Architecture-Based Development of Road Traffic Management Systems

Michel dos Santos Soares, Jos Vrancken, Yubin Wang

Abstract—Road traffic problems, such as congestion and accidents, are high in many countries, leading to economic losses, environmental damage due to increased pollution, waste of time in congestion and lives lost. In order to reduce these problems, Road Traffic Management Systems (RTMS) are applied, for example, in activities such as controlling, monitoring, and visualizing traffic in motorways and urban roads. The development of RTMS is challenging. RTMS are large socio-technical systems, which means that organizational and regulatory policies, rules, processes and constraints have to be taken into consideration. This framework needs to be represented in what is called the domain architecture. However, defining the domain architecture relates more to the RTMS domain than to the technical solution. The software architecture is fundamental for the organization of the system, comprising the sub-systems, important software components and the relationship between them and the environment, and to document all important technical decisions. In this article, both the domain and the software architecture are proposed for developing RTMS. The approach is used in practice as shown by the HARS case study.

I. INTRODUCTION

Road traffic is currently the most important and flexible means of transportation in most countries. Considering the EU-27 countries, road freight transportation represents about 73% [1] of the inland freight transportation market. The largest share of intra-EU passenger transportation, around 85%, is carried by road [2]. However, the current status of road traffic in many countries is extremely unpleasant. Road traffic is dangerous, expensive and has a high pollution rate. Road congestion is costing the EU-27 about 1% of its GDP. Accidents injure or kill thousands of people every year (around 43 thousand fatalities in 2007 in EU-27) [1]. Traffic congestion in many big cities has gone almost out of control. Environmental damage is another issue. CO₂ emissions from transportation in general and road transportation in particular have been rising faster than emissions from all other major sectors of the economy.

Basically two approaches can be applied in order to solve or at least minimize these transportation problems. The most straightforward solution is to build more infrastructure, such as bridges, roads and viaducts, in order to increase capacity. This solution is no doubt useful, especially for decreasing congestion, but it is not sufficient [3], [4]. Constructing new road infrastructure is limited due to environmental, social and financial constraints [5].

With difficulties of building more infrastructure and the aforementioned transportation problems, an approach in which already existing road capacity is better used is welcome. This second approach to traffic related problems is to control traffic by deploying Road Traffic Management Systems (RTMS), which contribute to efficiency as well as safety and environmental improvements. This is done by applying intelligence to the current infrastructure, switching from static to more dynamic road traffic control [6].

There are many issues in designing and deploying RTMS. As a socio-technical system, the organizational and regulatory policies, rules, processes and constraints have to be taken into consideration. These decisions have to be documented in what is called a domain architecture in this article. In addition to the policy decisions, the technical side of these systems is also challenging. Specific constraints such as interoperability with existing systems and the close relationship with the environment make the development effort extremely difficult. Besides, due to the high level of investments needed, these systems have to be flexible to be changed whenever new policies are to be implemented.

The proposed solution in literature to build these large software systems is to base the design and development in an architecture [7], [8]. Future systems’ maintenance and evolution are facilitated when the architecture is clear for all stakeholders [9]. Basically, architecture refers to the organization of the system, such as its components, sub-systems, interfaces, and how these elements collaborate and are composed to form the system [10]. Only relevant decisions are important at this level, i.e., those that have a high impact on cost, reliability, maintainability, performance and resilience of the future system.

In the research that led to this article a multi-disciplinary approach, by combining Traffic Engineering and Software Engineering, was used. Both the Software Engineering field and the Traffic Engineering field have developed architectures in past years, and the benefits are clear. However, it seems that the two communities are disjoint, solving their problems from their specific points-of-view [6]. In this article, both software and domain architectures are proposed and their relationships are explored. This is important from the practical point of
view, as more specific domain architectures provide better separation of concerns and a higher level of abstraction, which are considered good principles of Software Engineering [11] and scientific knowledge in general [12]. These architectures are the basis for the practical development of RTMS, as shown in the case study.

II. ROAD TRAFFIC MANAGEMENT SYSTEMS

RTMS are complex, large socio-technical systems that influence traffic by using a variety of actuators, such as traffic signals and Variable Message Signs (VMS), based on acquired data using various types of sensors, such as video cameras and inductive loops. These systems’ environment is inherently distributed, with vehicles and road-side equipment (sensors and actuators) geographically spread over the network. The purpose of RTMS is to monitor, predict, visualize and control traffic.

RTMS are systems that control a crucial infrastructure for modern society. Infrastructure systems, such as transportation systems are socio-technical systems. These systems include a variety of stakeholders such as users (drivers, pedestrians), network operators, policy makers and regulators. All these stakeholders influence the design of RTMS by explicitly expressing requirements. For instance, society may have as a goal that throughput in a certain region of the road network must be improved.

A. Domain Requirements

The characteristics and requirements of RTMS are important for the decisions on the fundamentals of the software architecture used for development. Many non-functional requirements are important for RTMS. The most important ones for the RTMS architectures to be described are presented as follows.

Scalability is of great significance because there may occur changes in the road network, which may include new sensors and actuators to be accommodated into the system. Thus, new software objects may be inserted into the software architecture each time the network grows. This is important to allow the system to evolve over time.

Performance is also an issue, as there are many components to be controlled. One of the drawbacks of distributed systems is related to performance. The performance problem is compounded by the fact that communication, which is essential in a distributed system, is typically quite slow. Responsiveness respecting real-time constraints must be observed as data should be updated regularly. An example of available data are road traffic measurements information that are geographically distributed within the network. Availability of data is of considerable influence to system reliability and overall performance. This means that applications and technical infrastructure must feature high reliability, as to not jeopardize human life by system malfunctioning.

Flexibility is essential due to the possible changes and adaptations of component types, interfaces, and functionality, as the system evolves over time. Finally, the presence of legacy systems along the roads, and the circumstance that it would be too costly to ignore the existing systems, implies that interoperability with legacy systems is an important requirement.

B. The Importance of Architecture for RTMS

Explicit decisions must be documented in order to be communicated to other stakeholders. These important decisions are documented in what is called an architecture. An architecture is a high-level representation of a system, in which important decisions are documented to be presented and shared to all stakeholders. The architecture shows the system in terms of its sub-systems, components, interfaces and their relationships. The architecture framework only cares about the most important parts, the ones that have higher possibilities of being reused across a series of products [13].

As RTMS are fundamentally dependent on both Traffic Engineering and Software Engineering domains, the decisions of each of these domains are documented in a specific architecture. As a matter of fact, two architectures are proposed in this article. The first is the domain architecture (Section III), which deals with the organization, information, society-goals and transportation related themes. The second is the software architecture for the implementation of a family of systems (Section IV), which deals with software components and their relationships, data communication, and the overall software organization. This approach to architecture, in which two architectures are proposed, is used to cover the gap found in practice between traffic engineers and software engineers.

III. DOMAIN ARCHITECTURE

A. ATC

The ATC (Architecture for Traffic Control) [14] can be used to develop and implement a set of coherent traffic management measures and the necessary technical and information infrastructure. The goal of ATC is to be a domain architecture to build RTC systems. Therefore, the ATC should be combined with a software architecture as a basis to develop the software systems.

The ATC framework, showing the main parts of the architecture, is depicted in Fig. 1. The three horizontal layers are most relevant here.

![Fig. 1. Architecture Framework for Traffic Control.](image)

The Applications and Technical Infrastructure layers form the technical parts of the architecture. The Applications layer describes the applications to implement the RTC features mentioned in the RTC layer, such as the control systems for each of the local measures, and the operator’s user interface and support tools in the Traffic Management Centers (TMC).
This layer includes roadside sensors and actuators used in control systems. Applications in this layer can be developed based on a common software architecture, which is discussed in Section IV.

The Technical Infrastructure is about the data communications network and the processing platforms along the roads and in the TMC. This layer also comprises a vitally important middleware layer that facilitates high-level communication between application components.

The RTC layer describes the ATC approach to RTC, in typical traffic engineering terms. The RTC layer can be expanded into the layered model depicted in Fig. 2. The purpose of this model is to show how society goals in traffic management, such as improved travel time reliability and less delay, can be translated, in five steps, into the signals shown to drivers.

Control strategies are general principles and strategies for RTC. Examples are the prioritization of certain types of roads, such as belt roads around cities, and the restriction of slow traffic to the right-most lane (in countries with driving on the right).

Control tactics are a set of general rules and principles to improve traffic conditions. Examples are buffering of less important traffic streams and rerouting of traffic streams to relieve more important roads. Control tactics focus on a specific region of the network.

The Control Scenarios layer is the layer at which operators in TMCs work. In this layer the control tactics for a specific part of the network is implemented. Control Scenarios are integrated programs of control measures that cover the area under control of the TMC. The focus of control scenarios are a specific region on specific moments in time. Scenarios are developed off-line and their execution is triggered by and responds to certain recurring patterns in traffic, such as the morning rush hour. The Control Scenario that matches best with the current traffic state is determined automatically online.

Control Measures are the usual local measures, such as traffic signals, speed instructions and warning signals. All coordination of local measures takes place at the Control Scenarios layer. In ATC, local measures are not supposed to communicate with each other. Instead, they receive instructions top-down from the TMCs.

Finally, the Signals layer deals with the various visual signals present on the road to be shown to drivers (e.g., VMSs, traffic signals).

B. DTCA

ATC has a number of limitations, as discussed in [6], primarily caused by its top-down, single-agent nature, which has serious limitations in handling complex situations and limited scalability. In addition, the scenarios are optimized to a specific traffic pattern, which may differ from real cases. To overcome these problems, DTCA (Distributed Traffic Control Architecture), an extension to the ATC architecture, is proposed.

In the ATC layer of DTCA, bottom-up control is added as an extra measure in the Control Measures sub-layer. This measure entails that network elements, such as junctions and road segments between junctions (links), but also routes and origin-destination pairs, become active agents, measuring their own traffic state and communicating with other network elements, mostly the adjacent ones (in the case of junctions and road segments). The communication consists of requests for information about traffic states in nearby elements and of requests to take certain measures, such as reducing the inflow to the requesting element. The bottom-up control is intended as an addition to and fine-tuning of the top-down control, certainly not as its replacement.

On itself, it is not obvious why the addition of multi-agent control could be useful. A multi-agent system is usually not very predictable in its behavior, but there are reasons why, in this case, this addition looks promising. Due to the many local measures spread in the road network, control of traffic is already strongly distributed. The essential addition in DTCA is that it makes local measures communicate with each other. Moreover, such local, direct influencing of network elements among themselves gives a much quicker, real-time response to local changes in traffic and a much shorter control loop than via the TMC. In addition, multi-agent control, by its local nature, can easily scale to any size of network.

In the Applications layer, components are added that represent the communicating network elements, such as road links, junctions and routes. This is an immediate consequence of the choice to add multi-agent control to the RTC layer.

In the Technical Infrastructure layer of ATC, an explicit choice is made for asynchronous Publish-Subscribe middleware (see the implementation view). This is the most appropriate kind of middleware, given the requirements of performance and scalability. In addition, it is known for efficient use of network bandwidth. This kind of middleware can accommodate both the top-down and the bottom-up control mechanisms, in addition to the communication needed for data collection.
IV. SOFTWARE ARCHITECTURE

A proven concept to build software systems and to avoid problems and failure causes, such as poor communication among stakeholders and sloppy and immature development practices, is to have a well-defined software architecture [7]. According to [15], having an architecture allows the development of systems that are better and more resilient to change when compared to systems developed without a clear architectural definition. In addition, adaptations can be easily implemented when the software architecture is known and well-documented. Software architecture affects the performance, robustness, and maintainability of a system [16]. The architecture style and structure may both depend on and influence these requirements, which are fundamental for large software systems.

A common approach in software architecture design is to apply separation of concerns by proposing multiple architecture views. Each view addresses a specific set of concerns of different stakeholders. A set of views is used to describe the different aspects of the architecture. Examples of multiple views architectures are the Siemens Four View model [17] and the 4+1 View model [18]. Two of the most representative views, the logical and the implementation view, are discussed below.

A. Logical View

The architecture shows which elements cooperate with each other in a high level manner, without concern about how this interaction is done. These network elements are:

- **Origin-Destination Managers (ODMGR)** represent the relation between an origin and a destination and comprise one or more routes.
- **Route managers** control the set of routes from one origin to one destination.
- **Links** come in two types, Main links and Accessor links. The Main link is the link from the merge point to the choice point and the Accessor link is the link from the choice point to the merge point.
- **Junctions** comprise the outgoing Main link and the incoming Accessor links of a crossing or motorway junction. A junction is a location where traffic can change its routes, directions and sometimes even the mode of travel. Normally a junction has many sensors, such as inductive loops, and actuators such as traffic signals.
- **Control Schemes** are coherent set of measures triggered by recurring patterns in the traffic state, such as the morning rush hours or the weekend exodus.

The representation of the logical view of the architecture is shown in Fig. 3 using a SysML Block diagram [19]. A SysML Block is a modular unit of system description. It can represent both structural and behavioral features, such as properties and operations, to represent the state of the system and behavior that the system may exhibit. SysML Blocks can model either the logical or physical decomposition of a system, and the specification of software, hardware, or human elements.

Each road network element represented by a SysML Block has a direct software object representation. The distributed components have to communicate with each other, as they work in cooperation. For instance, links have to communicate with other links in order to achieve a traffic state. They continuously measure the traffic state and communicate about it to other links in real-time.

```
<<block>>
Control Scheme
<table>
<thead>
<tr>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morning rush hour</td>
</tr>
<tr>
<td>Afternoon rush hour</td>
</tr>
<tr>
<td>Friday</td>
</tr>
<tr>
<td>Sunday</td>
</tr>
<tr>
<td>Quiet hour</td>
</tr>
</tbody>
</table>

<<block>>
ODMGR
<table>
<thead>
<tr>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Origin: String</td>
</tr>
<tr>
<td>Destination: String</td>
</tr>
</tbody>
</table>

<<block>>
Route Manager
<table>
<thead>
<tr>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..1 participates</td>
</tr>
</tbody>
</table>

<<block>>
Junction
<table>
<thead>
<tr>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..1 comprises</td>
</tr>
</tbody>
</table>

<<block>>
Link
<table>
<thead>
<tr>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: String</td>
</tr>
<tr>
<td>Main Link</td>
</tr>
<tr>
<td>Accessor Link</td>
</tr>
<tr>
<td>Origin Set</td>
</tr>
<tr>
<td>Destination Set</td>
</tr>
</tbody>
</table>

Fig. 3. Logical view and the relationship between control elements
```

B. Implementation View

The layered architecture model of implementation is depicted in Fig. 4 using a UML/SysML Package diagram. Layered models are ideally suited to express important parts of an architecture, describing a way to organize systems by separating similar and related components into layers. Each layer accommodates related semantic models, providing strong separation of concerns.

There are many advantages in using a layered architecture for developing systems [20]. Layers show diverse levels of abstraction, from specific, top levels, towards general, lower levels, hiding details whenever convenient. Each layer can be detailed into sub-layers, and sub-layers can be combined into layers, providing flexibility for development. Finally, regular changes may be easily implemented when they are related to a specific layer instead of to a whole system.

The Presentation layer is used mainly to present the applications results, and can be made for diverse devices/platforms. Developers may use a list of existing Generic Components and combine them as much as necessary to help in building Functional Components, which implement user requirements. The models of the Functional Components layer are designed
following an object-oriented approach with UML as modeling language.

C. Publish-subscribe middleware layer

The environment characteristics (distributed elements in a network) imply a need for a distributed system architecture, and the real-time constraints, which call for high performance, are important non-functional system requirements. Not only having correct data is important, but also the availability of data. Respecting real-time constraints is difficult as sensors and actuators are physically distributed, which increase the possibilities of missing data or receiving data outside of the bounded time. This indicates the importance of using an asynchronous middleware to coordinate communication and data exchange between software objects.

Distributed software objects in a network must communicate with each other by asking and providing services. There are many different means of implementing process communication [21]. The simplest is by using synchronous communications, based on point-to-point and blocking communication between processes. This type of communication presents many problems [22]. As processes have to wait for a response in order to continue further operation, performance is reduced. In addition, there is lack of transparency and flexibility, as it is necessary to know which process to call for a given service.

The chosen middleware here is of the Publish-Subscribe type [23], [24]. The basic functioning of a generic publish-subscribe middleware is given in Fig. 5. Components may subscribe to information, unsubscribe, publish and consume information. The Event Handler receives all these events through an interface, and can also notify components when the information is of relevance to them.

The publish-subscribe communication mechanism naturally supports an asynchronous (non-blocking), many-to-many communication between components in a network. The notion of clients and servers exchanging messages is substituted by an event based communication between components that may act as publishers of information and/or subscribers for information. Publishers (acting as servers), publish information through an event, which will be delivered to all interested subscribers that expressed their interest in a certain type of information by subscribing to it. This gives improved system performance. Publishers are not aware of which particular subscriber will receive the published information. In a similar manner, subscribers are indifferent to which specific publisher produces the information. They even do not need to run on the same machine. This means that publishers and subscribers are space-decoupled [25]. Another important characteristic is that publishers and subscribers are fully decoupled in time [25]: publishers and subscribers do not need to be connected at the same time. All these manners of decoupling (synchronization, space and time decoupling) increase scalability and reduce the necessity of coordination, which makes publish-subscribe middlewares most suited to distributed environments, fulfilling the specified requirements of RTMS [26].

V. CASE STUDY

HARS (Het Alkmaar Regelsysteem, the Alkmaar Control System in Dutch) is an implementation of DTCA. The purpose of the system is to prove the concepts behind DTCA, especially the assumption that in RTC, a dual control strategy is stronger than just top-down control. HARS can be considered successful if it has a noticeable, positive effect on traffic in Alkmaar, if the multi-agent control plays a substantial role in this and if the system turns out to meet the non-functional requirements mentioned in Section II.

HARS takes the existing local RTC measures and their systems, such as traffic signals at junctions and VMSs as starting point, and adds a software overlay layer to these local measures. In the HARS system, the functions of the ODMGR are enhanced and extended to enable more effective rerouting using the Route Manager. VMSs are used for rerouting and informing drivers on the current traffic state of routes downstream of the VMS. In case of congestion, drivers will thus know the extent of the congestion they can expect.

Top-down control is performed by Control Schemes, which are developed off-line and correspond to recurring patterns in the traffic state, such as the Morning Rush Hours or the weekend exodus (Friday). This is the top-down approach, in
which the TMC (the top) is the only entity allowed to take decisions.

The bottom-up control is implemented by means of communicating agents (links and junctions). The control is performed by so-called capacity services, (Reduce capacity, Increase capacity) which actually are operations configured in links. By using capacity service calls, links (Main links or Accessor links) are able to offer and ask for capacity from downstream and upstream links respectively. Services are implemented in various ways, according to the available local measures in the network element: a link may implement it by reducing the maximum speed or closing one or more of its lanes; a junction may implement it by reducing/extending green times of traffic signals.

The asynchronous publish-subscribe communication is used as the middleware layer. The publish-subscribe style of communication reduces the dependency among the components and makes the information available where and when needed. This should also reduce communication overhead between agents, since it is not necessary for the components to continuously request data and wait for response, which blocks the components. As a matter of fact, important quality factors for the system such as performance, scalability and flexibility are improved using this communication style.

VI. CONCLUSION

In this article, two architecture levels are proposed for the development of RTMS. The first is the domain architecture, expressed in traffic engineering terms. The DTCA consists of the ATC architecture with a number of additions in the horizontal layers for multi-agent control.

For the software architecture, both the logical and the implementation views of the architecture were presented. The software architecture is the basis for the development of RTMS, considering the domain architecture and requirements as initial constraints. The logical view shows the main elements of the system and their relationship. A layered approach is proposed for the implementation view. Publish-subscribe middleware and components framework were recognized as useful to facilitate development, by increasing abstraction levels, and coping with real-time constraints and modularity. The choice for publish-subscribe middleware contributes to flexibility, scalability, performance and the interoperability with legacy systems.

The proposed dual architecture for RTMS covers both the domain and technical aspects for systems design. In practice, both architectures are necessary as RTMS are socio-technical systems with many stakeholders. Future research concerns the combination and evaluation of architectures for other domains than road traffic.

REFERENCES