Navigation devices, including apps for smartphones, have thus far been proprietary and closed. The Open Mobile Alliance Dynamic Navigation Enabler lets developers create novel navigation services characterized by openness and interoperability across different information providers.

Navigation devices (NDs) are common driving-assistance tools that increasingly integrate local information, such as maps and user position, with remote data, such as real-time traffic information. However, an ND from one manufacturer is rarely capable of accessing data from a different service provider because current systems are either proprietary or totally closed.

Google Navigation and Apple Maps are well-known examples of proprietary solutions. They define a set of APIs that are exploitable by other parties, but they don’t guarantee interoperability with other solutions. More importantly, those APIs can have technical or legal limitations; for instance, Google doesn’t allow the use of its data in applications other than Google Navigation (see https://developers.google.com/maps/documentation/directions and https://developers.google.com/maps/terms). In addition, Google APIs are subject to change at any time. Other solutions, such as TomTom or Garmin, either rely on the old radio-based broadcast communication channel or exploit a closed architecture (services and protocols), hence preventing users from switching to another provider. Moreover, previous solutions, being monolithic, don’t allow the use of alternative components (for instance, better real-time traffic information sources) and offer limited opportunities for customization.

To overcome these limitations, the Open Mobile Alliance (OMA) recently standardized an open protocol for dynamic road navigation services. The OMA Dynamic Navigation Enabler (DynNav) introduces a bidirectional communication channel and a modular approach, while reusing existing standards for some specific features. By
filtering data based on routes, geographic areas, and time, the bidirectional channel allows users to ask for or receive only the information they are interested in.² The standardized modular approach allows multiple services to be created on top of the defined components and enables the same service to be provided by multiple operators (avoiding user lock-in) and implemented in multiple flavors. In this way, new actors can participate in provisioning navigation services by creating valuable or specialized components, such as route computation algorithms optimized for vertical markets (goods delivery, for example), precise traffic information, and more. Modularity and openness let users not only exploit multiple providers in their solution (for instance, one for route computation and another for traffic information), but also seamlessly change providers.

Among the possible players, telecom mobile operators might see huge benefits from an open standard for navigation services. First, they are constantly looking for ways to provide new services to their customers, and dynamic navigation is an appealing option for many users. Second, mobile operators can obtain traffic information by exploiting their own assets—in particular by (anonymously) tracking the position of their mobile users, hence providing real-time traffic information and rerouting capabilities based on real-time data.

In this context, and especially in smart cities in which multiple sources of real-time data are available, DynNav-enabled solutions could not only provide navigation services but could also become aggregators of multiple information sources, helping users to find localized information such as restaurants or attractions that match their preferences and interests.

**The DynNav Solution**

OMA—the leading industry forum for developing market-driven, interoperable mobile service enablers—completed the standardization process for DynNav in September 2012. This represents an additional step toward the full support of navigation applications by the OMA standardization framework.¹

**Architecture**

Figure 1 presents a possible navigation service architecture based on DynNav. The server is the middle block, whereas the left block represents a typical ND (for example, a dedicated device or smartphone). A DynNav client can also be an application residing on a server (right block), such as a Web-based journey planner.

---

**Figure 1.** DynNav enabler architectural diagram. DynNav specifies the interaction between a DynNav client and a DynNav server (solid lines), whereas the interactions with other components (dashed lines) rely on protocols that are outside the scope of the DynNav specification.
DynNav specifies the interaction between a DynNav client and a DynNav server (the solid lines in Figure 1), whereas the interactions with other components (the dashed lines) rely on protocols that are outside the scope of the DynNav specification. Additional elements not depicted in the figure can be introduced to optimize bandwidth consumption, which is critical in mobile communications. For example, proxy servers can cache frequent responses.

**Protocol**

The DynNav protocol is based on the Representational State Transfer (REST) paradigm and follows the OMA RESTful Network NetAPI guidelines.

A server that implements DynNav provides the following main functions:

- analysis of client-defined trip parameters and proposal of a set of routes based on real-time and forecast traffic data;
- real-time and forecast traffic information related to a set of routes (or geographical areas) previously proposed by the ND or DynNav server itself;
- new route proposals when the quality indicators associated with the current route become unacceptable or the user deviates from the current route;
- complementary information, such as points of interest (POI), related to either a route or an area; and
- subscriptions to specific routes that notify clients with available information when they are driving along those routes.

Note that a client might not be allowed to access all the functions listed: the DynNav service provider can allow or deny access to some functions depending on the user contract or other criteria.

**Data Structures**

Whenever possible, the DynNav specification reuses data structures already defined in previous standards, with the objective of speeding up the implementation and facilitating the integration of additional components and services into the framework. In particular, it exploits the Transport Protocol Experts Group (TPEG) standards for road traffic messages and location entity description. Moreover, it exploits IETF RFC 4776 for civic addresses and the W3C “Points of Interest Core” draft for POI information. The drawback of this approach is the impossibility of obtaining a fully optimized protocol, given that the data formats and protocol messages can be more verbose than necessary. Because no public standards are currently available for encoding route information, a novel encoding schema has been proposed: A route is represented by a sequence of segments (that is, road sections without intersections). Each one includes the segment origin, destination, name, measured or forecast performance parameters (traveling time in regular conditions, delays, expected speed, and so on), and segment shape (a sequence of points for its graphical representation on the map).

Figure 2 depicts a portion of the DynNav resource tree (the complete structure is shown in prior work) and a sample of request and response messages. The curly brackets identify parametric parts of the URIs. The trips resource contains a collection of trip resources, each one identified by its unique tripId. A trip resource includes information about the journey (such as source and destination) along with nested resources representing alternative routes to the destination. These resources, identified by their unique routeId, include routing information and might also include references to sets of events grouped by category. Users can obtain the details by sending to the server a request that contains the list of events to be retrieved. The selection of interesting events can be done either manually or automatically by the client application. This choice can reduce the amount of bytes transmitted over the network because only interesting events are retrieved, even if it introduces a small delay due to the additional request. If the same event is shared by multiple resources, such as different routes or areas, that event is transmitted only once. Events can be stored in proxy servers, if present, reducing the number of requests to the main server.

**Use Case Example**

A typical application of the DynNav service refers to an ND without path computation capabilities, which can originate the message flow depicted in Figure 3 (a detailed description is available in prior work). In this scenario, the user first specifies the parameters of a trip by sending message 1,
which triggers the creation of a new trip resource in the server, including all the details about the trip as provided by the user. Trip creation also forces the server to calculate a set of routes that satisfy the constraints.

The client then retrieves and selects one of the possible routes the server proposed using messages 2 and 3. Each route includes links to events that have occurred (whose descriptions are stored on the server) and a field that specifies each event’s category (traffic, weather, and so on). A client application might decide not to retrieve all those events from the server (message 4) based on user preferences or other criteria (such as priorities). Message 5 creates a subscription to the notification service to receive real-time traffic information updates and proposals of alternative routes. The DynNav server will use user-provided location data to update the user’s status and send notifications about traffic and other events specified in the user’s subscription. If the estimated traveling time becomes too high—for instance, because of road congestion—a new route can be suggested. Message 6 represents a notification from the server that contains references to related events, which can be retrieved as in message 4.

**Deployment Scenarios**

The flexibility of the DynNav standard enables the creation of rich navigation services without locking users into a specific service provider or using a fixed set of functions. This is possible because the same service (such as route computation) can be provided by different entities, and the basic set of DynNav messages can be combined in (almost) arbitrary ways to create complex applications. For example, DynNav supports both smart NDs, which calculate routes and rely on central services only for real-time information (traffic and so on), and lightweight NDs, which delegate everything but the user interaction (for example, route display) to central services.

The ability to exploit services provided by multiple entities also enables the creation of applications targeting very specific vertical markets. For example, a shipping goods company can opt for
a lightweight ND for its employees, relying on an in-house service for route computation that can minimize delivery costs, and on a telecom operator to obtain real-time traffic information. Along the same lines, a company can set up only a specialized service that suggests the best points of interest based on the user's preferences or other parameters, relying on the fact that other providers can offer the other information required for building a whole navigation service.

Validation
We implemented the DynNav standard in a prototype that includes both the client and server portions of the specification. We used those components to carry out an extended set of tests aimed at validating the characteristics of the standard in a real-world environment and giving insight into the performance this solution can achieve. Particularly, we were interested in checking whether DynNav is suitable for delivering navigation services to real mobile devices. Although our prototype was not engineered to compete with commercially available solutions, we compared DynNav with the widely used Google Directions Web Services (https://developers.google.com/maps/documentation/directions), which offers similar primitives and exploits a similar data structure.

Prototype
The DynNav server was developed in Java and installed on a GlassFish application server; the ND hosting our application was a low-end Android 2.2 smartphone clocked at 768 MHz with 512 Mbytes of memory.

We set up some specific server functions by reusing existing external components: route calculation is delegated to an external Web service (http://openrouteservice.org), while traffic data are simulated using a real data source to feed the simulator.

The DynNav prototype described here has been shown in an OMA demo session (see the presentation and the interview at http://openmobilealliance.org/video/telecom-italia-and-politecnico-di-torino-dynamic-navigation/).

Figure 3. A possible DynNav message flow. A lightweight navigation device interacts with a server to create a trip, read the best route, and create the subscription needed to obtain possible events associated with that trip.
Tests
We focused on a common worst-case usage pattern in which a lightweight DynNav client asks for all the information related to a trip, delegating route computation to the server. This implies that the client device is forced to request even the data necessary to display the chosen path on the map.

First, we measured the amount of data generated by the protocol, which, if excessive, could have a negative impact on performance in low-bandwidth environments. The overall results, computed by averaging several trips of different complexities, showed that DynNav messages are on average 40 percent bigger than Google messages. However, this overhead also includes information about real-time driving times and events associated with the route (such as traffic jams or accidents), which Google Directions doesn’t report. To limit this overhead, we enabled the HTTP compression supported by most HTTP implementations, which reduces the total bytes transferred. In our experiments, we observed a compression rate of about 2.8:1 in route messages, with a 10 percent processing time increase (due to compression and decompression) in the ND. With compression enabled, DynNav messages are on average 35 percent bigger than (compressed) Google messages.

In the second test, we measured the time elapsed between starting a route request and receiving the associated response. This time needs to be as short as possible because it represents the waiting time for the ND user. Tests showed that the average latency for obtaining the route is approximately 300 ms, excluding the time required by the server to compute the route, calculated on 25 realistic trips of different complexity. Although the average latency is three times that experienced in Google Directions, it is still reasonable because the difference is barely noticeable to humans in terms of responsiveness between the two systems. The higher latency is due to the larger size and number of messages (that is, HTTP GET requests) needed to complete the same operation. This is required to enable greater flexibility in the protocol and support different deployment scenarios with a limited set of primitives, such as NDs with and without route computation capabilities or navigation maps.

A third test evaluated the protocol’s processing overhead in the ND, with the final objective of assessing the possibility of executing DynNav services on low-end user terminals. Our measures showed that the average processing cost for a message transporting route information in the selected ND is about 2.5 ms for each kilometer of the route, which represents an acceptable value even for long routes. This confirms that the choice of using rich XML messages, which are known to be more computationally demanding than binary encoded messages, does not represent an issue on modern user terminals, at least in this use case.

Table 1 shows how the performance of the system changes when dealing with routes of increasing lengths, using HTTP compression. These data confirm that the DynNav performance figures remain acceptable within the typical route complexity range.

<table>
<thead>
<tr>
<th>Route length (km)</th>
<th>Message size (Kbytes)</th>
<th>Transmission time (ms)</th>
<th>Processing time on client (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Google Directions</td>
<td>DynNav</td>
<td>Google Directions</td>
</tr>
<tr>
<td>53.73</td>
<td>4.29</td>
<td>5.24</td>
<td>72.15</td>
</tr>
<tr>
<td>244.84</td>
<td>7.18</td>
<td>9.34</td>
<td>76.30</td>
</tr>
<tr>
<td>600.96</td>
<td>20.63</td>
<td>31.48</td>
<td>95.62</td>
</tr>
<tr>
<td>878.95</td>
<td>34.07</td>
<td>45.53</td>
<td>114.88</td>
</tr>
<tr>
<td>1,034.00</td>
<td>29.40</td>
<td>39.43</td>
<td>108.19</td>
</tr>
</tbody>
</table>

Based on the new scenarios enabled by DynNav and on our experiments, we conclude that DynNav has the potential to change the way navigation services are conceived, implemented, and deployed by making them more open and interoperable. Improvements are also possible, and OMA is working on an enhanced version. Future work should be directed to optimize the definition of resources and to add new resources, such as detailed parking information, public transportation, indoor navigation, and weather conditions. In fact, the warm acceptance of this standard among...
different players (ND and smartphone manufacturers, and telecom operators) is pushing for further evolutions, particularly with respect to value-added services (including POI and support for vertical applications such as logistics). Additional studies, in collaboration with other mobile operators, are currently ongoing to define optimized mechanisms for reducing and compressing the amount of transferred data.

The prototype that has been developed to validate the DynNav solution shows excellent results, even on a nonoptimized implementation. Although in some cases, the performance looks inferior to that of proprietary solutions (Google Directions)—albeit hardly noticeably by end users—our DynNav prototype enables interoperability and greater flexibility thanks to its additional features, such as push-based notification services and customizable real-time information.

References

Piergiuseppe Bettassa is a PhD student in the Control and Computer Engineering Department at Politecnico di Torino, Italy. His research interests include formal methods applied to security protocols and security-aware applications, software engineering, and model-driven development. Contact him at piergiuseppe.bettassa@polito.it.

Fulvio Risso is an assistant professor in the Control and Computer Engineering Department at Politecnico di Torino, Italy. His research interests include network protocols, traffic analysis, efficient packet processing, and software-defined networks. Risso has a PhD in computer engineering from Politecnico di Torino. Contact him at fulvio.risso@polito.it.

Riccardo Sisto is a professor of computer engineering in the Control and Computer Engineering Department at Politecnico di Torino, Italy. His research interests include formal methods and communication protocol engineering. Sisto has a PhD in computer engineering from Politecnico di Torino. Contact him at riccardo.sisto@polito.it.

Cecilia Corbi is the application and services standards manager at Telecom Italia. Her research interests include mobile networks and services, and standardization. Corbi has more than 25 years of experience in the telecommunications field, has worked as a project manager for cellular network operators in several areas of expertise in innovation and strategy, and serves as board vice chair of the Open Mobile Alliance. Contact her at ceciliamaria.corbi@telecomitalia.it.

Giorgio Ghinamo is a researcher at Telecom Italia. His research interests include positioning technologies and related applications. Ghinamo contributes to navigation-related standardization frameworks in the Open Mobile Alliance. Contact him at giorgio.ghinamo@telecomitalia.it.

Francesco Vadalà is a researcher at Telecom Italia, where he works in innovation, technical strategy, and R&D, including analyzing emerging technologies and coordinating standards activities in service layer areas. Vadalà serves as the technical plenary chairman of the Open Mobile Alliance. Contact him at francesco.vadala@telecomitalia.it.

Pietro Leggero is a senior software engineer and head of mobile development at Reply. His interests include Android, Java, RESTful Web services, geolocation, platform as a service, and near-field communication technologies. Leggero develops and contributes to investigations on new emergent mobile technologies. Contact him at pietro.leggero@gmail.com.