Towards A Global Traffic Control (Dispatcher) Algorithm - Interface Requirements Analysis

Jonathan Beebe

Jonathan Beebe Ltd., 2 Heap Bridge, Bury, BL9 7HR UK, jonathan@jonathanbeebe.com

Keywords: Global dispatcher, standard interface, group control, requirements analysis, use case

Abstract. This paper presents an analysis of the requirements of a Global Dispatcher Interface for the control of a group of lifts. The information passed to and from the interface is defined as well as the common processing which will be executed on that information in order to generate the response. Using recognised software development methods, requirements are elicited from a consideration of the significant use cases and the architectural configurations which must be supported by the interface. The analysis concludes by defining the roles and responsibilities of the key objects of the software. A final section proposes relevant standard open technologies that avoid proprietary and potentially incompatible and high maintenance solutions.

1 INTRODUCTION

This paper presents an analysis of the requirements for a Global Dispatcher Interface for controlling a group of lifts, a concept which was earlier proposed by Peters[1]. While definitions of standard group control algorithms have been documented[2], the reality of group control to date is that manufacturers have created proprietary designs that are inextricably linked to their own lift equipment so that it is not possible accurately to compare or predict 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 be 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.

Additionally, 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[3] including personal mobile devices[4], an interface that allows simplified and standardised access to the group call assignment mechanism becomes increasingly desirable.

The objective of the current paper is not to deliver a complete design for such a control system instead a structured statement of the requirements that such a system must satisfy followed by an analysis of those requirements is presented as the initial phase of development of such a system. It is therefore a guide for designers proposing to implement a Global Dispatcher Interface.

Furthermore, it is the standard requirements of the dispatcher *interface* that are analysed here omitting analysis of any specific dispatcher (i.e. the logic that determines which is the best lift in the group to be assigned to answer a call, which is sometimes referred to as the group or traffic "control algorithm"[9]).

A key consideration for this paper, in addition to established conventions of lift system design, is a review of current trends and possible future developments in lift group controller technology so that the requirements identified are sufficiently broad and flexible to avoid the analysis becoming prematurely outdated.

The software requirements analysis process that has been used here forms an early phase of a lightweight Software Development Lifecycle Process (SDLC) and is supported by a UML Modelling tool[5] to undertake:

- Use case identification and analysis[6] for the significant processes which the interface must support

leading to:

- Domain analysis[7] by requirement identification and refinement

which is achieved through analysis of published reference works [9],[2].

The paper starts with an analysis of the operation of a generic group of lifts from the perspective of the travelling passenger. This is presented as the passenger use cases. There follows a section which presents the main dispatcher system use cases, which are the result of analysing the passenger use cases. The analysis concludes with overview definitions of the key collaborating objects, deduced by analysing the dispatcher system use cases. These are necessary for the Global Dispatcher Interface to be globally applicable in a number of identified potential interaction modes. Finally, under a number of topic headings, a discussion is presented of the requirements that will enable a Global Dispatcher Service to operate safely, securely, flexibly and manageably in a networked environment with recommendations that open standards (that are non-specific to the environment of passenger lift systems) are adhered to. Thus, by the conclusion of the paper we have an analysis of the Global Dispatcher Interface, which is a suitable starting point for the design of specific implementations.

In order to present a paper of the size and detail that is suitable for publication, the current document is necessarily a summary of a large body of work consisting of diagrams and documentation relating to a Global Dispatcher Service, which has been collected in the form of a UML model[8]. References are included in this paper to on-line documents which have been generated from the model and which provide more detailed elaboration supporting the concepts presented here. However, it is intended that it should be possible simply to understand the discussion presented here without the need to make cross-reference to such elaborations.

2 SOME RELEVANT TERMS

The paper begins with an overview of some relevant terms in the domain of lift control which makes reference to definitions in earlier publications [9],[10].

Landing call (LC) – a request for travel from a *landing*. This may be for

- a specified destination floor

or simply

- to travel in specified direction.

Although it may be possible to infer a direction of travel from a landing call which only calls a lift to a floor with no information about intended travel, this is not considered relevant to the analysis presented here, because it would thereby be converted into one of the above cases.

The process of registering a landing call may take place on the landing or at a physically remote location (via dedicated hardware or possibly a personal mobile device). It is helpful to group landing call devices in order to take account of the approximate time taken for passengers to travel from one to the other after registering their call. These groups have traditionally been referred to as "risers" because of the cabling used, but in the context of this paper the grouping covers a broad range of configurations according to their distance from the entrance(s) of lifts to which a call might be assigned

Group of lifts – may be several cars, each travelling vertically in an independent dedicated shaft or, as in recently developed examples, multiple cars capable of both horizontal and vertical travel (or a combination), occupying the same shaft concurrently, with the ability to transfer between shafts.

Cars – may carry passengers on one or more decks (stacked vertically) so that passenger arrival and destination floors must be coordinated to minimise or remove the possibility of a car stop where the passengers on one of the decks see no-one enter or leave, which would be frustrating or possibly unnerving.

Algorithm – the intelligence, in whatever appropriate form, which determines which lift will be made responsible (i.e. "assigned") for servicing a passenger's landing call. Often called a "Group Control Algorithm" or "Traffic Control Algorithm"[9]. This intelligence is encapsulated in software that is referred to in the current paper as "Cost Function + Heuristics" or simply the "dispatcher".

Standard Elevator Information Schema (SEIS) [10] – is a standard for communicating **static information**, such as configuration details, **dynamic information** such as current floor and registered calls and **events** such as call registrations and car trips, between all manner of systems and users of passenger lifts. It comprises a set of definitions of complex and simple data types and the structures in which they may be used. This paper makes frequent references to the schema, both for data types of parameters passed in messages and also for the internal data structures of the dispatcher interface, and these are indicated by text in CamelCase which includes a hyperlink to the definition on the website where the schema is published.

Global Dispatcher Service – is a standard, non-proprietary mechanism for assigning a landing call to the lift car most suited to serving that call. It therefore includes an element - the "dispatcher" - which can produce the optimal assignment decision. The **Global Dispatcher Interface** encapsulates the dispatcher and is the route by which all access to the Global Dispatcher Service is made.

3 USE CASE ANALYSIS

This section simply identifies the use cases that are of key relevance to the Global Dispatcher Interface. The complete set of detailed use case documentation, which provides the "Use Case Story" is published separately[11]. The text of each use case story is essential to understanding the operation of the Global Dispatcher Interface. However, only the most significant use cases are included in the current paper for reasons of size.

Users (passengers) interact with the system (Global Dispatcher Interface) in two separate (although loosely interlinked) processes, which are defined as the "Passenger Use Cases":

- Request to travel (Landing Call)

and

- Travel to destination

These processes must be supported by the Global Dispatcher Interface and are described and elaborated using use case analysis.

In each use case the passenger undertakes a different activity and since each use case process depends on the independent and unpredictable actions of individual human beings, we must conclude that these use cases represent concurrent and unsynchronised (asynchronous) processes.

3.1 Use Case Analysis - Passenger Use Cases

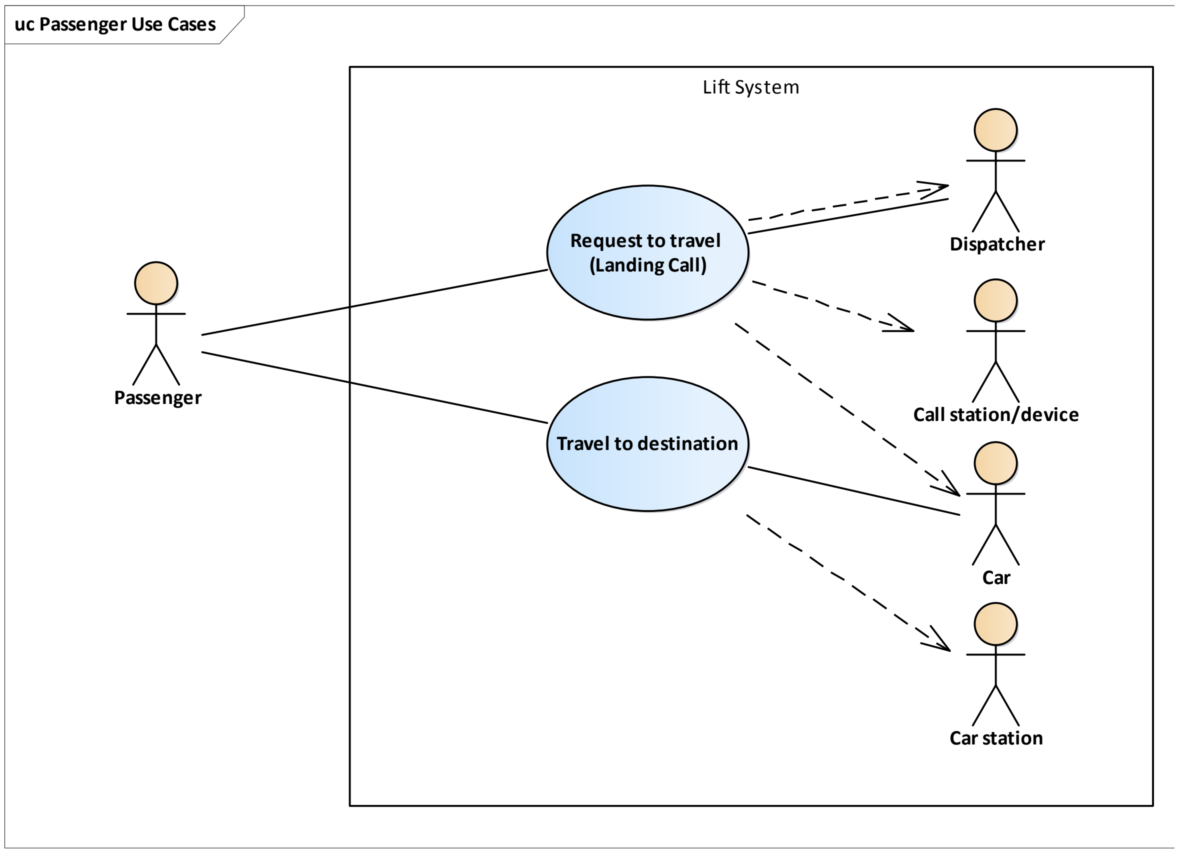


Figure 1 Passenger Use Cases

The Passenger use cases describe the flow of interactions between the passenger and the elements of the lift system to achieve the use case objective. They are a view of the interaction of the lift system with its external environment (see Figure 1).

The main actor of these use cases is "the passenger" but it is important to note that many instances of these use cases may be active concurrently and asynchronously so they may exist in as many different stages of completion. One passenger, having partly completed one of the above use cases may affect the experience of another passenger or be affected by another passenger who is at a different stage of completion of the same or a different use case. This means that other passengers are referenced in the use case text who are not the main actor and should not be considered as actors in the use case in any sense.

The flow of events of each use case in the achievement of its specific goal is described in words by the Use Case Story:

3.1.1 Request to travel (Landing Call) - Use Case Story

3.1.1.1 Main Flow

The passenger approaches a call station or activates an application on a smart device.

The passenger registers a request to travel which includes:

- Origin floor
- Destination direction OR destination floor

Optionally other information may be supplied in the request such as:

- Priority status of the passenger or requested journey
- Number of passengers travelling as a group
- Access security details
- The expected delay before the passenger will be ready to enter the lift

The dispatcher responds via the call station (or smart device) to the passenger to:

- Confirm the request (e.g. illuminated button) if not already activated

plus optionally:

- Inform the passenger which car to travel in

The dispatcher also communicates the assignment directly to the assigned car.

The car responds by adding the assigned call to its travel plan and in due course departs with a destination of the passenger's origin floor.

On arrival at the passenger's origin floor the car cancels the passenger's call and informs the dispatcher of the call cancellation.

The dispatcher indicates the cancellation to the passenger via the call station and optionally other indicator devices (including possibly the smart device, if in use).

The dispatcher records the System Response Time for performance analysis purposes.

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, although the dispatcher retains an expectation that one or more new car calls may be generated as a result of passengers entering the car. However, if the destination floor is known then the call is retained but its status is changed to "Answered".

3.1.2 Travel to destination - Use Case Story

3.1.2.1 Main Flow

The car arrives at the passenger's origin floor and opens its doors to allow:

- any arriving passengers to exit from the car
- the waiting passengers to enter the car.

A delay is initiated for the car doors to stay open in order to allow these transfers.

If the dispatcher already knows the passenger's destination floor it will establish with the arriving car a car call for the destination floor, which may be displayed on the car station.

If the dispatcher does not know the passenger's destination floor, then the passenger registers their destination floor via the car station.

Whichever mechanism is used to register the car call results in a stop at the passenger's destination floor being included in the car's travel plan (although this may already be in place due to the requests of other travelling or waiting passengers).

After allowing passengers to transfer to and from the current stop floor, the car closes its doors and departs for the next destination in its travel plan.

On arriving at the passenger's destination floor the car cancels the car call for that floor and informs the dispatcher.

The car opens its doors to allow the arriving passenger to depart and then any further waiting passengers to enter the car to begin travel to their own destination floors.

3.2 Use Case Analysis - Dispatcher System Use Cases

Now that the Passenger Use Cases have been established it is possible to extract from them the roles and responsibilities of the Dispatcher, namely:

- To monitor and retain the current state of each car (floor position, motion, doors, registered/cancelled calls).
- To accept passenger landing call requests and inform passengers, by various means what response to expect.
- To assign (and it is reasonable to assume possibly to re-assign) landing calls to the most appropriate cars.

From these the main system use cases of the dispatcher can be developed. Figure 2 shows the dispatcher system use cases and the actors which initiate them.
N.B. *The two "Register client" use cases have been inferred since the dispatcher will necessarily need to know what clients are available (which may be a dynamic property). However, these use cases are not felt to be concerned directly with the primary activity of the dispatcher in making call assignments so have been omitted here for brevity, although they are documented elsewhere[11].*

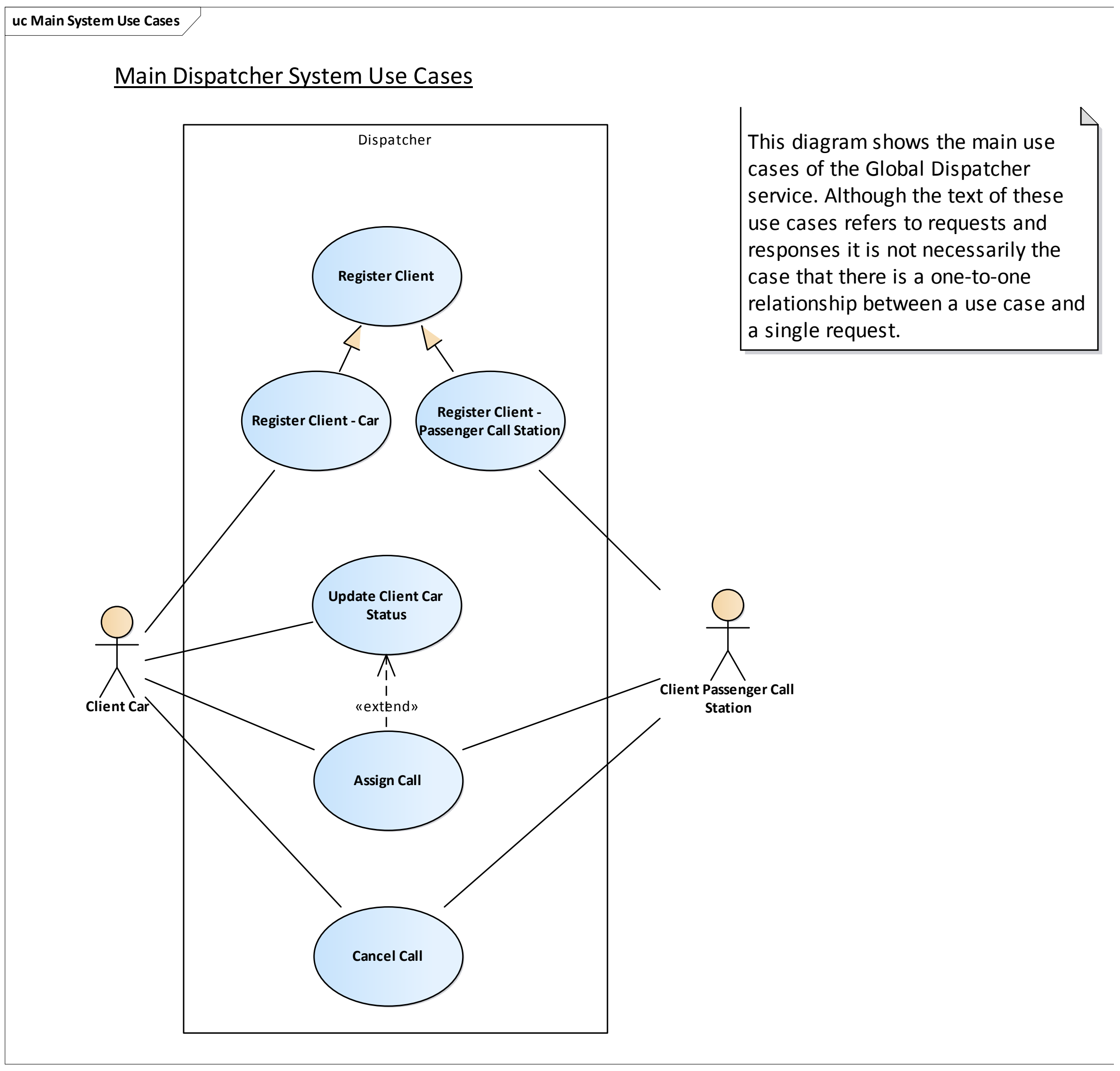

uc Main System Use Cases
Main Dispatcher System Use Cases
Dispatcher
This diagram shows the main use cases of the Global Dispatcher service. Although the text of these use cases refers to requests and responses it is not necessarily the case that there is a one-to-one relationship between a use case and a single request.
Register Client
Register Client - Car
Register Client - Passenger Call Station
Update Client Car Status
«extend»
Assign Call
Cancel Call
Client Car
Client Passenger Call Station

Figure 2 Main Dispatcher System Use Cases

The dispatcher system use cases describe the flow of interactions between the elements of the lift system in order to support the fulfilment of the Passenger use cases. In that sense they are a simple view onto the internal operation of the lift system. (N.B.: *there is not a simple relationship between the Passenger and System use cases*).

The next sub-sections contain the use case story of each of the two most significant dispatcher system use cases:

- Update Client Car Status

and

- Assign Call

N.B. *References to "Car" in the following text are purely abstract and do not relate to the physical car or its control equipment. Similarly for references to the "dispatcher"*.

3.2.1 System Use Case - Update Client Car Status

3.2.1.1 Main Flow

A client car announces a change of its state to the dispatcher by providing details of its dynamic information at the time they occur:

- The ID supplied by the Service in response to the Introduce Client request
- The client's own identifier
- Current Floor
- Direction
- Drive state
- Door state
- Travel Plan
- Shaft
- Currently registered calls (Car, Landing, Park)

All this information may be supplied in a single request (CarDynamicData) or, preferably a sequence of incremental requests as each change of state event occurs (LogEventType). All dynamic data and events are time-stamped to avoid errors due to communication delays.

At the discretion of the specific dispatcher algorithm (not the dispatcher interface) an update of car status may initiate the re-assignment of calls, which is shown by the extends relationship to the Assign Call use case.

3.2.2 System use case - Assign Call

Any client (including a passenger signalling device) may imply a request for a CallAssignment in the form of a CallRegistration event (see Figure 3).. The event includes the following information:

- Call Floor
- Direction or Destination Floor
- Registration time

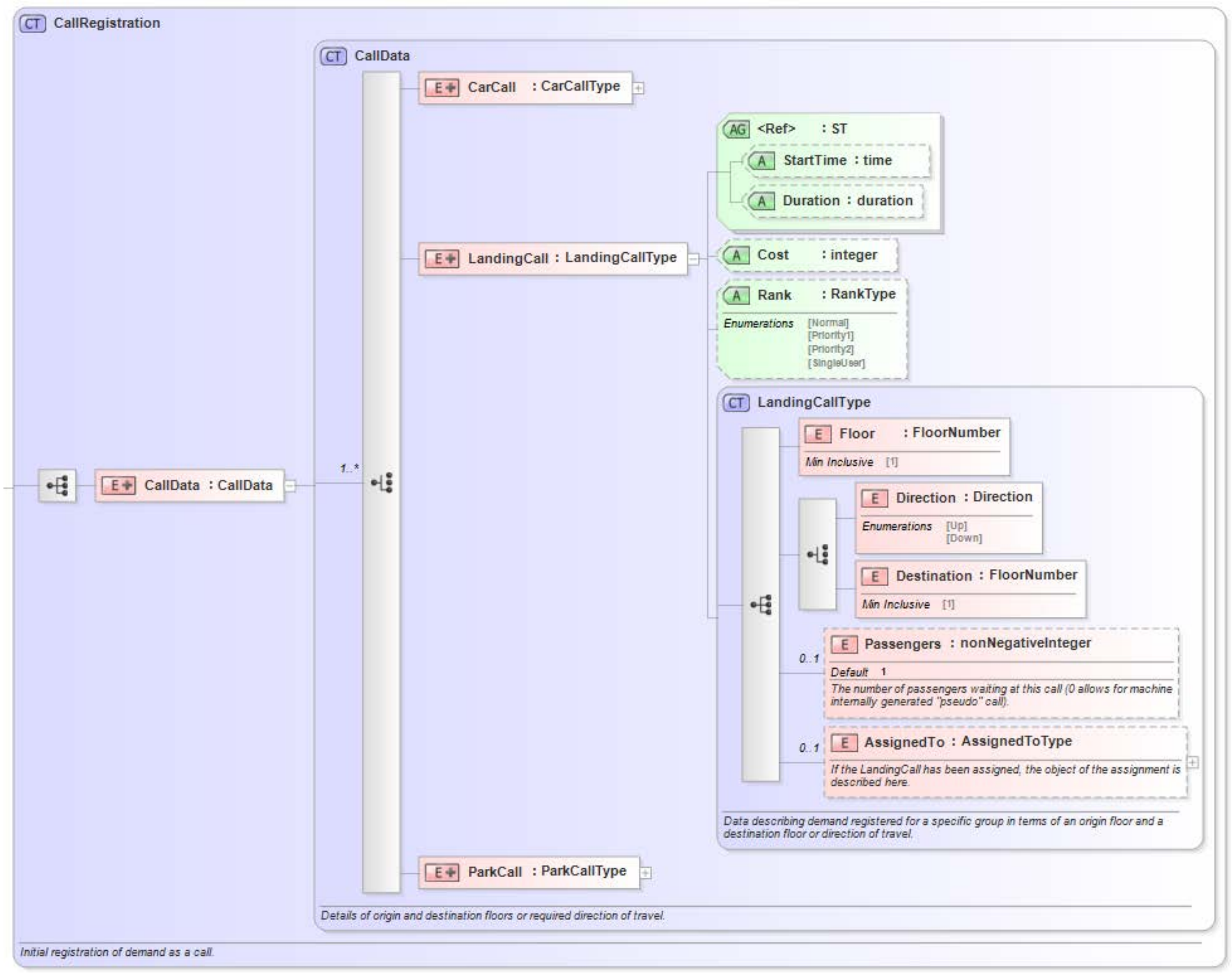

CT CallRegistration
CT CallData
E CarCall : CarCallType
AG <Ref> : ST
A StartTime : time
A Duration : duration
E LandingCall : LandingCallType
A Cost : integer
A Rank : RankType
Enumerations [Normal] [Priority1] [Priority2] [SingleUser]
CT LandingCallType
E Floor : FloorNumber
Min Inclusive [1]
E Direction : Direction
Enumerations [Up] [Down]
E Destination : FloorNumber
Min Inclusive [1]
E CallData : CallData
1..*
0..1
E Passengers : nonNegativeInteger
Default 1
The number of passengers waiting at this call (0 allows for machine internally generated "pseudo" call).
0..1
E AssignedTo : AssignedToType
If the LandingCall has been assigned, the object of the assignment is described here.
Data describing demand registered for a specific group in terms of an origin floor and a destination floor or direction of travel.
E ParkCall : ParkCallType
Details of origin and destination floors or required direction of travel.
Initial registration of demand as a call.

Figure 3 Structure of CallRegistration event

(N.B. *inclusion of StartTime means this request may (re)occur at any time after the initial call registration event and may therefore be a request for a call to be re-assigned or simply a delayed request if no car was available for assignment earlier. In the case of re-assignment the dispatcher will have a record of the currently assigned car and this information may influence the result of the new assignment, though this would be a characteristic of the specific dispatcher algorithm and not of the dispatcher interface.*

At the discretion of the specific dispatcher algorithm (not the dispatcher interface) an Assign Call request may initiate the re-assignment of other previously assigned calls, which will result in multiple executions of the Assign Call use case (i.e. once for each call to be re-assigned).

3.2.2.1 Main Flow

The dispatcher calculates the cost of assigning the call to each of the registered client cars according to its own internal algorithm design. The term "cost" is not restricted to a purely financial cost and may be evaluated in terms of one or more criteria such as:

- waiting time,
- system response time,
- energy consumed,
- etc

as a function of the increase in the value of that parameter after the call has been assigned compared to the cost before it was assigned to the car. The algorithm may include penalties or incentives that are derived from a logical analysis which will modify the simple cost.

If an overriding criterion is included in the algorithm, such as never assigning a call that would cause the car to become overloaded, then that car will be marked as "blocked" (N.B. *this type of logic is later termed a "heuristic"*).

N.B. *Availability for assignment may be determined by a variety of properties such as:*

- *the operating mode of the car*
- *whether the car is able to service the call floor(s)*

However, the availability decision is a characteristic of the specific dispatcher algorithm and not of the dispatcher interface.

The response is broadcast by the dispatcher to all registered clients and includes (but is not limited to):

- the CallRegistration call event updated with the cost of assignment, where cost analysis is made in terms of the cost-function that is specific to the algorithm used by the dispatcher

plus optionally:

- a CallAssignment event where the Assigned To element is populated with the minimum-cost assignment details.
- a TravelPlan (see Figure 4) if the dispatcher is configured to hide assignments so that the car will by-pass an assigned call until it becomes the next landing call for the car.
- if the call was previously assigned to a different car the response will also include a CallDeassignment event.

The Assignment is only considered to be complete after a positive acknowledgement has been received from the designated car.

N.B. *An important conclusion that can be drawn from this use case is that a single CallRegistration event message results in multiple further requests to more than one client of the local dispatcher interface. The secondary messages are requests in their own right and should not be confused, in the context of the local dispatcher interface, with response messages.*

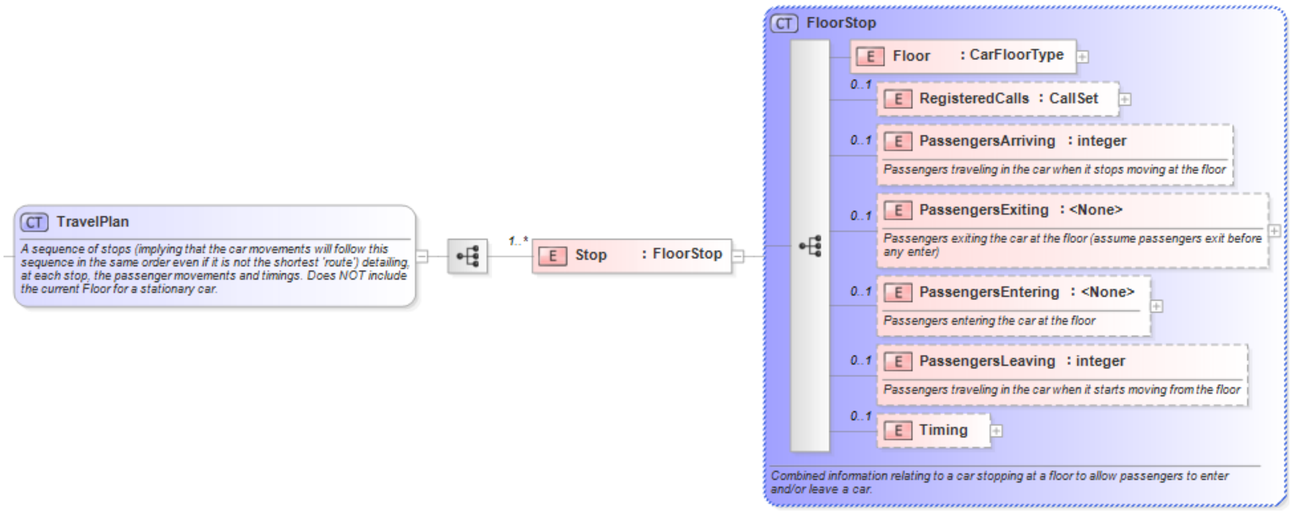
3.2.2.2 **Alternative Path 1)**

If no acknowledgement or a negative acknowledgement is received from an assigned client car then the call assignment will be repeated with that car excluded.

3.2.2.3 **Alternative Path 2)**

The dispatcher may be distributed as a number of collaborating instances, each responsible for a unique set of one or more registered client cars. In this case each dispatcher instance will respond with a "bid" publicising the minimum cost of assignment for adding the call to the travel plan of one of its registered client cars. If having bid, the dispatcher instance is notified by another instance of a lower bid, the assignment will be relinquished in favour of the lower cost bid.

In this mode, the Assignment is only considered to be complete after a positive acknowledgement has been received from the dispatcher responsible for the assigned car.



CT TravelPlan
A sequence of stops (implying that the car movements will follow this sequence in the same order even if it is not the shortest 'route') detailing, at each stop, the passenger movements and timings. Does NOT include the current Floor for a stationary car.
1..*
E Stop : FloorStop
CT FloorStop
E Floor : CarFloorType
0..1
E RegisteredCalls : CallSet
0..1
E PassengersArriving : integer
Passengers traveling in the car when it stops moving at the floor
0..1
E PassengersExiting : <None>
Passengers exiting the car at the floor (assume passengers exit before any enter)
0..1
E PassengersEntering : <None>
Passengers entering the car at the floor
0..1
E PassengersLeaving : integer
Passengers traveling in the car when it starts moving from the floor
0..1
E Timing
Combined information relating to a car stopping at a floor to allow passengers to enter and/or leave a car.

Figure 4 Structure of TravelPlan

4 ARCHITECTURAL REQUIREMENTS

Throughout the analysis process, requirements are identified and these are catalogued[15] to provide a precise definition of the functions, features and capabilities that the system must support once it has been developed. The requirements catalogue therefore forms the basis of the contract between developers and users, as well as the basis for the suite of tests which the system must pass before it can be declared to be finished.

This section addresses the architectural requirements that are derived in part from the use cases and also from study of published reference works.

4.1 Controller Interaction Modes

A lift group controller architecture is required that will enable the dispatcher system use cases to provide the basic interactions between the car and dispatcher allowing:

- the dispatcher to notify the car of its currently assigned LC(s)

  and
- the car to update the dispatcher with its current status (changes) and call registration/cancellation events.

Controller interaction modes to be supported include (though may not be limited to) the following:

Dedicated Group Controller (Simple Hierarchy)[9] **–** where a Group Controller monitors landing call stations, assigns LCs to cars and cancels the call at the landing when informed by the car controller that the assigned car has answered the call. This architecture permits the use of very simple car control, allowing standard hardware to be used across single and multi-lift installations and which might, if desired, be implemented in a non-computerised technology. It should be remembered that this architecture is extremely vulnerable as the service of entire group of lifts is dependent on a single component and so a backup control policy that provides a minimal service must be provided in case of failure.

Master/Slave[9] - only car controllers exist but all are capable of assuming the role of Master, which oversees the assignment of LCs to specific cars. A decision process is defined for designating which car controller is the current master and also selecting a new master in case of failure. This interaction mode provides a resilient solution, where failure of one or more elements leads to a "graceful degradation" of service.

Assignment Bidding (N.B. *This is a suggestion by the author as a logical development of the master/slave mode; no published reference available.*) – every car controller is capable of preparing a bid for responding to an LC and the one offering a minimum "cost" is awarded the assignment, either directly by all other controllers or by a designated group master. This option becomes more attractive as the number of cars in a group and the number of floors served increases, where the number of possible solutions to the assignment task results in a highly demanding computing load that is too great for a single processor to deal with while providing a timely response. The complexity of calculations and consequent computing demand will increase significantly with the introduction, recently announced[12], of vertical transportation systems based on multiple small rope-less cars that share and can transfer between shafts[13]. It is likely that an architecture supporting this interaction mode will become more widely used.

Some algorithms operate by generating assignments (and re-assignments) for all currently registered LCs at the same time, i.e. in a single complete execution of the algorithm, for example when employing so-called "genetic" technology. In this case the assignments must be only for those client cars which are registered locally with the dispatcher instance, thereby excluding the Assignment Bidding architecture since the bidding process is effectively internal to the algorithm (distribution of computational processes, with its associated benefits of scalability and resilience is not excluded but becomes proprietary). However, triggering of the assignment process will be caused, as in all of the architectures by a change of state in some significant data - primarily but not exclusively, the registration of a new LC at a landing call station. The cost of assignment is still the fundamental driver for the decision process and should be reported as an element of each optimised assignment because it is therefore critical to any auditing or analysis of the performance of the algorithm.

The above interaction modes place requirements on the software architecture of the Global Dispatcher Interface. However, it remains a subsequent design decision as to whether the software is to be implemented in a hardware architecture that exactly replicates that of the software, with network communications between distributed electronic components or whether, in reality, it is aggregated within a single device/computer (or indeed any other intermediate level of integration). The advantage of considering the software architecture separately from the hardware is that the designer is then free to choose a distribution of software functionality and a configuration of hardware elements that meets other requirements, for example for resilience and fault-tolerance, communication latency and bandwidth as well as physical location. Again, with the introduction of machine-room-less lifts, the control components and communication distances may be very significant, requiring specific solutions.

4.2 Key Collaborating Objects

Careful examination of the text of the dispatcher system use cases[11] exposes the names, roles and responsibilities of the key structural elements that will come to constitute the eventual solution (both external and internal to the dispatcher). It is important however to understand that these elements are not optional nor are they specific to a particular implementation of the Global Dispatcher - they are necessary for the Global Dispatcher Interface to be truly global and to operate according to the system use cases in any one of the interaction modes.

The aim of this analysis is to understand and define, in abstract terms, the essential elements of software of the Global Dispatcher Service independently of a consideration of constraints, such as the hardware of existing products or limitations of current technologies, in order to provide an optimal breakdown of the overall system into its fundamental components, which can then readily be implemented in many different configurations with the greatest flexibility and a minimum of limitations.

N.B. *A further phase of analysis will lead to the production of a set of Use Case Realisations[6] in the form of interaction diagrams that demonstrate the operation of each of the architectural configurations described above (to be discussed in a subsequent paper).*

The key collaborating elements (objects) are identified in the above dispatcher system use cases, which together comprise the Global Dispatcher Service.

4.2.1 Dispatcher Interface

The dispatcher interface provides an encapsulation of the entire dispatcher service and is its presentation to the external environment. For example, in addition to fulfilling assignment requests in the form of call registration events and accepting status update events from lifts, this interface also provides access to performance monitoring and fault logging functionality.

The dispatcherinterface acts in a similar fashion to a database in that it processes events with generic Create/Read/Update/Delete (CRUD[17]) operations rather than function calls with names that relate specifically to the operation of a dispatcher.

The assignment bidding interaction mode requires that several instances of the dispatcher interface must be able to co-exist and interoperate as parts of the same Global Dispatcher Service though this interoperation may be via a different (internal) interface (i.e. not the Global Dispatcher Interface).

4.2.2 Heuristics + Cost Function

This object is responsible for estimating the cost for the identified car of adding one call to its list of assigned calls, by considering the current state of all cars and other assigned calls. The estimation is initially calculated by applying a Cost Function. This calculation is modified and possibly overridden (e.g. to block the car completely from bidding for the assignment) by the application of rules or Heuristics.

So the capability of this object supports all manner of assignment algorithm mechanisms - from simple relay ladder logic (as in a Programmable Logic Controller) to neural networks and genetic algorithms.

4.2.3 Bid Manager

The Bid Manager is responsible for eliciting bids for all the cars that are local to the dispatcher and, for the assignment bidding interaction mode, capturing bids for all cars that are registered with remote dispatcher instances. Then selecting the minimum cost bid.

Finally, the Bid Manager must notify all the local registered cars plus remote dispatchers of the winning bid, in the form of a CallAssignment event to the winner and CallDeassignment events to the losers.

4.2.4 All Cars State

Responsible for holding the current state of all cars in the group (i.e. not just those that are registered locally). The stored information includes both static and dynamic data for each car. The StaticData of all cars (local and remote) in the group is populated and held in the AllCarsState entity for future reference by the dispatcher while servicing subsequent client requests.

When a state change occurs to a "local" lift, this must be replicated in the data of the AllCarsState object. The change is signalled by a LogEventType request message sent via the local dispatcher interface. This LogEventType message must be replicated to all "remote" instances of the dispatcher interface. In a complimentary manner, the remote instances of the dispatcher interface must signal all change of state events to the "local" instance of the dispatcher interface so that the state of the local instance of AllCarsState can be updated. The LogEventType message is an efficient means of signalling only the information which has changed and implies an event-driven mode of operation. Communication/interaction between asynchronous processes is well suited to the Event Driven Pattern[16] of programming - for example: change of next possible stopping floor, call registration, etc.

4.2.5 Current Landing Calls State

Responsible for holding the current state of all currently registered landing calls. Destination landing calls continue to be held after the assigned lift car arrives at the passenger's origin floor until the car arrives at the passenger's destination floor. The call state includes registration time, cancellation time and allocated car. The CallCycle (see Figure 5) is an efficient means of storing the complete landing call state.

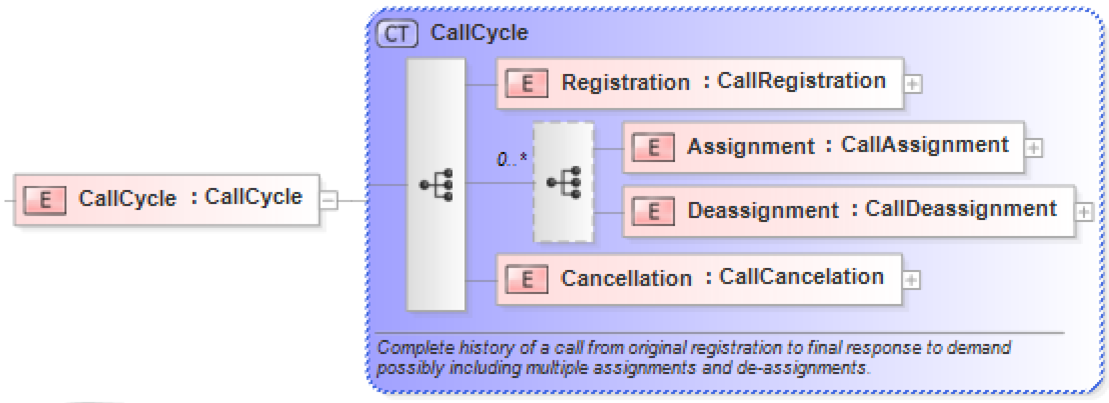

CT CallCycle
E Registration : CallRegistration
0..*
E Assignment : CallAssignment
E Deassignment : CallDeassignment
E Cancellation : CallCancelation
E CallCycle : CallCycle
Complete history of a call from original registration to final response to demand possibly including multiple assignments and de-assignments.

Figure 5 Structure of CallCycle

4.2.6 RegisteredClients - Cars

The list of registered car clients. Local car clients register directly with the dispatcher interface with which they are associated. Remote car clients register indirectly via the dispatcher interface with which they are associated.

4.2.7 Registered Clients - Passenger Call Stations

The list of registered passenger call station clients. Call stations are grouped (as "risers") to enable the dispatcher to factor the time needed for a passenger to arrive at the entrance of the assigned lift. The term "riser" is derived from conventional lift installations but in this context may designate a group of mobile devices that might not even be in the same building as the lifts.

Local call station clients register directly with the dispatcher interface with which they are associated. Remote call station clients register indirectly via the dispatcher interface with which they are associated.

4.2.8 Car Gateway

A further software element is needed although it is not an integral part of the Global Dispatcher Service. This is referred to as the Car Gateway and it is required to ensure that call assignment decisions from the dispatcher can be effected by the lift car to which the assignment has been made.

The objective of the Global Dispatcher Service is to provide, via the Global Dispatcher Interface, assignments to a wide variety of lift equipment from a range of manufacturers and so a standard car interface must be defined. In addition to requiring car status information to be supplied in a standardised form, the dispatcher interface requires the car controller to appear to operate a Directional Collective (also known as Simplex Collective see [9] sec 9.2.2.3) control policy, which allows multiple concurrent LC assignments to a car. If the actual car controller being used does not operate in this way, it will be necessary to create a gateway software component which is specific to the car controller concerned and which then offers the standard, Directional Collective interface to the dispatcher.

Amongst other duties, the gateway will accept multiple assignments and then select one that is the most immediate LC to pass on to the underlying car controller. This aspect of the gateway functionality would be most effectively and efficiently based on the car's Route data element (of type: TravelPlan)[10], though this is not in itself a requirement.

It is noteworthy that the complex type :FloorStop(), which is part of the TravelPlan type, optionally includes shaft occupancy as well as floor level information and is therefore forwards-compatible with (ropeless) vertical transportation systems based on multiple cars that share and can transfer between shafts. Whether or not a car's TravelPlan is communicated, a change of the shaft currently used by each car is, in any case, communicated via the CarShaftEvent type.

The gateway could also assemble statistical information on car static data, for example:

- door timings (DoorProfile)
- inter-floor trip times (SpeedProfile)
- energy consumption (ReferenceEnergyProfile)

for use both by the dispatcher cost function and also for more general logging and maintenance reporting.

5 REPORTING FUNCTIONS

Whilst it is common practice for data-logging functionality, which supports remote monitoring of lift status, door operation, fault conditions, etc to be included in group control equipment and even though all such information is available to the dispatcher, this capability is not specific to the dispatcher and so will not be described here. However, there is a definite requirement for the dispatcher to report:

- bid preparation (assignment cost calculation and application of rules)
- bid selection
- error and exception conditions (e.g. preventing assignment, client registration failures, etc)

This capability is essential since the dispatcher is expected to be globally compatible with all configurations of groups of lifts and without it, the dispatcher would remain a "black box" making inexplicable and apparently arbitrary decisions.

It is the responsibility of the dispatcher interface to pass the reports to the external environment, though a separate communication channel from that which handles assignment requests, etc may be employed actually to pass the data.

6 SECURITY

The dispatcher interface will normally be connected to a network, so all the elements of the Global Dispatcher Interface must be protected from infiltration and attack by malevolent agents. Precautions, such as firewalls, authentication (built into the client registration process) and possibly encryption of communication channels should be implemented using standard components which can be obtained from third parties specialising in these complex technologies and who will provide maintenance updates to protect against newly discovered threats and weaknesses. It is the intention of this paper to note the requirement rather than to present the details of such measures.

7 CONFIGURATION AND UPDATE MANAGEMENT

There is as yet no standardisation, either of the names or the nature of the parameters that are required for configuring a dispatcher to match the building in which it will operate or the lifts it will control. Furthermore, different parameters will be required, depending on the design of the algorithm and the implementation of a particular manufacturer. However, there remains a requirement for the dispatcher interface to provide a mechanism whereby parameters may be set during installation and possibly also modified during the service lifetime of the lifts, for example if the building occupancy or utilisation changes, or simply an unusual event occurs - a conference is scheduled, an emergency, etc.

It is to be expected that new versions of the dispatcher software will be released, either to fix errors or deficiencies in the current version or simply to introduce new features and improvements. While this update capability might be undertaken by a technician on site substituting physical components with replacements that have been pre-loaded with the updates, it would be quicker and easier to control the process if such updates could be performed via secure remote access to an update mechanism. This is increasingly the method of choice for applications running on general purpose computers and mobile devices. While not an essential requirement of the Global Dispatcher Interface, a remote software installation and update facility remains a highly desirable feature.

8 RESOURCE MANAGEMENT AND REGISTRIES

Lift systems exist to provide a service to the users of buildings (passengers, managers, maintenance technicians and owners) and increasingly will need to integrate with other sophisticated services of smart buildings and cities within which they are located. In order to achieve such integration it will be necessary to publish the capabilities and to control access to dispatcher services as resources in a standardised form. For example, information such as the physical location of the building and the lifts within it and the configuration of dispatcher services needs to conform to agreed standards. Third party registries (e.g. Waher[18]), which provide categorisation and publication of resources already exist and are subject to ongoing development, with inbuilt security. These repositories should be used, in preference to proprietary solutions, so as to avoid inconsistencies and resulting fragmentation of the potential for integration of the Global Dispatcher Service.

9 CONCLUSIONS

While taller buildings and novel configurations of lifts place greater computational demands on the call assignment mechanism (dispatcher), lift manufacturers develop increasingly sophisticated group control algorithms in order to optimise the available resources. The task is further complicated with the introduction of new passenger signalling devices and modes of interaction, including mobile devices. Currently, there is no accurate means by which the algorithms of different manufacturers may be compared nor is there a reliable method for integrating the lifts of one manufacturer with the group control policy of another. In this paper a Global Dispatcher Interface has been described that brings standardisation to this problem domain and thereby allows architects and building owners to design, manage and maintain modern buildings with greater confidence.

The first phase of the process of defining the Global Dispatcher Interface, that of capturing and analysing the requirements has been described, using established software development methods, through the documentation of Passenger Use Cases and Dispatcher System Use Cases. The architecture of modern lift control systems has also been discussed yielding further requirements and leading to the identification of key software elements of the Global Dispatcher Service with descriptions of their roles and responsibilities. An overview of the analysis work has been presented in this paper, while references are given to the complete details in the form of a structured UML model and supporting documents generated from that model. The referenced documentation is published on the worldwide web.

A further paper is planned to describe the design and implementation of a prototype Global Dispatcher Interface, which is currently the subject of ongoing research and development.

The material presented in this paper is, by its nature, a proposal and has yet to be implemented commercially. The author welcomes comments and questions, via the Editor, regarding possible improvements, errors and omissions.

REFERENCES

[1] Peters, P. (2016). *Global Dispatcher Interface - Proceedings 6th Symposium on Lift & Escalator Technologies* (pp. 18-1 18-9). Northampton: CIBSE.

[2] G. C. 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.

[3] thyssenkrupp Elevators Corporate Website . *AGILE.*, From: thyssenkrupp-elevator.com: http://www.thyssenkrupp-elevator.com/agile/
[Accessed February 16, 2018]

[4] KONE Corporate Website, "*KONE DESTINATION SOLUTIONS*", 19/02/2016, https://www.kone.co.uk/new-buildings/kone-people-flow-intelligence/destination-solutions/
[Accessed February14, 2018].

[5] Sparx Systems Corporate Website. *Sparx Enterprise Architect*. From: http://www.sparxsystems.com. [Accessed Jan, 2018].

[6] Bitner, K and Spence, I. (2003). Use Case Modelling, Pearson Education Inc., Boston USA, ISBN0-201-70913-9.

[7] Eriksson, H.-E., & Penker, M. (1998). *UML Toolkit - Domain Analysis*, (pp. 247 & 324 - 325). John Wiley & Sons, Inc.

[8] Beebe, J. (2018). *Global Dispatcher Service - UML Model*. Retrieved from: dispatcher.std4lift.info/html/index.htm
[Accessed February 17, 2018]

[9] CIBSE. (2015). *CIBSE Guide D. 2015 Transportation Systems in Buildings*. The Chartered Institution of Building Services Engineers.

[10] Beebe, J. "*Standard Elevator Information Schema*", http://www.std4lift.info/ [Accessed February 17, 2016].

[11] Beebe, J. (2018). *Global Dispatcher - Use Cases*., From: Jonathan Beebe - Global Dispatcher: http://dispatcher.std4lift.info/GlobalDispatcher-UseCases.pdf [Accessed February 17, 2018]

[12] thyssenkrupp Elevators Corporate Website. *MULTI*., From: thyssenkrupp-elevator.com: http://www.thyssenkrupp-elevator.com/en/products-and-service/multi/# [Accessed January 23, 2018]

[13] Gerstenmeyer, S., & Peters, R. (2016). *Multicar Dispatching*. 6th Symposium on Lift & Escalator Technologies - Proceedings Vol 6 (pp. 8-1 - 8-12). Northampton: The CIBSE Lift Group.

[14] Emre Oner Tartan, Cebrail Ciftlikli.(2016) *A Genetic Algorithm Based Elevator Dispatching Method For Waiting Time Optimization*; IFAC-PapersOnLine; Volume 49, Issue 3(pp. 424-429)

[15] Beebe, J. (2018). *Global Dispatcher - Requirements Catalogue*., From: Jonathan Beebe - Global Dispatcher: http://dispatcher.std4lift.info/GlobalDispatcher-Requirements.pdf [Accessed February 17, 2018]

[16] Richards, M. (2015*). Software Architecture Patterns. Chap. 2. Event-Driven Architecture*, O'Reilly Media, Inc.

[17] Wikipedia, *CRUD definition*, https://en.wikipedia.org/wiki/Create,_read,_update_and_delete, [Accessed May 26, 2018]

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