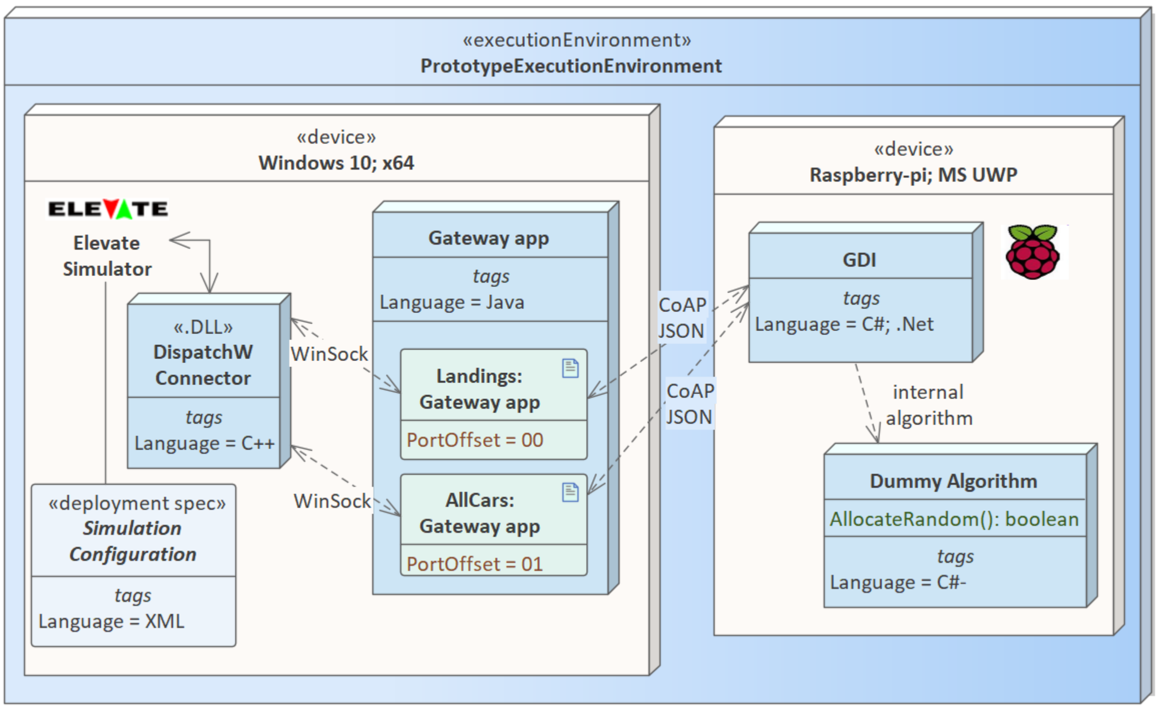

Figure 3 Prototype Implementation

More general technical considerations, such as network performance, security, robustness, etc., are addressed only in as much as it is necessary to achieve a realistic and operational prototype.

In an attempt to demonstrate that the prototype design is not dependent on specific technologies, the selection of network protocols, software frameworks, programming languages, computer hardware and operating system environments has been chosen to be as diverse and heterogeneous as possible and is summarised in the following sub-sections.

4.1 Lift System Simulator

For this prototype, an accurate lift system simulation [9] of lift cars, passengers and call stations has provided a realistic, flexible and permanently accessible solution. Of great importance to the prototype, the dispatcher algorithm, which controls the assignment of landing calls, may be configured as being provided via a system that is external to the simulator. The simulator runs on the Microsoft Windows platform.

4.2 Simulator Connector .DLL

In this case the user-programmable "algorithm" is replaced by connection software ("DispatchW Connector" component in Figure 3 - a Microsoft Windows .DLL). Developed in C++, this doesn't itself contain the algorithm but instead simply splits the information from the simulator, according to its subject matter (cars or landings calls), into separate streams of data events to produce a more realistic environment (using Microsoft sockets API – WinSock [18]).

4.3 Landing Call-station and Car Gateway Application

In a further attempt at realism, a Gateway software application has been developed to undertake a variety of transformations of the data events received from the simulator and similarly for information being returned in the opposite direction. In order to demonstrate the "global" applicability of the GDI we must consider that any part of the lift equipment interacting with it may not be able to produce the necessary information at the appropriate time or in a suitable format. It may be that some manufacturers would integrate such a gateway with their equipment, thereby maintaining the confidentiality of their own intellectual property. Others may prefer to delegate the development of gateway software to a third party. A similarly flexible approach is specified for the connection of lift systems for data monitoring by the National Standards Committee of the People's Republic of China [19].

The gateway is written as a Java application, allowing it to run in a very wide variety of operating environments. It communicates with the GDI using the Eclipse Californium CoAP library [20].

4.4 Global Dispatcher Interface Executable

The prototype GDI itself is written in C# and executes on a separate computing device – a Raspberry Pi running the Windows IoT core on the Universal Windows Platform (UWP).

The GDI software is based on a Windows .Net library implementation [21] of CoAP [22] – Constrained Application Protocol – which is specifically designed to minimise processing and communication demands and which:

- supports a RESTful interface and
- enables discovery of resources through the "/.well-known/" URI
- supports subscription to observable resources
- supports a number of message payload formats including JSON, XML and plain-text, which may be used concurrently and interchangeably in a single implementation.

Whilst there are several available alternatives to CoAP, it was chosen because it offers the above capabilities and because the computing power and network bandwidth available to such an application, probably running in the lift motor room, are likely to be ‘constrained’. However, an eventual commercial product may well employ a different open standard protocol.

4.5 Global Dispatcher Prototype Sequence Diagram

Now that the components of the prototype have been identified, the design process continues by developing the sequence diagrams illustrating the CoAP message interactions between actual elements of software. Each sequence diagram is a refinement of the corresponding analysis use case sequence diagram. For the purposes of this paper, a single example of a design sequence diagram is presented in Figure 4. For the prototype, the interacting software elements are implemented as independent components, but this is not a requirement, and an integrated single executable may be preferable for eventual commercial product implementation.

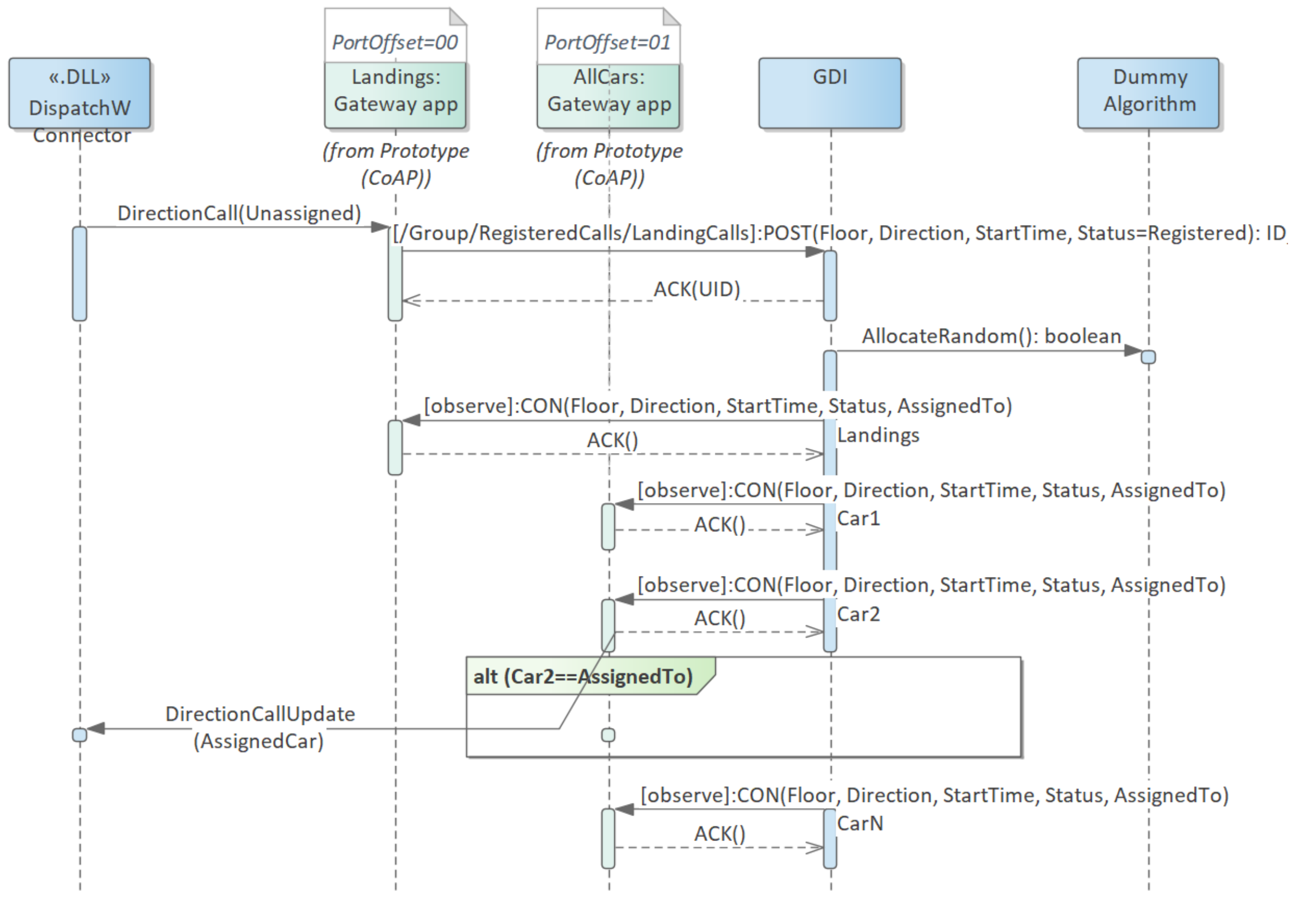
«.DLL» DispatchW Connector
PortOffset=00
Landings: Gateway app
(from Prototype (CoAP))
PortOffset=01
AllCars: Gateway app
(from Prototype (CoAP))
GDI
Dummy Algorithm
DirectionCall(Unassigned)
[/Group/RegisteredCalls/LandingCalls]:POST(Floor, Direction, StartTime, Status=Registered): ID
ACK(UID)
AllocateRandom(): boolean
[observe]:CON(Floor, Direction, StartTime, Status, AssignedTo)
ACK()
Landings
[observe]:CON(Floor, Direction, StartTime, Status, AssignedTo)
ACK()
Car1
[observe]:CON(Floor, Direction, StartTime, Status, AssignedTo)
ACK()
Car2
alt (Car2==AssignedTo)
DirectionCallUpdate (AssignedCar)
[observe]:CON(Floor, Direction, StartTime, Status, AssignedTo)
ACK()
CarN

Figure 4 Prototype - Direction Call Registration and Assignment

5 CONCLUSIONS

An overview of the design of a Global Dispatcher Interface has been presented using standard software design methods. Starting with a set of analysis use cases and associated catalogue of requirements, sequence diagrams illustrating the interactions between software components have been developed to document the design of the GDI. A prototype demonstration environment has been built, implementing the GDI design, interoperating via custom gateway software with a realistic software simulation of passengers and lifts in order to validate the design.

The material presented in this paper is part of an ongoing research and development project and has yet to be implemented commercially. The author welcomes comments and questions regarding possible improvements, errors and omissions. The next iteration of the prototype will explore and validate the potential offered by a distributed dispatcher interface.

5.1 End Note - Security

The introduction to this paper noted that the prototype, which is its subject, does not address general requirements of the GDI that are not specific to the domain of lift systems dispatching. However, it is important to stress in these concluding remarks that security must be placed at the forefront of considerations when developing a commercial product embodying the GDI. Even where the GDI is not connected directly to external networks, it is nonetheless capable of acting as an unintended route for malicious agents to gain access to the lifts or other external systems. Therefore, a full risk assessment must be carried out on a regular basis (extending throughout the product lifetime), and any exposed risks mitigated by regular updates. Refer to *CIBSE Guide D. 2020 Transportation Systems in Buildings* [2] for a more detailed discussion.

REFERENCES

[1] Barney, G.C. and Al-Sharif, L. (2016) Elevator Traffic Handbook, Second edition, Chap.12, Routledge, Abingdon UK, 2016, ISBN 978-1-138-85232-7.

[2] CIBSE-Ch14. (2020). *CIBSE Guide D. 2020 Transportation Systems in Buildings*. Chap 14, The Chartered Institution of Building Services Engineers.

[3] Beebe, J (2018). "*Towards A Global Traffic Control (Dispatcher) Algorithm - Requirements Analysis*", Transportation Systems In Buildings, Vol 2, No 1 (2018), University of Northampton, Available from: http://journals.northampton.ac.uk/index.php/tsib/article/view/147 . (accessed 05-Jun-2022)

[4] SDLC(2021) see "*Software development process*", Wikipedia, Available from: https://en.wikipedia.org/wiki/Software_development_process. (accessed 11-Jul-2022)

[5] Beebe, J (2021), "*Analysis products*"; https://dispatcher.std4lift.info/ (accessed 11-Jul-2022)

[6] Sparx (2021), Sparx Systems Enterprise Architect. Available from: https://www.sparxsystems.com/ (accessed 11-Jul-2022)

[7] Beebe, J (2022). "*Towards A Global Traffic Control (Dispatcher) Algorithm - Interface prototype design*", Transportation Systems In Buildings, Vol 4, No 1 (2022), University of Northampton, Available from: http://journals.northampton.ac.uk/index.php/tsib/article/view/158 (accessed 05-Jun-2022)

[8] Software Prototyping (2021), see Software prototyping, Wikipedia, Available from: https://en.wikipedia.org/wiki/Software_prototyping. (accessed 11-Jul-2022)

[9] Peters Research (2021). Elevate™ traffic analysis and simulation software. Available from: https://www.peters-research.com/index.php/elevate/about-elevate. (accessed 11-Jul-2022)

[10] Alhir, Sinan Si.(1998), "UML in a nutshell"; pp85-94 "*Sequence Diagrams*", O'Reilly & Associates, Inc., Sebastopol CA, USA, 1998, ISBN 1-56592-488-7.

[11] Bitner, K and Spence, I.(2008), "Use Case Modelling", pp196, "*What is a Scenario*", Addison-Wesley, London, 2008, ISBN 02011709139.

[12] Beebe, J (2021), "Design products"; https://dispatcher.std4lift.info/GlobalDispatcher-PrototypeDesign.pdf (accessed 11-Jul-2022)

[13] Beebe, J. (2021)"Standard Elevator Information Schema", https://www.std4lift.info/ .. (accessed 11-Jul-2022)

[14] Publish-Subscribe (2021), *Publish–subscribe pattern*, Wikipedia, Available from:: https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern (accessed 11-Jul-2022)

[15] Gamma, et al (1995) Erich Gamma; Richard Helm; Ralph Johnson; John Vlissides (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley. pp. 293ff. ISBN 0-201-63361-2.

[16] SeparationOfConcerns (2021). *Separation of concerns*. Wikipedia, Available from: https://en.wikipedia.org/wiki/Separation_of_concerns (accessed 11-Jul-2022)

[17] REST (2021), *Representational State Transfer [Internet]*, Wikipedia, Available from: https://en.wikipedia.org/wiki/Representational_state_transfer (accessed 11-Jul-2022)

[18] WinSock (2021), Microsoft, *Winsock Network Protocol Support in Windows*. Available from : https://docs.microsoft.com/en-us/windows/win32/winsock/network-protocol-support-in-windows (accessed 11-Jul-2022)

[19] PRC (2021) National Standards Committee of People's Republic of China. GB/T 24476-2017 - *Specification for internet of things for lifts, escalators and moving walks*. 2018. Available from: https://www.chinesestandard.net/PDF/English.aspx/GBT24476-2017 (accessed 11-Jul-2022)

[20] Californium (2021), Eclipse Foundation, "*Eclipse Californium*". Available from: https://www.eclipse.org/californium/ (accessed 11-Jul-2022)

[21] Waher, P. (2018), *Mastering Internet of Things*, Chap 10 *The Controller*, Packt Publishing Ltd. (www.packtpub.com),ISBN 978-1-78839-748-3.

[22] CoAP (2014), Internet Engineering Task Force (IETF), "*The Constrained Application Protocol (CoAP)*", RFC 7252. Available from: https://tools.ietf.org/html/rfc7252.(accessed 11-Jul-2022)

[23] JSON (2022), *JavaScript Object Notation*, Wikipedia, Available from: https://en.wikipedia.org/wiki/JSON (accessed 11-Jul-2022)

[24] URI (2022), *Universal Resource Identifier*, Wikipedia, Available from: https://en.wikipedia.org/wiki/Uniform_Resource_Identifier (accessed 11-Jul-2022)

BIOGRAPHICAL DETAILS

Jonathan Beebe has more than 40 years' experience of bringing the latest software design and data modelling technologies into the domain of lift system control and monitoring. Following his PhD thesis entitled "Lift Management" (completed at a time when there were virtually no computer-controlled lifts anywhere in the world) he was employed to design and implement software for dispatcher algorithms and single car controllers coupled with a remote performance monitoring system.

Throughout his career Jonathan has maintained an active interest in research, resulting in the publication of The Standard Elevator Information Schema (SEIS) in 2003. SEIS is published under the Creative Commons licence with free and open access to anyone interested.

Jonathan's current research project is developing a Global Dispatcher Interface (GDI), based entirely on the SEIS. GDI looks forward to the era of smart buildings and cities in which vertical transportation systems will play a fundamental role.