Research School for Operations Management and Logistics

Reference Architecture for Mobility-Related Services A reference architecture based on GET Service and SIMPLI-CITY Project architectures

A. Husak, M. Politis, V. Shah, R. Eshuis, P. Grefen

Beta Working Paper series 477

BETA publicatie	WP 477 (working paper)
ISBN	
ISSN	
NUR	804
Eindhoven	July 2015

Reference Architecture for Mobility-Related Services

A reference architecture based on GET Service and SIMPLI-CITY Project architectures

A. Husak, M. Politis, V. Shah, R. Eshuis, P. Grefen
School of Industrial Engineering
Eindhoven University of Technology

Table of Contents

Table of Figures

Table of Tables

1 Introduction

This report is dedicated to the design of a reference architecture for mobility-related services.The main aim is to facilitate the design of concrete architectures of business information systems that realize mobility services using state of the art information technology (Angelov, Grefen, & Greefhorst, 2012). The reference architecture is designed based on the architectures of the services of two European projects - Green European Transportation (GET Service) and SIMPLI-CITY.

The Service Platform for GET Service provides transportation planners and drivers of transportation vehicles with the means to plan, re-plan and control transportation routes efficiently and in a manner that reduces the emission of CO2. The GET Service project advances current transportation and route planning systems to the next major level by empowering transport management and route planning systems with information from multiple sources and enabling the incorporation of transportation-related tasks into transportation planning. In doing so, GET Service facilitates (Velde, Saraber, Grefen, & Ernst, 2013):

- real-time aggregated planning
- synchromodal re-planning
- reduction of empty miles
- co-modal planning.

SIMPLI-CITY - The Road User Information System of the Future is a project funded by the European Commission to promote the usage of road user information systems and help drivers make their journey safer, more comfortable and environmentally friendlier. SIMPLI-CITY provides the technological foundation for bringing the "App Revolution" to road users by facilitating data integration, service development and end user interaction (Abels, 2013).

GET Service and SIMPLI-CITY are thus projects concerning mobility-related services. Both GET Service and SIMPLI-CITY have their business information system architectures in place. The main aim of this report is to design a reference architecture for mobility-related services, based on analyses of GET Service and SIMPLI-CITY architectures. A reference architecture is designed such that it can serve as a blueprint for projects in which architectures for mobility-related services have to be developed.

The report starts with an overview of the GET Service and SIMPLI-CITY projects. After briefly discussing the two projects, a comparison is made, based on criteria such as the project vision, main goal, target users, tools and approaches and expected results/impact. Section 3 focuses on the architecture of GET Service and the various components of the architecture are described. In Section 4, the architecture of SIMPLI-CITY is described and the components making up the architecture are explained. The concept of a reference architecture is defined in Section 5. Examples of reference architecture are provided to give the reader a clearer idea of reference architectures.

Considering the common building blocks and also the unique and important components of the GET Service and SIMPLI-CITY architectures, a reference architecture is constructed in Section 6. Each of the components i.e. Client Side, Server Side, External Sources and Developer Support is explained. The subcomponents that form these components are also described. In Section 7, the architectures of GET Service and SIMPLI-CITY are projected to the reference architecture. This is done to validate the proposed reference architecture and to see how the reference architecture is applicable to the two architectures of GET Service and SIMPLI-CITY. In Section 8, we present a brief overview of related work. The conclusion is presented in Section 9 and the main contribution of the report is reiterated.

2 Overview of GET Service and SIMPLI-CITY projects

This section first provides an overview of GET Service and SIMPLI-CITY projects. After briefly discussing the GET Service and SIMPLI-CITY projects, an in depth comparison of the two projects is conducted. The vision, main goal, target users, tools and approaches and the expected results/impact of both the projects are compared.

2.1 Overview of GET Service

As mentioned in Section 1, the Service Platform for GET Service provides transportation planners and drivers of transportation vehicles with the means to plan transportation routes more efficiently and respond quickly to unexpected events during transportation. To this end, it connects to existing transportation management systems and improves on their performance by enabling sharing of selected information between transportation partners, logistics service providers and authorities.

Currently, basic systems for transportation and route planning exist. The GET Service platform (Velde, Saraber, Grefen, & Ernst, 2013) lifts these systems to the next major level, by:

- Enabling improved transportation and route planning, by incorporating transportation and logistics-related tasks, such as transfer of goods and administrative tasks, into the planning;
- Facilitating more accurate transportation and route planning, by using real-time information from multiple information sources;
- Facilitating quick effectuation of changes to transportation plans, including the execution of necessary transportation-related tasks, such as (de-)reservation of necessary resources and unloading of already loaded goods;
- Enabling holistic planning, where transportation routes and placement of transportation resources is planned jointly to optimize resource usage.

To achieve these objectives, the GET Service platform is developed, with subsystems for information aggregation, real-time planning, transportation control and transportation service development.

The project as a whole has the aim to design a service platform for joint transport planning and execution to improve key performance indicators (KPIs) such as transport costs, CO_2 emissions and customer service. To this end, the project will develop a proof concept service platform in which planners of transport receive real-time planning options when scheduling their operations. The options involve inter-modal transport (i.e. road, rail and barge). If unexpected events occur, planners will receive (re-)planning options, including necessary steps to take to effectuate changes to the transport plan (for instance communication with customs, terminals, etc.). Furthermore, the transport execution will be supported by the GET Service Platform by functionalities such as transport monitoring and control as well as data exchange.

2.2 Overview of SIMPLI-CITY

“SIMPLI-CITY – The Road User Information System of the Future” will foster the usage of full-fledged road user information systems – helping drivers to make their journey safer, more comfortable, and more environmentally friendly. Therefore a holistic framework is needed which structures and bundles potential services that could deliver data from the various sources to road user information systems as well as allows road users to make use of the data and to integrate it into their driving experience (Abels, 2013).

Thus, by providing such a framework, SIMPLI-CITY will facilitate two main results:

- A next-generation European wide service platform allowing the creation of mobility-related services as well as creation of corresponding apps. This will enable third party providers to create a wide range of interoperable, value-added services, and applications for drivers and other road users.
- An end user assistant allowing road users to make use of the information provided by applications and to interact with them in a non-distracting way – based on a speech recognition approach.

To reach these results, SIMPLI-CITY will define tools and approaches in following areas:

- Adding a “software layer” to the hardware driven “product” mobility, analogously to the “App-Approach”. Thereby, the project will support third party developers to efficiently realise and sell their mobility-related service ideas by a range of methods and tools, including the Mobility Services and Application Marketplaces.
- Delivering a Mobility Service Framework as the foundation for all services and end user applications facilitated by the project. Based on the latest developments in Service-oriented Computing, the framework will introduce mobility-specific extensions allowing service invocation on different mobile devices whilst considering connectivity quality and other context-related information.
- Following the Mobility-related Data as a Service approach allowing data from sensors, cooperative systems, telematics, open data repositories, user-centric sensing, and media data streams to be modelled, accessed, and integrated in a unified way. The Mobility-related Data as a Service approach of SIMPLI-CITY goes far beyond the current possibilities to exploit data in common apps: It will enable service developers to easily integrate data coming from arbitrary, technologically heterogeneous sources.
- The elaboration of the SIMPLI-CITY Personal Mobility Assistant (PMA). This PMA will allow end users to interact with services in an intuitive, non-distracting way. The PMA is a voice-based, multimodal user interface and execution environment. It allows application interaction without distracting the potential users, notably drivers, in a much more convenient and precise way than current solutions are able to do. New applications can be integrated into the PMA in order to extend its functionalities for individual needs.

While SIMPLI-CITY enables a wide range of mobility-related services, the project will present its real-world application in two dedicated use cases:

- The “Meeting the Increased Mobility Demand” use case will show how SIMPLI-CITY helps users in their journeys to big events and also provides the innovative topic “personalised traffic restrictions”.
- The “Enhancing the Driving Experience” use case will especially focus on the enhancement of environmental friendliness using services developed in SIMPLI-CITY as well as dedicated comfort and leisure services like media streaming.

2.3 Comparison of GET Service and SIMPLI-CITY

Before starting to build a reference architecture based on the GET Service and SIMPLI-CITY projects, it is important to make a comparison between them. In this section, a comparison of the GET service and SIMPLI-CITY projects is provided. Table 1 presents a general comparison of the two projects, based on different aspects such as vision, main goal, target users, tools and approaches and the expected results/impact of each project.

Table 1: Comparison between GET Service and SIMPLI-CITY projects

	GET	SIMPLI-CITY
Vision	**"GET Service - The Service Platform for Green European Transportation"** will support a European transportation ecosystem that is demonstrably more environmentally friendly and efficient and provides new business opportunities for transportation information providers and organizations that can use this information to provide innovative services.	"**SIMPLI-CITY – The Road User Information System of the Future"** will foster the usage of full-fledged road user information systems – helping drivers to make their journey safer, more comfortable, and environmentally friendlier.
Project coordinator	Eindhoven University of Technology, The Netherlands	Vienna University of Technology, Austria
Main Goal	Develop a Service Platform for Green European Transportation (GET Service) that facilitates real-time aggregated planning, synchro-modal (re-planning), reduction of empty miles and co-modal planning.	Develop a next-generation European wide service platform allowing the creation of mobility-related services as well as creation of corresponding applications. An end user assistant will allow road users to make use of the information provided by apps and to interact with them in a non-distracting way - based on a speech recognition approach.
Target users	Logistics Service Providers, Transport Service Providers, Freight Forwarders and truck drivers	Personal car drivers, disabled drivers, truck drivers, cyclists, pedestrians, passengers and application developers
Tools and Approaches	• **Transportation planning & control:** Aggregate real-time data from multiple sources to enable predictive planning. Include transportation-related tasks in planning and control to support more efficient (co-modal) planning. • **Service development & composition:** Make existing techniques applicable in the transportation domain by	• **Software layer:** Adding a "software layer" to the hardware driven "product" mobility, analogously to the "App-Approach". The project will support third party developers to efficiently realise and sell their mobility-related service ideas by a range of methods and tools, including the Mobility Services and Application Marketplaces.

	specializing them, extending them driven by the domain-specific requirements and implementing them for the concrete transportation scenarios of the project. Develop domain specific constructs. • **Service composition orchestration and reconfiguration:** Enable dynamic reconfiguration of service composition to support advanced transportation re-planning problems. Ideally, the project develops dynamic reconfiguration mechanisms that support arbitrary adaptations to a service composition. • **Information aggregation:** Exploit transportation plans and control structures to automatically derive aggregate information needs and missing information.	• **Mobility Service Framework:** Delivering a Mobility Service Framework as the foundation for all services and end user applications facilitated by the project. The framework will introduce mobility-specific extensions allowing service invocation on different mobile devices whilst considering connectivity quality and other context-related information. • **Mobility-related Data as a Service:** Allowing data from sensors, cooperative systems, telematics, open data repositories, user-centric sensing, and media data streams to be modelled, accessed, and integrated in a unified way. The Mobility-related Data as a Service approach of SIMPLI-CITY will enable service developers to easily integrate data coming from arbitrary, technologically heterogeneous sources. • **Personal Mobility Assistant (PMA):** The PMA is a voice-based, multimodal user interface and execution environment that will allow end users to interact with services in an intuitive, non-distracting way. It allows application interaction without distracting the potential users, notably drivers, in a much more convenient and precise way than current solutions are able to do. The PMA functionalities can be extended by integration of new applications into it.
Main Expected Results/ impact	• **Demonstrable reduction of CO2 emission of transportation:** by developing services that enable 'green' planning of transportation and route planning. Particular usage scenarios that will be covered by the planning services include co-modal planning and efficient planning of resources to reduce empty miles. Specific tasks are planned to validate the potential of the platform to	• **Mobility Service Framework:** A next-generation European Wide Service Platform (EWSP) allowing the creation of mobility-related services as well as the creation of corresponding apps. This will enable third party providers to produce a wide range of interoperable, value-added services, and apps for drivers and other road users.

	reduce CO2 emission. • **Optimized planning transportation algorithms:** The GET Service platform aims to make European transportation more efficient, using optimized planning algorithms. Since the profit margins in this sector are very small, improvements in efficiency have a strong impact on the competitiveness of the transport industry as a whole. • **Information provisioning services and services for green transportation planning:** Business models will be developed for monetizing the services provided by the GET Service platform. Particular focus will be on information provisioning services and services for green transportation planning. During the development of these business models, the markets that benefit from the GET Service platform and its services will be determined.	• **Mobility-related Data as a Service:** The integration of various, heterogeneous data sources like sensors, cooperative systems, telematics, open data repositories, people-centric sensing, and media data streams, which can be modelled, accessed, and integrated in a unified way. • **Personal Mobility Assistant:** An end user assistant that allows road users to make use of the information provided by apps and to interact with them in a non-distracting way – based on a speech recognition approach. New apps can be integrated into the Personal Mobility Assistant in order to extend its functionalities for individual needs.

3 GET Service Architecture

This section discusses the various components that make up the GET Service architecture as described in GET Architecture Definition (Velde, Saraber, Grefen, & Ernst, 2013). Since each component consists of a number of subcomponents, a brief description of each of these subcomponents is also given.

3.1 Architecture Overview

The major components of the GET Service architecture are as depicted in Figure 1:

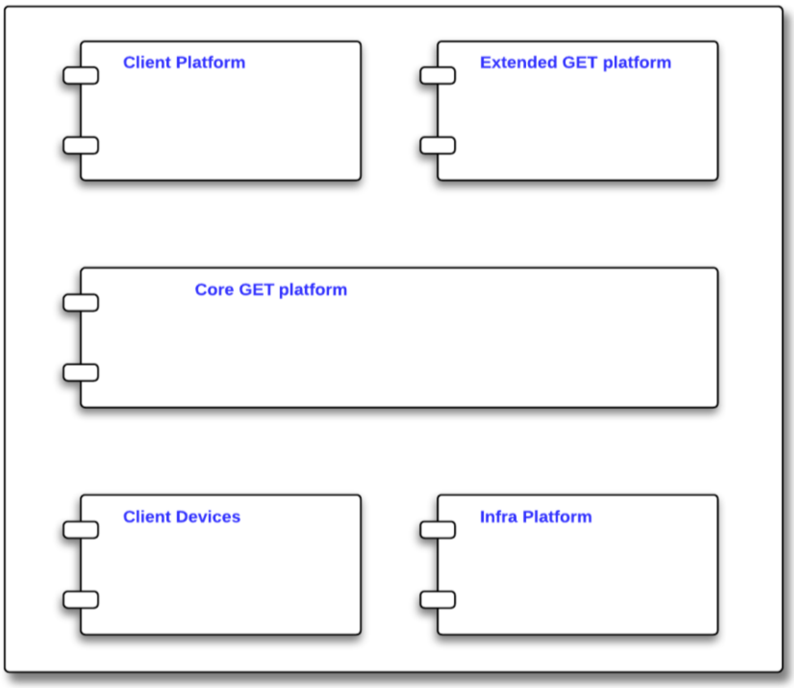


Figure 1: Major Components of the GET Service architecture

In the Figure 1 the highest aggregation level of GET Service is presented. In the following sections we vertically move down to the lower aggregation levels of the cube (Grefen, Business Information System Architecture, Spring 2014) exploring each of the levels of the GET Service. A description of the different components of the GET Service architecture and their subcomponents is provided in the next section.

3.2 Components

In this section, we describe the main components of the GET Service architecture.

3.2.1 Client Platform

The Client platform provides the Transport Order information to GET Service platform, which is used in order to generate transport options for the Offline planner. The Client platform receives information from the GET Service platform on the capacity that is booked or is still available. Multiple Client platforms can be connected to the centralized GET Service platform: a client platform represents a Transport Management System (TMS) for each participating Logistic Service Provider. The Client Platform consists of 2 components as presented in Figure 2:

- Backend system
- Planner

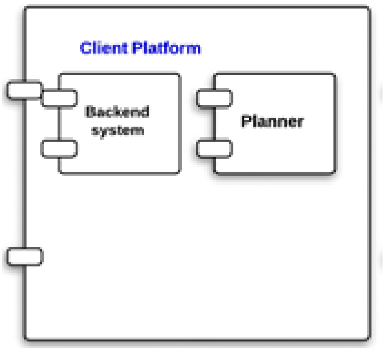


Figure 2: GET Service Client Platform

Table 2: GET Service Components - Client Platform

Components	Description
Backend System	Existing Transport management System used by the client to manage the transport orders. The backend system can be directly coupled to the Client Device
Planner	The logic and resulting interfaces between the Extended GET Service platform and the Backend system for the offline, asset and execution planner. The planner requests alternative transport plan options (including updates) from the proposer in the Extended GET Service platform

3.2.2 Client Devices

Client devices are the on-board systems. A mobile operator sends status and location information ("whereabouts") to the GET Service platform by means of on-board devices, apps, and phones. This information can be sent directly to the GET Service platform, or via the Client Platforms. As shown in Figure 3, Client Devices consists of:

- Process Client
- Event Source

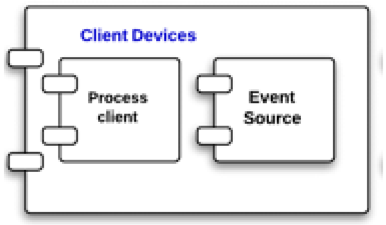


Figure 3: GET Service Client Devices

Table 3: GET Service Components - Client Device

Components	Description
Process Client	Provides current status information about the transport process from on-board device. The client can receive updated instructions on a running transport process from the client platform (synchronization interface). This flow takes place outside the GET Service platform. Adding this flow to the GET Service System will add complexity, since the GET Service platform will have to be interoperable with a wide range of existing systems.
Event Source	Translates on-board data (static/dynamic) into events and publishes the events to the Core GET Service platform.

3.2.3 Infra Platform

An Infrastructure Information Provider distributes information on traffic, tides, bridges and locks. The GET Service platform can subscribe to events that contain infrastructure status information. GET Service does not send any information back to these parties. As depicted in Figure 4 the Infra Platform has the Event Source as its subsystem component.

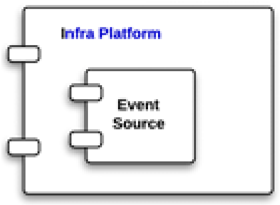


Figure 4: GET Service Infra Platform

Table 4: GET Components - Infra Platform

Components	Description
Event Source	Translates infrastructure data (static/dynamic) into events, publishes the events to the Core GET Service platform.

3.2.4 Core GET Service Platform

The Core GET Service platform correlates and aggregates the external events from multiple Infra and Client platforms. It also contains an Information warehouse which stores static (schedules, master data) and dynamic (capacity) data. The aggregated events are published or can be retrieved by Client platforms or the Extended GET Service Platform. The Core GET Service Platform consists of:

- Event Channel
- Event Store
- Information Store
- Event Aggregator

- Event Correlator
- Subscription Store
- Log Manager
- Community Passport Manager

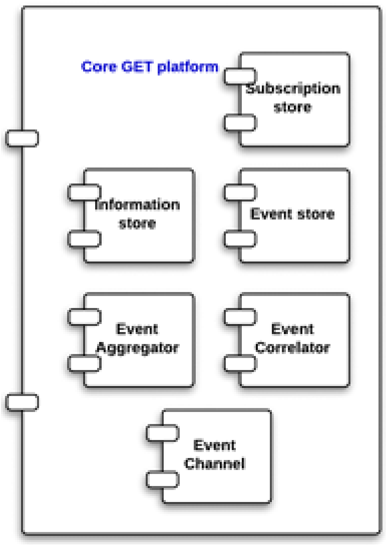


Figure 5: GET Service Core Platform

Table 5: GET Service Components - Core Platform

Components	Description
Event Channel	Receives & normalizes events from the ***Event sources*** through subscribe interfaces. The Events are stored in the ***Event Store***.
Event Store	Stores all raw/aggregated events for a predefined time window. Aggregated events from the event store can be retrieved by the Client Platform and also by the Extended GET Service Platform.
Information Store	Stores data that can be used in offline planning. Capacity and schedule data from the ***backend system*** is also stored in the Information store. Static/dynamic data from the Information store can be retrieved by the Client Platform (Query interface). Aggregated events from the Information store can be retrieved by the Extended GET Service Platform (Query interface).
Event Aggregator	Events can be combined based on dynamic rules and filters. The aggregated events are put back into the ***Event store.*** The ***proposer*** and ***orchestration*** engine subscribe to published aggregated events.
Event Correlator	Matching function by using static and dynamic data from the ***Information store*** and the ***Event store.*** The Event Correlator uses an event model to identify events, and a Query language to correlate them.
Subscription Store	The subscriptions for the Extended GET Service platform (***proposer & planner***) are stored here.

Log Manager	The GET Service project will use a Core Platform Component to track messages. The Log Manager provides interfaces for logging and listing messages.
Community Passport Manager	As part of the Core Platform the Community Passport Manager (CPM) provides service interfaces for registration and authentication to all GET Service components. The CPM is responsible for a centralized passport store.

3.2.5 Extended GET Service Platform

The Extended GET Service platform is the part of the platform that combines aggregated events, capacity and schedule information and initial transport plans from the planners. From this information, it provides alternative transport plan options which the Execution planner can use. It also contains a Plan & Process warehouse which stores static (plans) and dynamic instance (process) data. The Extended GET Service Platform consists of:

- Proposer
- Process Store
- Orchestration Engine
- Process Development Environment

Table 6: GET Service Components - Extended Platform

Components	Description
Proposer	The Extended GET Service platform proposes a (recalculated) transport plan to the ***planner*** in the Client platform. The proposer can propose alternatives for running processes that are in the ***process store.***
Process Store	Dynamic instance data is persisted for each running process. This assures that the process can continue after a system restart The Client platform can also retrieve dynamic instance (process) data.
Orchestration Engine	Is used to orchestrate tasks for a running process. External events can influence the process path. This can result in a new / updated transport plan by the proposer.
Process Development Environment	Used to develop new processes + orchestration steps

A complete picture of the GET Service architecture is depicted in Appendix B.

4 SIMPLICITY Architecture

In this section we discuss the different aggregation levels of the SIMPLI-CITY architecture. Each aggregation level is comprised of components and subcomponents. A brief description of each of these subcomponents is also given.

4.1 Architecture overview

The diagram in Figure 6 gives an overview of the architecture of SIMPLI-CITY. As shown in the figure, the architecture of SIMPLY-CITY may be split into four major components. These major components are, the “Vehicle & PMA (Personal Mobility Assistant)”, the “SIMPLI-CITY Server Side”, the “External Data Sources” and “Developer Support”.

The first major component focuses on the personal mobile device, which will be the most visible element of SIMPLI-CITY from an end user perspective, this major component is called “Vehicle & PMA”. The second major component covers subcomponents that are located in the “Server Side”, meaning that they do not run inside the mobile device. The third major component of the SIMPLI-CITY architecture represents the “External Data Sources” and the last major component called “Developer Support” contains subcomponents that support developers in the creation, deployment and updating of applications and services.

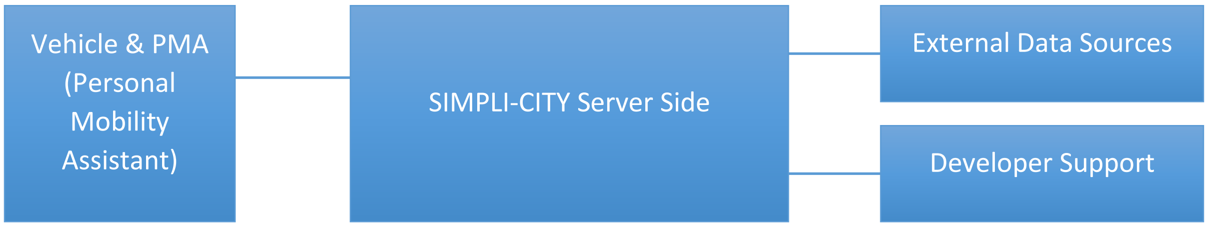


Figure 6: SIMPLI-CITY Architecture

A more detailed diagram of the SIMPLI-CITY architecture can be found in Appendix C where all the subcomponents of each major component are presented.

4.2 Components

In this section, the four different major components of SIMPLI-CITY architecture are described. A general overview of each of the subcomponents contained in the four major components is provided.

4.2.1 Vehicle & PMA

This section describes all components that are located at the mobile device or within the vehicle environment. All of these components together form the personal mobility assistant which is the main contact point between SIMPLI-CITY and the end-user. End users will use this mobile device to access all SIMPLI-CITY functionality by making use of apps. The architecture of the Vehicle & PMA component of SIMPLI-CITY service, is presented in Figure 7, and described briefly in Table 7.

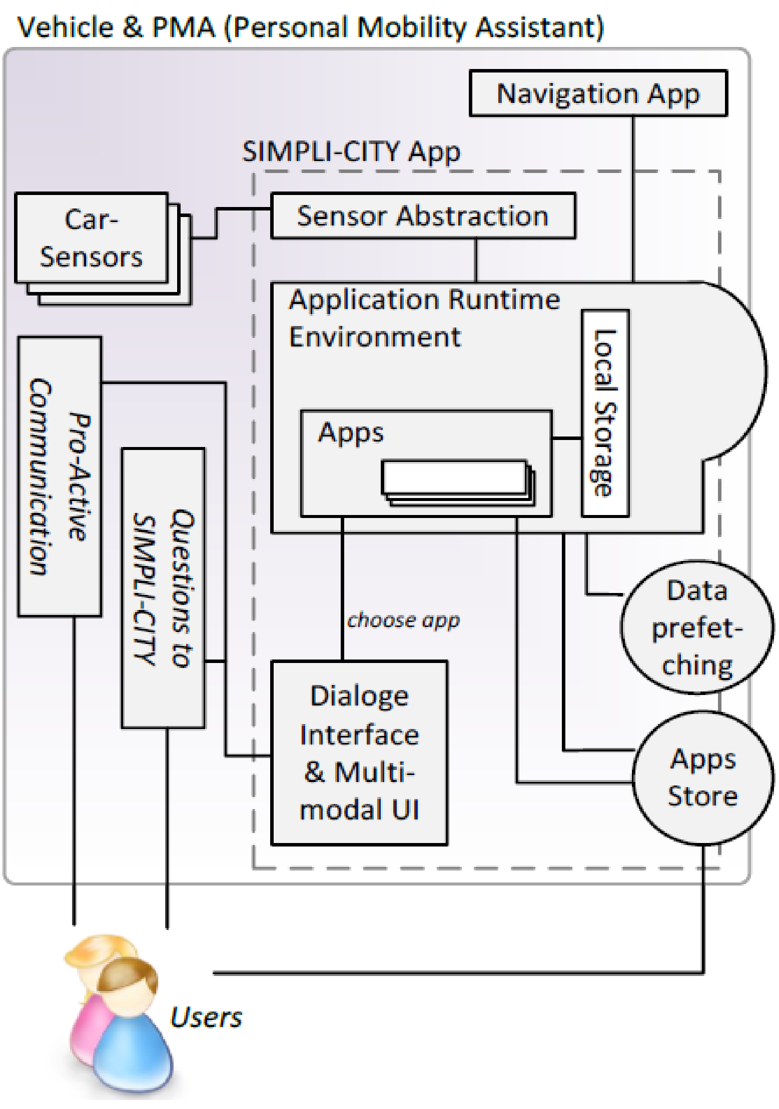


Figure 7: SIMPLI-CITY Vehicle & PMA Component

Table 7: Component Descriptions - SIMPLI-CITY Vehicle & PMA

Components	Description
Application Runtime Environment	Meets the requirement that the central element for user acceptance is the extendibility of SIMPLI-CITY. The project is not a closed information system but it rather allows developers to add own services and apps to it.
Application Marketplace	Allows users to find new apps in order to add new functionalities to the their mobile device – the PMA. Also allows developers to add their apps to the marketplace.
Dialogue Interface & Multimodal User Interface	This component is the user interface layer of SIMPLI-CITY, taking user input in the form of utterances managing the need for further user input, and transforming them into applications calls. The result of the application call is then fed back to the user, using speech.
Sensor Abstraction	Is the local extension to the sensor abstraction and interoperability interfaces described in the Table 8. In contrast to that component, this component is a service running locally on the mobile device.

4.2.2 SIMPLI-CITY Server Side

This section describes the server-side component of SIMPLI-CITY. These components will handle service executions, cloud based storage requests and other issues like data source integration and access to external sensors. The communication between these components and the apps of SIMPLI-CITY will be performed via the application runtime environment and the service runtime environment.

Figure 8 and Table 8 show the components that are not located on the personal mobility assistant. These components together create the European Wide Service platform, which is a major outcome of SIMPLI-CITY.

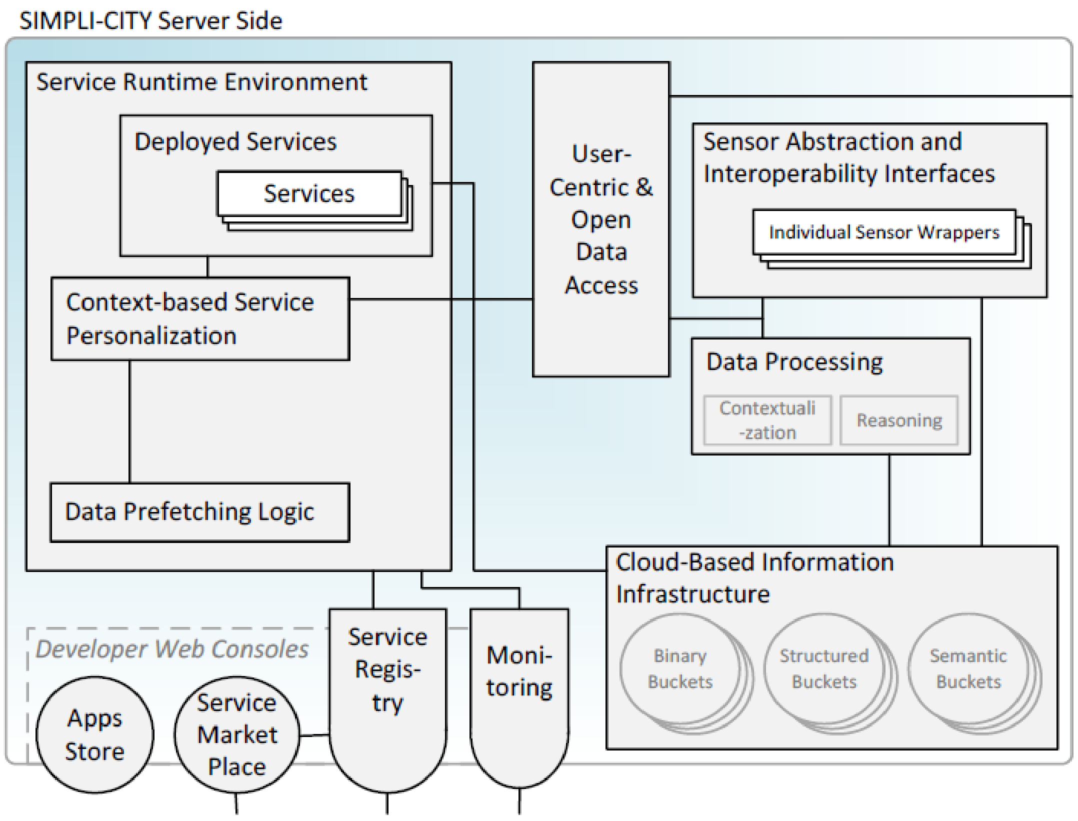


Figure 8: SIMPLI-CITY Server Side

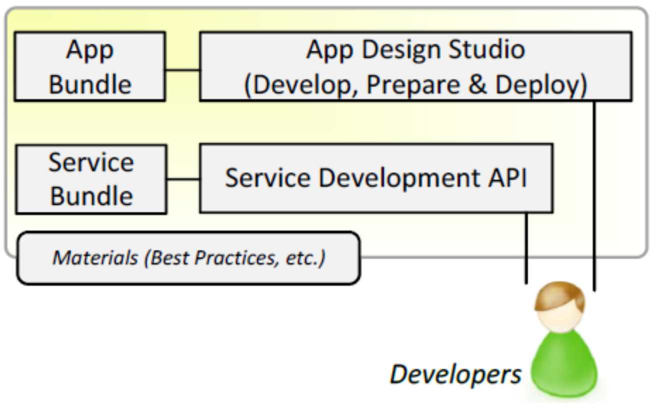
Table 8: Component Descriptions - SIMPLI-CITY Server Side

Components	Description
Service Runtime Environment	Provides a deployment and the execution framework for (composed) services, which offer the business logic for end user apps in SIMPLI-CITY, and provides additional features that are required in conjunction with the execution of services.
Media Data Streams / Data Prefetching Logic	Handles the data prefetching and media streaming aspects of SIMPLI-CITY. As such it may be split into two aspects: 1) Streaming and prefetching (personalized) media information by reacting to app request and pre-buffering media data. 2) Prefetching information from

	data services by automatically invoking data services and buffering their content.
Context- Based Service Personalization	Deals with services in SIMPLI-CITY that can make use of context information in order to realize personalized services for a particular user. Two aspects: 1) Location-based data service selection and 2) proactive user notification (e.g. in the case there is an important update)
Monitoring	In order to check if services are able to meet these non-functional requirements, each service invocation is monitors regarding: 1) services are responsible at all, and 2) service level objectives (SLAs) are being complied with. It also serves the purpose to generate service execution statistics and log error messages if they occur.
Service Marketplace	It will be used by developers for two main purposes: 1) for providing services to other developers allowing them to consume them (also updating service endpoints or activating/deactivating entries.) and 2) for discovering SIMPLI-CITY enabled services.
Service Registry	In general, is used to store information about services and find services based on some search parameters. Also offers service repository functionalities.
Cloud – Based Information Infrastructure	This infrastructure will act as a service which is dedicated into managing different types of data in a persistent, scalable and efficient storage.
User – Centric / Open Data Access	This component comprises sub-components which combine and process incoming data from different sources: 1) User-centric data, 2) Open/Governmental data, 3) Sensor data and 4) Historical.
Sensor Abstraction and Interoperability Interfaces	Has the role to seamlessly integrate heterogeneous sources of sensor reading and providing corresponding sensor data to SIMPLI-CITY. It has to be able to allow pulling of data from several sources and to be able to deal with event based information systems. Each external data source will be registered to the system with a specific wrapper that transforms the external information into a common data format.

4.2.3 Developer Support

The Developers Support component is presented in Figure 9, while a brief explanation of the component is given in Table 9. The component assists developers to develop new applications by giving them the opportunity to use tools, services and existing applications.



App Bundle
App Design Studio (Develop, Prepare & Deploy)
Service Bundle
Service Development API
Materials (Best Practices, etc.)
Developers

Figure 9: SIMPLI-CITY Developer Support

Table 9: Components Descriptions - SIMPLI-CITY Developer Support

Components	Description
Application Design Studio	Offers an appropriate step-by-step procedure to app development.
Service Development API	Is a component aimed at third party developers that allows them to create and configure their own services on the SIMPLI-CITY platform. These services will be later used within end-user applications and will be responsible for the provision of the external data needed during their execution. There are three types of services: 1) data services, 2) backend services and 3) external services.

4.2.4 External Data Sources

There is a range of external data sources (Personal Data sources, Sensors and various data sources), which are shown in Figure 7. These data sources are shown as examples; however, they are provided by external data suppliers and are therefore not components to be implemented within SIMPLI-CITY.

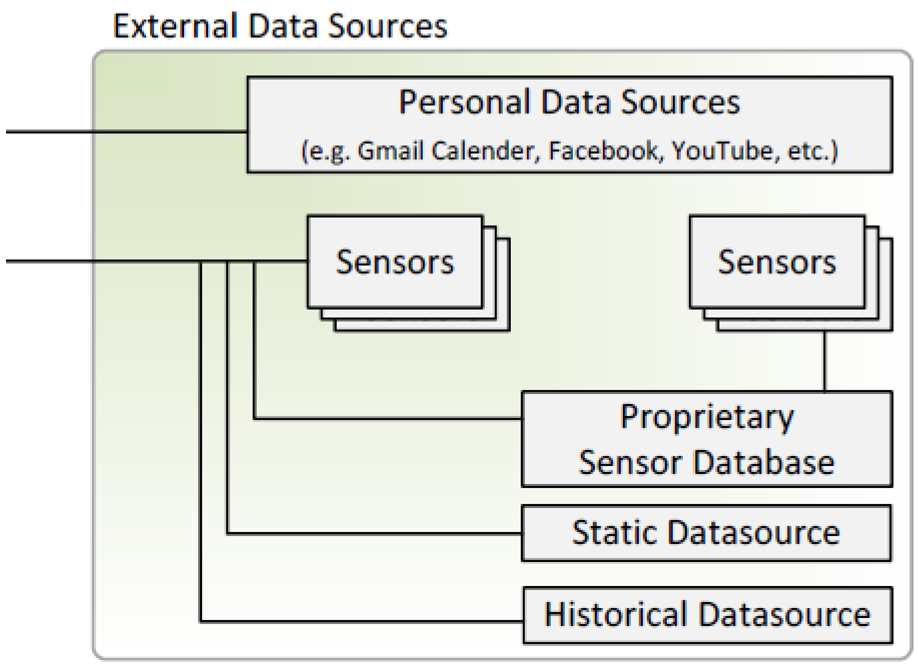


Figure 10: SIMPLI-CITY External Data Sources

5 Reference Architecture Definition

Many projects spend an extreme amount of time researching, investigating, and pondering architectural decisions (See Definition 1). This is inefficient when it is clear that if prior project teams had taken the time to document their experiences and build up a reference architecture.

Definition 1: The architecture of a software system defines that system in terms of computational components and interactions among those components.
(Software Architecture; Shaw & Garlan; Prentice Hall, 1996)

A reference architecture (See Definition 2) is a resource containing a consistent set of architectural best practices for use by other teams. It is helpful to have a high-level, general architecture designs that can be reused and tailored for specific situations – such that the wheel does not need to be reinvented over and over again.

Definition 2: A reference architecture is a general design (abstract blueprint) of a structure for a specific class of information. (Grefen, Business Information System Architecture, 2014)

“Harvesting” of best practices is the first step towards building a strong, versatile architecture (Reed, 2002). Briefly, a reference architecture consists of information accessible to all project team members that provide a consistent set of architectural best practises.

A reference architecture provides a template, often based on the generalization of a set of solutions, these solutions may have been generalized and structured for the depiction of one or more architecture structures based on the harvesting of a set of patterns that have been observed in a number of successful implementations. Further it shows how to compose these parts together into a solution. Reference Architectures will be instantiated for a particular domain or for specific projects.

Adopting a reference architecture within an organization accelerates delivery of a new business information system architecture through the re-use of an effective solution and provides a basis for governance to ensure the consistency and applicability of technology use within an organization.

The following examples are provided to illustrate the importance of reference architectures. Four different reference architectures are presented that are used successfully by large companies (e.g., IBM Java Platform).

Examples:
1) The Java Platform, Enterprise Edition (Java EE) architecture is a layered reference architecture which provides a template solution for enterprise systems developed in Java.
2) The IBM Insurance Application Architecture is a reference architecture for the Insurance domain.
3) AUTOSAR is a component-based reference architecture for automotive software architectures.
4) Unisys 3D Blueprints are a collection of reference architectures from Unisys Corp. for diverse domains such as financial services, communications, public sector, transportation, consumer & industrial products.
(Wikipedia, 2014)

According to the RUP (Rational Unified Process) (Kruchten, 2000), a reference architecture defines levels of abstraction, or "views", thereby providing more flexibility in how it can be used. Ideally, these views map the 4+1 Views of software architecture outlined in the RUP (see Figure 11) and embodied in the RUP's Software Architecture Document.

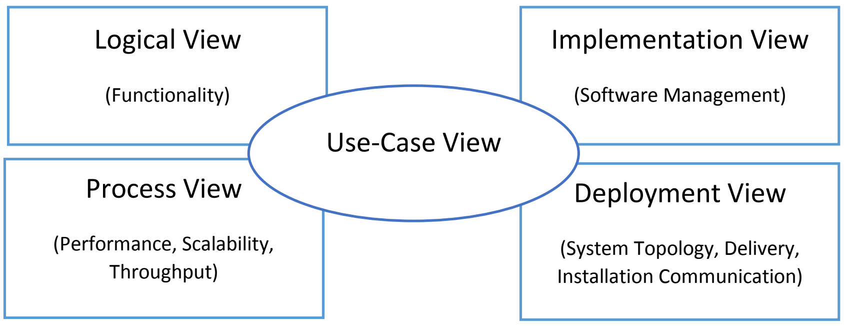


Figure 11: 4+1 aspect framework according to (Kruchter, 1995)

Note that according to the RUP, only the Use Case and Logical Views are used in all projects. The other views should be used if the particular system to be constructed requires them (e.g., the Process View is necessary when there are multiple threads of control; the Deployment View is necessary when the system is to be distributed across more than one node). To describe the logical and the use-case view, we follow the three dimensional cube approach (Figure 12).

In this report we use the presented theory to create a reference architecture based on two architectures described in sections 3 and 4. Both GET Service and SIMPLI-CITY projects provide architectures that can be analysed and finally contribute to the development of a reference architecture. Our reference architecture will be a blueprint of those two projects and by following the use-case view (see Figure 11) we will be able to make the projection of each project. Finally to create the reference architecture we follow the three-dimensional design cube (See Figure 12). We start from a concrete, detailed, Information Technology oriented architecture specification and in a number of steps, we arrive at a more abstract, aggregated and Business-oriented specification. Thus we traverse the cube from the lowest to highest level of abstraction and aggregation.

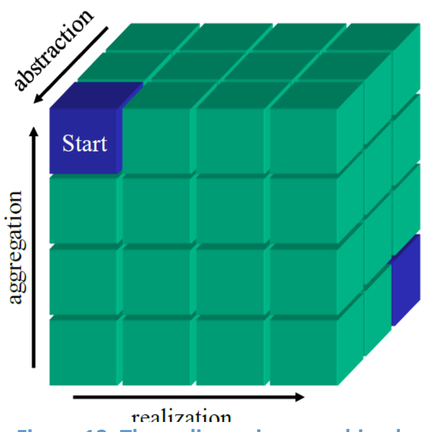


Figure 12: Three dimensions combined (Grefen, 2014)

6 Design of the Reference Architecture

While analysing the GET Service architecture and the SIMPLI-CITY architecture, we observed that both have similar major components. Both GET Service and SIMPL-CITY have a component which acts as an interface between the client and the server. We define this major component as the Client Side. Both GET Service and SIMPLI-CITY have a number of components which form a framework for execution and coordination of services and data storage. This framework offers the business logic for end users of the services and provide additional features that are required in conjunction with the execution of services. We define this framework as a major component which is called the Server Side.

Furthermore, in both architectures, there are components that collect information via personal data sources and sensors and publish them to the Server side. Together these components can be defined as External Data Sources. Then, there exists a component called Developer Support in SIMPLI-CITY. Developer Support will assist developers for creating new services and apps for SIMPLI-CITY and will also provide a studio to prepare the submission and publication of those services and apps. Although this component is not present in GET Service architecture, a component with similar functions is included in the reference architecture.

Thus, we come to the conclusion that there are some common building blocks between the two architectures. Considering these common building blocks and also important components of each architecture, the following reference architecture is constructed (See Figure 13). Each component of the reference architecture is described in detail in the following sections.

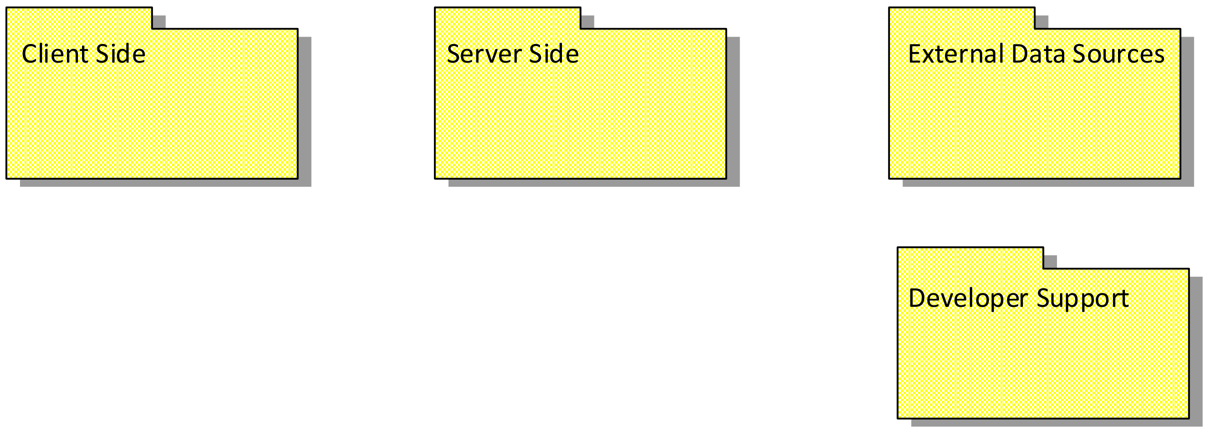


Figure 13: Reference Architecture Overview

6.1 Client Side

As it is described in Sections 3 and 4.2.1 both projects consist of the client side which describes the components which are used by the user/client and are not part of the server side. For SIMPLI-CITY, the client side is as presented in Figure 7 whereas for GET service, the client side is composed of two components (Client Platform and Client Devices) as presented in Figure 1. For the reference architecture our design decision at the aggregation level zero, is to merge (in the case of GET service) the two components to one. In Table 10 a brief comparison of the two projects' Client Side architecture is provided. Hence, it can clarify the decisions about the reference architecture.

Table 10: Comparison of GET Service & SIMPLI-CITY (Client side)

Main Functions of the Client Side		Similarities	Differences
GET Service Client Platform and Client Device	SIMPLI-CITY Vehicle & PMA		
Provide information about events from the truck. *(Event Source)*	Provide information from the Car-Sensors and the current transport process. *(Car-Sensors)*	Provide information (Status from the clients/users device).	SIMPLI-CITY provides extra information, about the current transport process
Provides current status information about the transport from on-board device. The client can receive updated instructions on a running transport process. *(Process Client)*	-	-	For SIMPLI-CITY, such information are provided to the server by the Car-sensors as mentioned above in the table. Furthermore, the updated instructions are provided at the PMA (see below).
The user can request transport plan options from the server. Can also request alternative transport plan options from the server. *(Planner)*	In the PMA the user requests information via the applications to the server. *(Application Runtime Environment)*	Both have similar function, to request information from the server.	In GET Service, the requests are more specific (Transportation plan options), whereas in SIMPLI-CITY the requests can be for several purposes.
The Client uses its own existing Transport management system to manage the transport orders. (Backend System)	The user may have other applications that can provide information similar to SIMPLI-CITY.	The two services can be used while the user/client can obtain similar information from other services.	In the case of SIMPLI-CITY the backend system is not included in the architecture.

In Table 10 we can observe both similarities and differences of GET Service and SIMPLI-CITY client side components in a high aggregation level. The two projects have indeed some differences in their Client Side architecture and the most important one is that the client side of GET service is split into two different components. However, these two components can be easily seen as one and be compared with SIMPLI-CITY's client side as the functionalities of both are similar.

The mapping of the highest possible aggregation level of Client Side to the two projects is presented in Figure 14. More specifically we have a single component for the client side of our reference architecture which when we will arrive at the next aggregation level, will be decomposed. Through this figure it is possible to make a visual comparison of the two projects and identify the similarities and the differences that have been presented in Table 10.

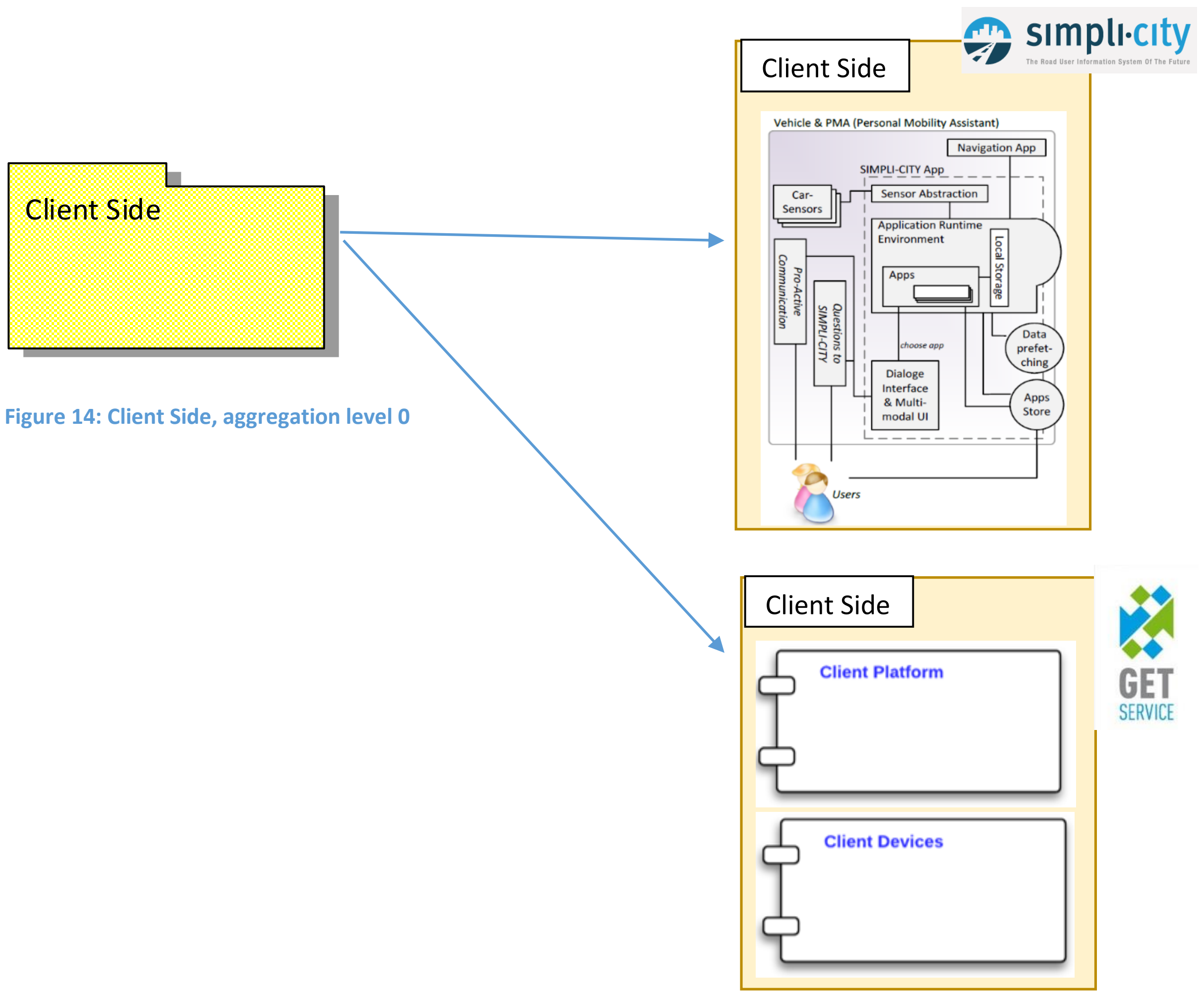


Figure 14: Client Side, aggregation level 0

Moving a step lower in the aggregation level of the three dimensional design cube that was presented in Figure 12, we decompose the Client Side into two components. The two components are the Status provider and the Client Platform (see Figure 15). We see two subsystems, which together provide the same functionality as at level 0.

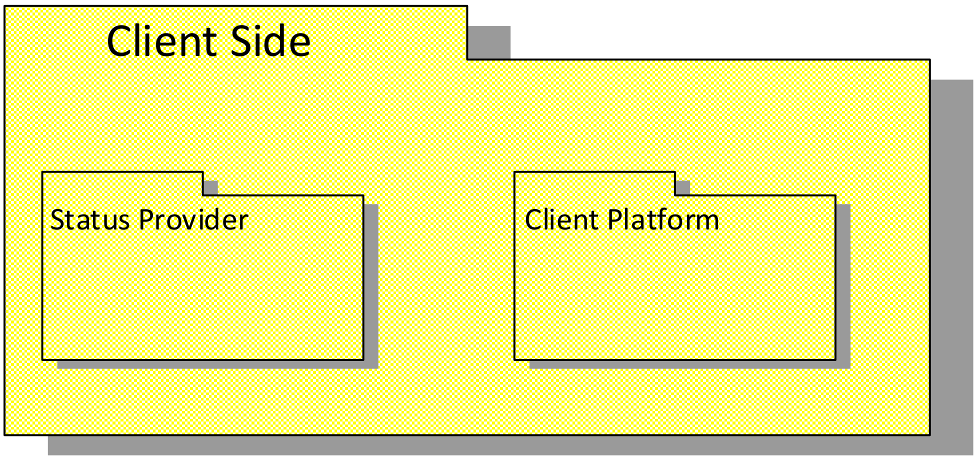


Figure 15: Client Side, aggregation level 1

Client interface is composed by:

- **Status Provider** which provides to the Server Side on-board data, events, information about the transport process and the current status of the vehicle.
- **Client Platform** is the interface of the end user which can communicate and exchange information with the Server Side, also contains further services with similar functionalities of the service which can be used as backend systems.

The decision to decompose the client side into those two different components was made for two reasons. First of all, these two components can represent the complexity of the GET Service, in which we have two separated (also physically) subsystems. Moreover, the second reason is that the two components can exemplify and separate two different functionalities, the data/status provision and the management of the information (e.g., requesting information).

Finally, the least aggregated level of our reference architecture is presented in Figure 16.

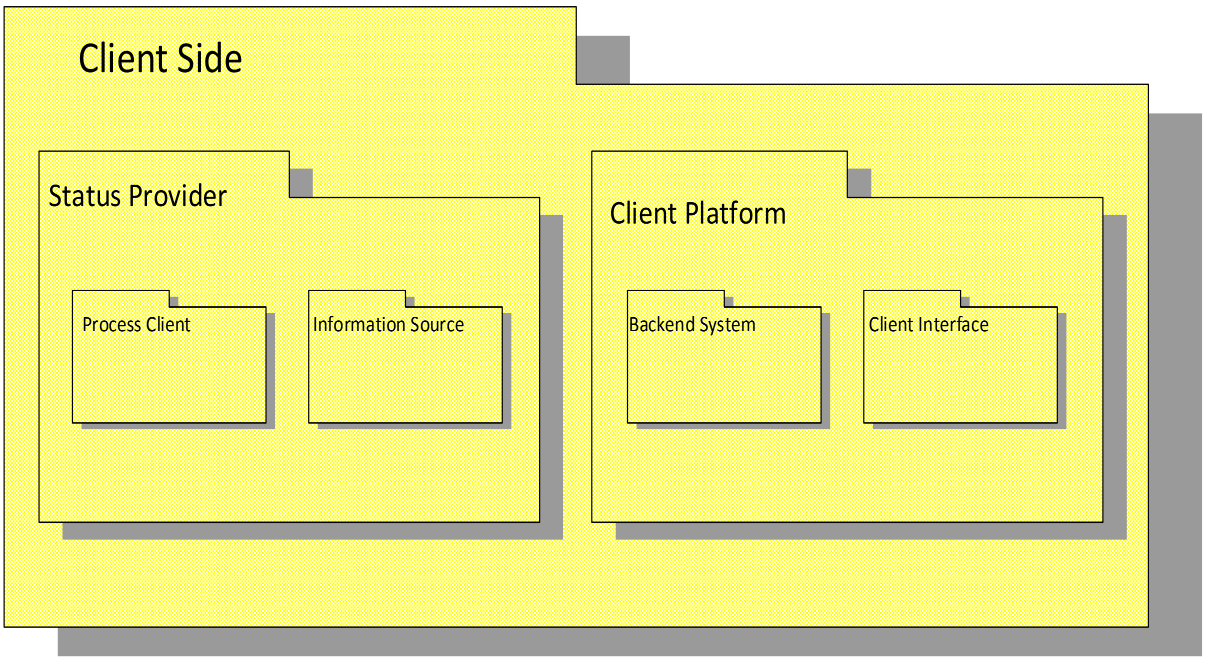


Figure 16: Client Side, aggregation level 2

Table 11: Reference Architecture - Major Client Side Components

Main Component	Inner Component
Status Provider	**Process Client** Provides current information about the transport process from on-board device. The client/user can receive updated instructions on a running transport process from the Server Side via the Client Platform. **Information Source** Provides information from on-board sensors and devices. Publishes the events and gives the current transport process status.
Client Platform	**Backend System** Is the existing applications/services (e.g., transport management systems) that are used by the client/user to manage his requests. **Client Interface** Is the logic and resulting interfaces between the client/user with the server side. Through Client Interface the client/user can request information from the server side and ask questions. Moreover, in that component the type of the application can be selected to be used.

6.2 Server Side

The Server Side of the reference architecture has the main function of handling service executions. It also receives, integrates and stores data from external sources. Data is processed to provide useful information and services to the end users (Client side). Services that the system provides are also monitored by the server.

The Server side for SIMPLI-CITY consists of a number of components as shown in Figure 17. The main component of the server side is the Service Runtime Environment which is responsible for hosting and controlling all deployed services. It also coordinates communication with the PMA. The server side also covers data storage facilities and handles communication with external data sources. For GET Service, the Extended GET Service Platform and Core GET Service Platform together form the Server side. This is because both components play a fundamental role in service architecture and are equally responsible for handling service executions.

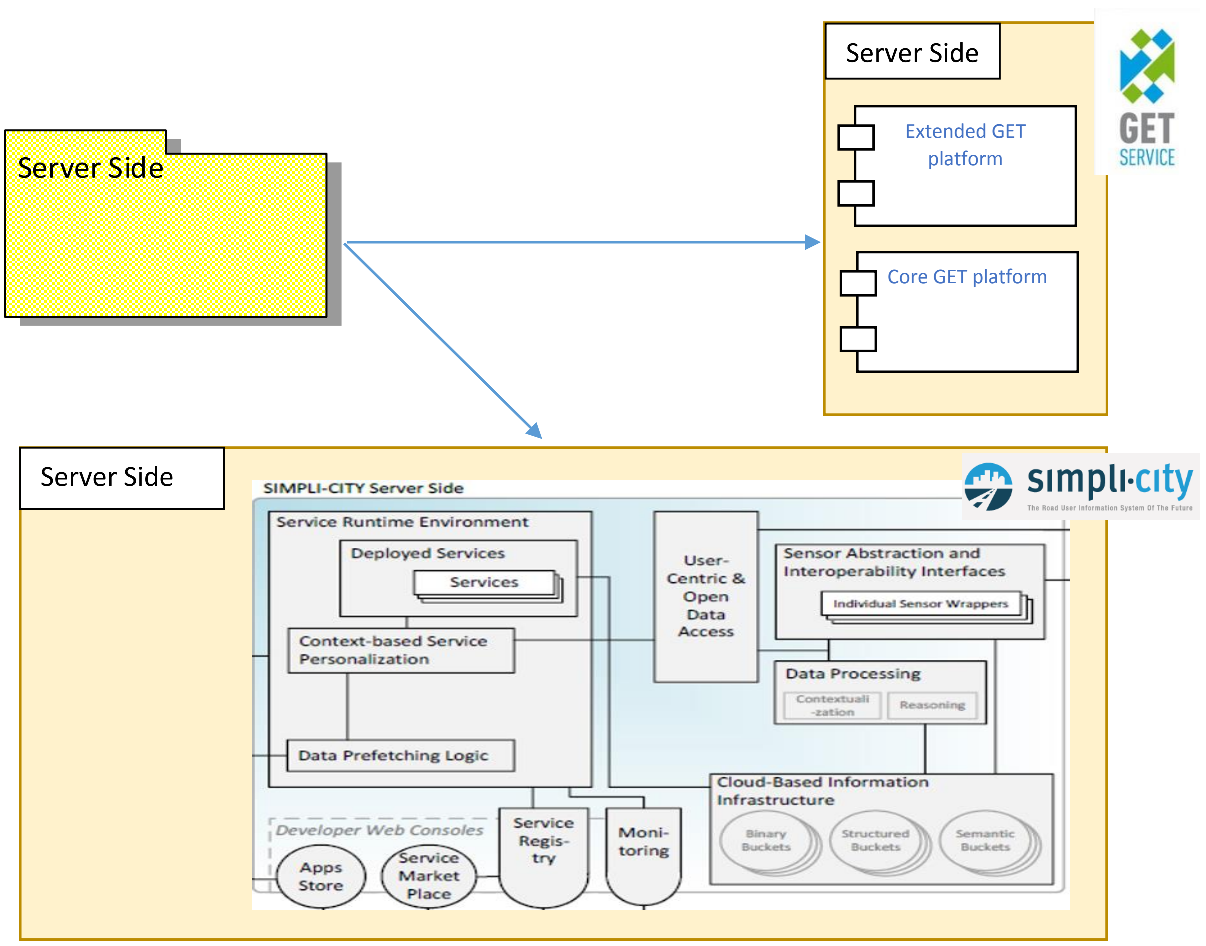


Figure 17: Server Side, aggregation level 0

A lower-level architecture design of the Server side is displayed in Figure 18. The Server side is decomposed to comprise of the **Data Platform, Logic Platform** and **Developer Web Console**.

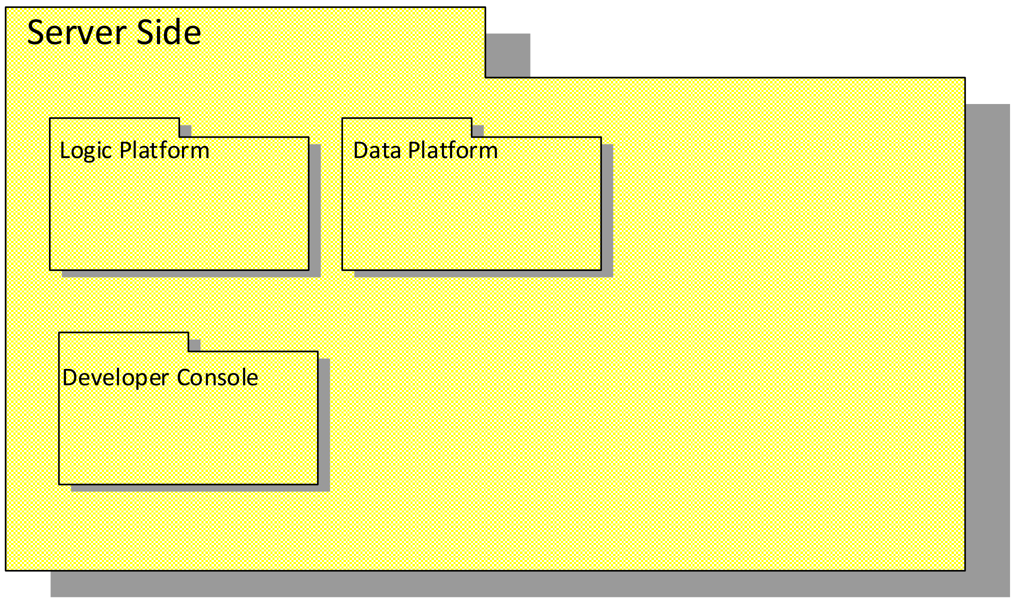


Figure 18: Server Side, aggregation level 1

Logic Platform is concerned with arranging and structuring services before they can be fed to the end user (applications). The Logic Platform also makes use of end user context to figure out what kind of services or data/information could aid the user.

Data Platform collects data or events, processes them and stores them. Data is fed into the Data Platform by external sources or sources within the system. Processing of data can occur in the form of filtering, correlating, merging, summarizing and/or splitting. Once data has been processed it can be passed on to other components or stored in the Data Platform.

Developer Console is a component which provides controlled access to data from the Data and Logic Platforms for governance of the services. It includes the registration and monitoring of services. It is a platform or marketplace where developers or service providers can publish the services they have developed. It can thus be considered to be an evolving catalogue of information about the available services. If the services will be used by end users through applications, such applications are also available in the developer web console for download and installation.

The **Data Platform** and **Logic Platform** can again be decomposed into further subcomponents as shown in Figure 19.

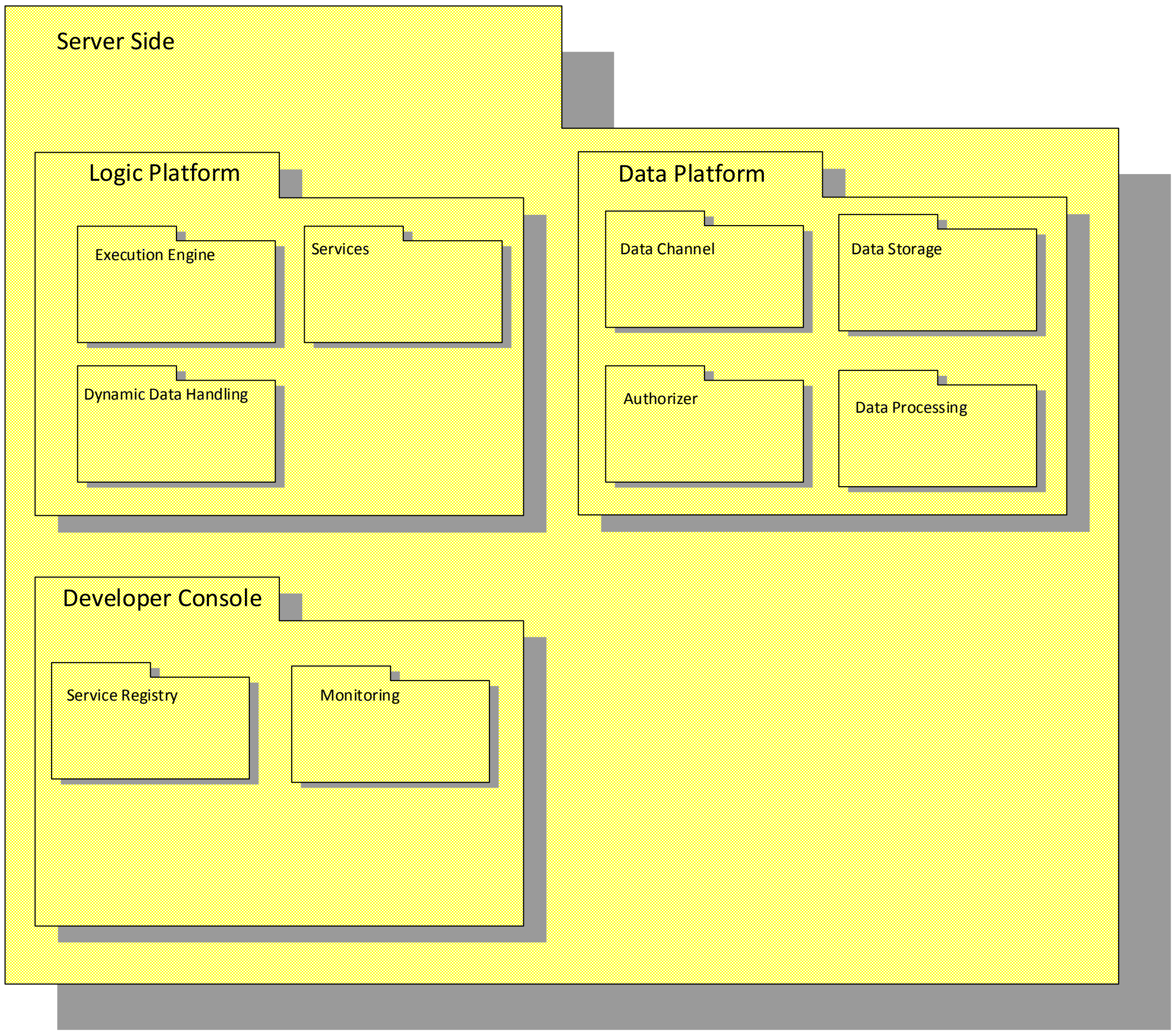


Figure 19: Server Side, aggregation level 2

The Logic Platform consists of:

- **Execution Engine** is responsible for orchestrating and bundling services. Services are used by end users through applications and/or user interfaces. Services can deliver data from various sources to the end user. It can also be the case that such services are coupled to end user applications.
- The **Services** component consists of services which are hosted and controlled by the server. Services can range in number and will have different functions such as providing optimal transportation routes between 2 points, displaying nearest free parking spaces or suggesting alternative routes of transport when an unexpected event occurs.
- **Dynamic Data Handling** persists dynamic instance data. This assures that the process can continue after a system restart. Moreover, it handles the data prefetching and media streaming aspects. Finally, it uses context information in order to realize personalized services for a particular user. Such services can range from asking the user if he wants information about free parking spaces in a city as he approaches the city, providing alternative route/transport plan to the user if he reports a highway blockade because of an accident, to asking the user if he wants to download music in an area where there is free Wi-Fi.

The Data Platform consists of:

- **Data Channel** basically integrates data coming from different sources, processes them and/or publishes them. Data Channel deals with different kinds of data such as vehicle sensor data, on-board data, user profile data, government data, data about certain events, historical data about specific sensor sources, infrastructure data, etc.
- **Data Storage** offers persistent, scalable and efficient storage of data. Data can be of different types - static, dynamic, aggregated, etc. Data can be fed from external data sources or from applications within the system and can be stored for a fixed time frame.
- **Data Processing** filters data or events. It can also correlate, merge, summarize and split data or events. Other components can subscribe to the data or events processed by Data Processing.
- **Authorizer** is responsible for validating the user's login credentials with the saved information. If these two match an authorization token will be returned to the user. The user can use other services through this authorization token. The Authorizer is also responsible for registration of the user.

The Developer Console consists of:

- **Service Registry**: All services that will be offered have to be registered in the Service Registry. Thus, the Service Registry acts as a service directory which is used to store information about services and find services based on some search parameters.
- **Monitoring** component is responsible for overseeing services that the system should deliver. It checks if the services are fulfilling their objectives, if they are responsive and respond within the maximum response time and if any error messages are generated during the execution of a service. Such error messages can be logged. Monitoring also generates service execution statistics.

While analysing both architectures, we identified inconsistencies regarding the placement of components which perform similar functions as Service Registry and Monitoring. In GET Service architecture, the components – Service Registry and Log Manager are part of the Core GET Platform. The Core GET Platform corresponds to the Data Platform. However, in SIMPLI-CITY the components – Service Registry and Monitoring belong to the Developer Console. Developer Console is a component which exists only in SIMPLI-CITY. Considering this inconsistency we decide to include the components – Service Registry and Monitoring in the Developer Console of the reference architecture. In Section 7.2.1, we explain how we deal with this inconsistency while projecting the GET Service architecture to the reference architecture.

6.3 External Sources

External Data Sources such as sensors and databases are responsible for absorbing information from the external environment and providing them to the Server Side. There is a range of External Data Sources and they are provided by external data suppliers. External Data Sources can provide information related to, for e.g. traffic conditions on a certain highway, whether a certain bridge has been opened after maintenance, how many vehicles are running within the speed limit on a certain road, etc. In SIMPLI-CITY, External Data Sources comprise of personal data sources, sensors, proprietary sensor database and static and historical data sources as shown in Figure 20 below. In GET Service, the Infra Platform acts as an external data source.

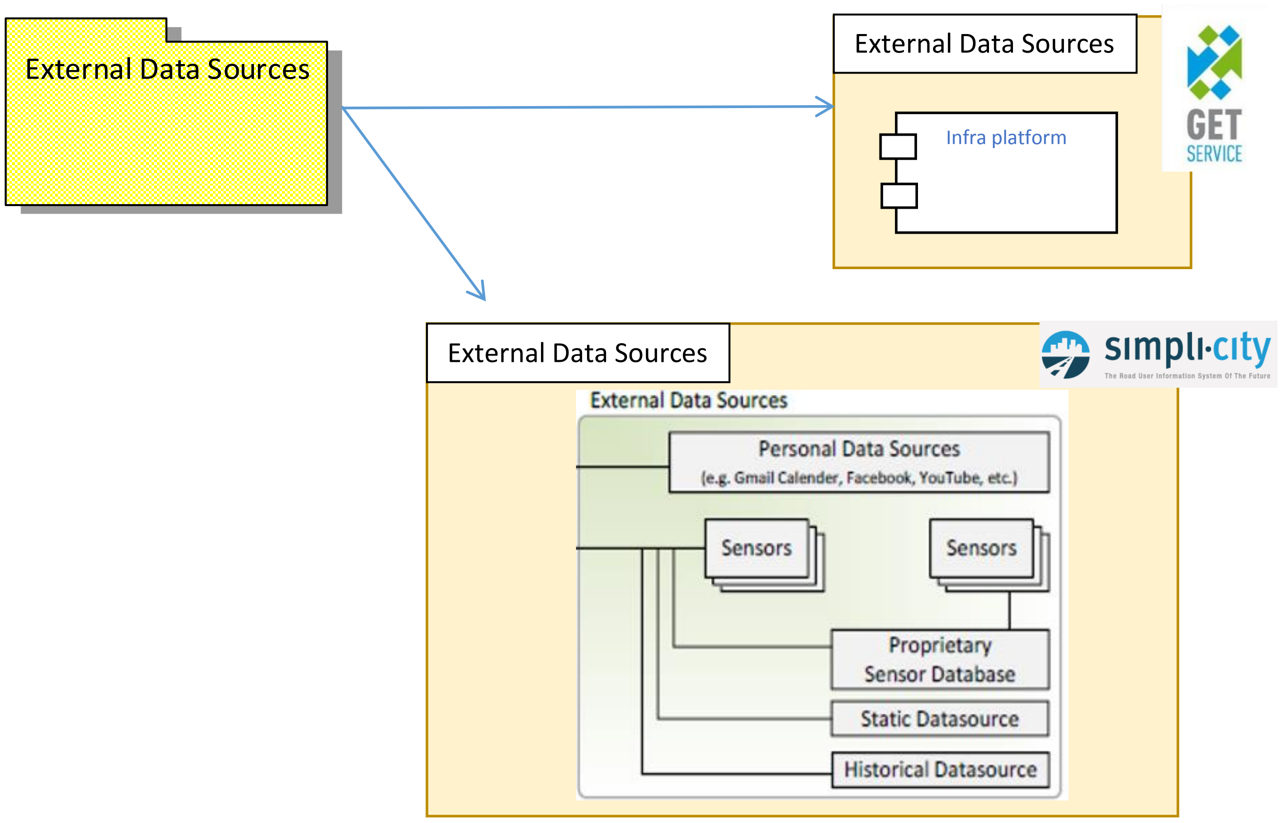


Figure 20: External Data Sources, aggregation level 0

External data sources can be classified into 2 types (Figure 21):

- **General Data Sources**: These can include sensors (such as traffic cameras, PIR sensors, etc.) which collect environmental data, databases and data sources containing static, dynamic and historical information.
- **Personal Data Sources**: These can include the Gmail calendar of the user and his accounts on social media sites such as YouTube, Facebook, Twitter and Tumblr or client specific data obtained from client/company database. The user/client has to authorize the system to access his personal accounts.

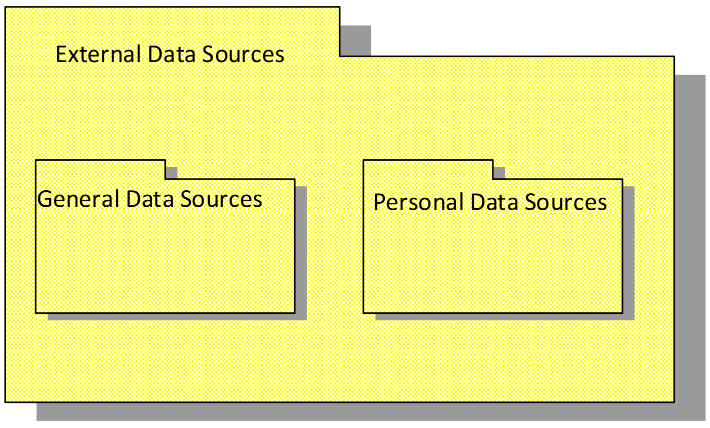


Figure 21: External Data Sources, aggregation level 1

6.4 Developer Support

Developer Support is a platform which is used by the system to provide help and support to the developers of services and application in the form of tools, guidelines and APIs. A step-by-step guide to service/app development tips on making the service/app compliant and ensuring a holistic feel for best possible user experience are some of the functions of Developer Support. Developer Support does not exist in GET Service. In SIMPLI-CITY, it exists in the form show below in Figure 22.

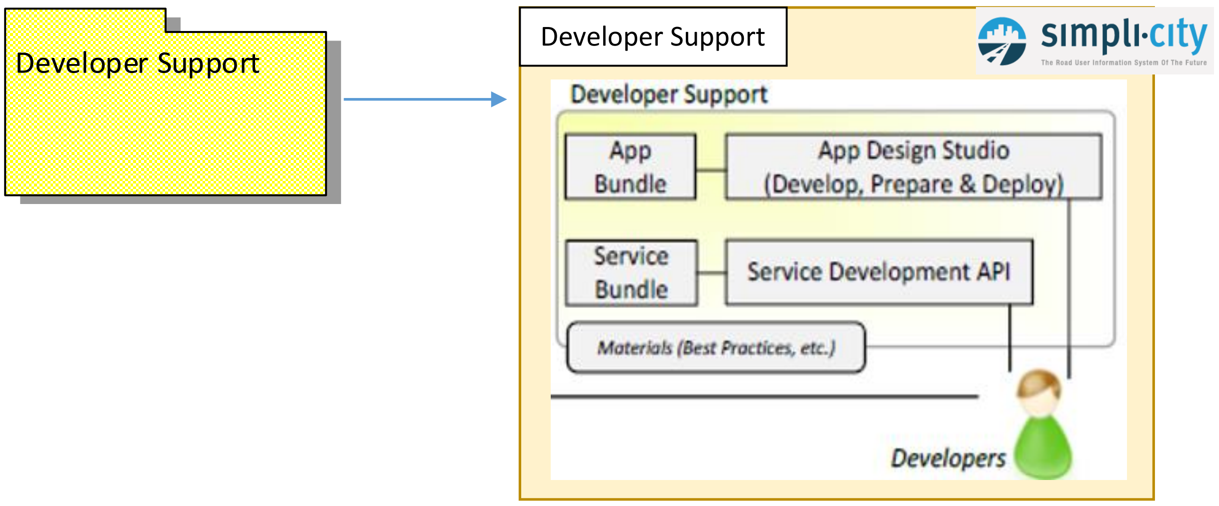

Developer Support
Developer Support
simpli-city
The Road User Information System Of The Future
Developer Support
App Bundle
App Design Studio (Develop, Prepare & Deploy)
Service Bundle
Service Development API
Materials (Best Practices, etc.)
Developers

Figure 22: Developer Support, aggregation level 0

7 Projections – Use Cases

In this chapter, we present projections of GET Service and SIMPLI-CITY services onto the developed reference architecture. In Section 7.1, we present the client side, in Section 7.2 we present the server side.

7.1 Client Side

In this section the projections of GET Service and SIMPLI-CITY services for the Client Side will be presented. As it was mentioned in Section 6.1 and finally visualized in Figure 16, the Client Side is composed of the Status Provider (Process Client and Information Source) and the Client Platform (Backend System and Client Platform). Sections 7.1.1 and 7.1.2 are two use cases of the reference architecture.

7.1.1 Client Side – Get Service

The reference architecture and the architecture of the GET service project for the client side, are in the same level of aggregation and abstraction so we will not go further in the analysis of that subsystem. In Figure 23 it is observed that the reference architecture can be applied unchanged for the GET service project. More specifically, the components Process Client and the Backend System have similar functionalities with those from the reference architecture (See Table 11). Furthermore, the Information Source represents the Event Source while the Client Interface the Planner, as it was presented in the Figure 1.

To go more in detail, the Event Source and Information Source share similar functionalities. The Event Source publishes events to the Core GET Service platform and provides on-board data. The Event Source translates information into events before publishing them. Furthermore, the Planner can be represented by the Client Interface.

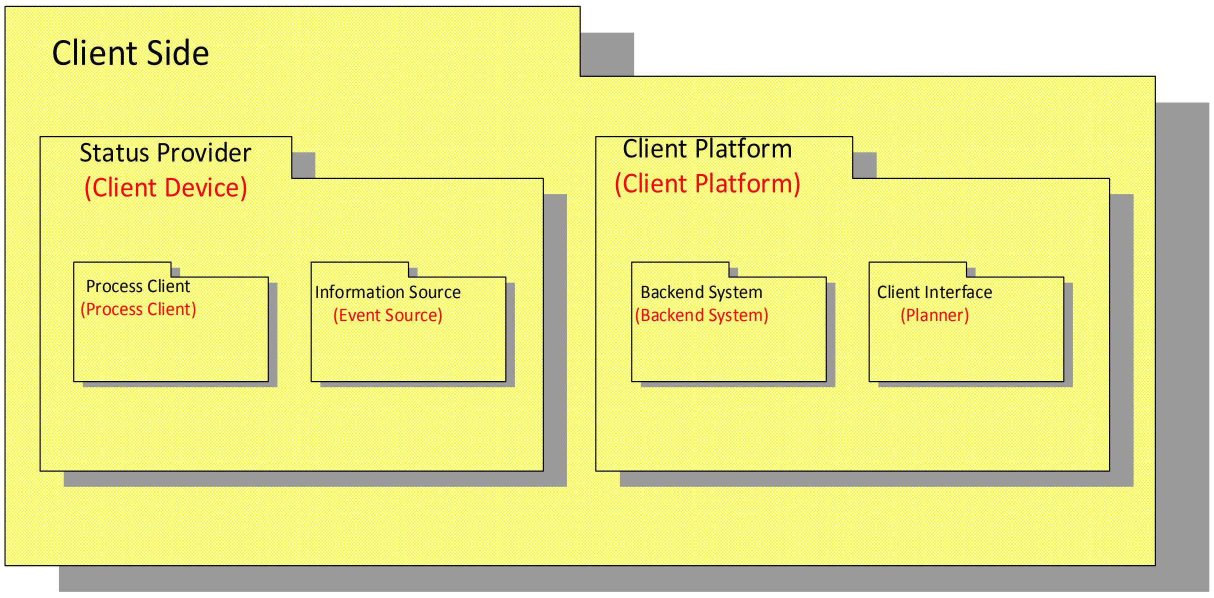


Figure 23: Client Side – GET Service projection

7.1.2 Client Side – SIMPLI-CITY

In this section, the reference architecture will be used to build and present the Client Side of SIMPLI-CITY service. The architecture of SIMPLI-CITY has more levels of aggregation, so starting from level 2 of the reference architecture we will build the SIMPLI-CITY Client Side step by step.

In Figure 24 it is observed that the reference architecture can be applied with some small changes for the SIMPLI-CITY service project. More specifically, the components Process Client and the Backend System are not presented in SIMPLI-CITY architecture though can be applicable (See Table 11). Furthermore, the Information Source represents the Car-Sensors while the Client Interface represents the Personal Mobility Assistant (PMA) and it will contain the SIMPLI-CITY App and the Navigation App as it will be discussed further.

More specifically, the Car-Sensors and the Information Source share similar functionalities. The Car-Sensors provide information from on-board devices to the Client Interface which are then transferred to the Server Side. Furthermore, the Personal Mobility Assistant (PMA) which includes the SIMPLI-CITY App and the Navigation App can be represented by the Client Interface. Like the Client Interface (see definition in Table 11: Reference Architecture - Major Client Side Components), the PMA will be used by the end users via their mobile device to access all SIMPLI-CITY functionality by making use of apps.

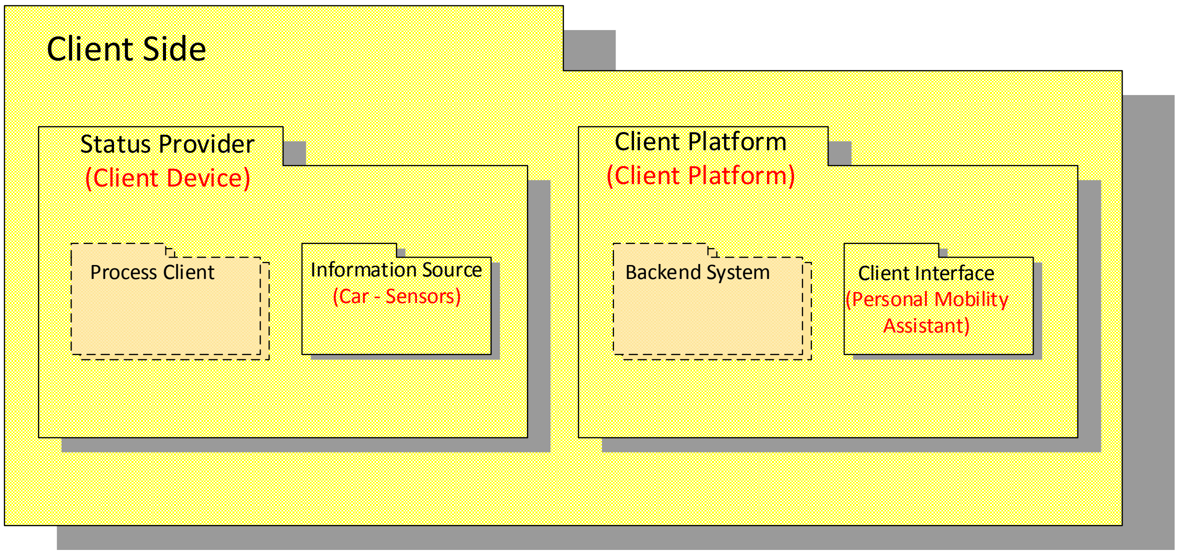


Figure 24: Client Side – SIMPLI-CITY projection

As it was mentioned before, the architecture of SIMPLI-CITY has more aggregation levels than the reference architecture so a further analysis of the use case has to be presented. In the next aggregation level (level 4) as it is presented in Figure 25 the Client Interface - Personal Mobility Assistant (PMA) is decomposed into two components, the SIMPLI-CITY App and the Navigation App. These two components are sharing the functionalities of the Client Interface as it was mentioned above and can be easily determined by looking at the descriptions in Table 7 and Table 11.

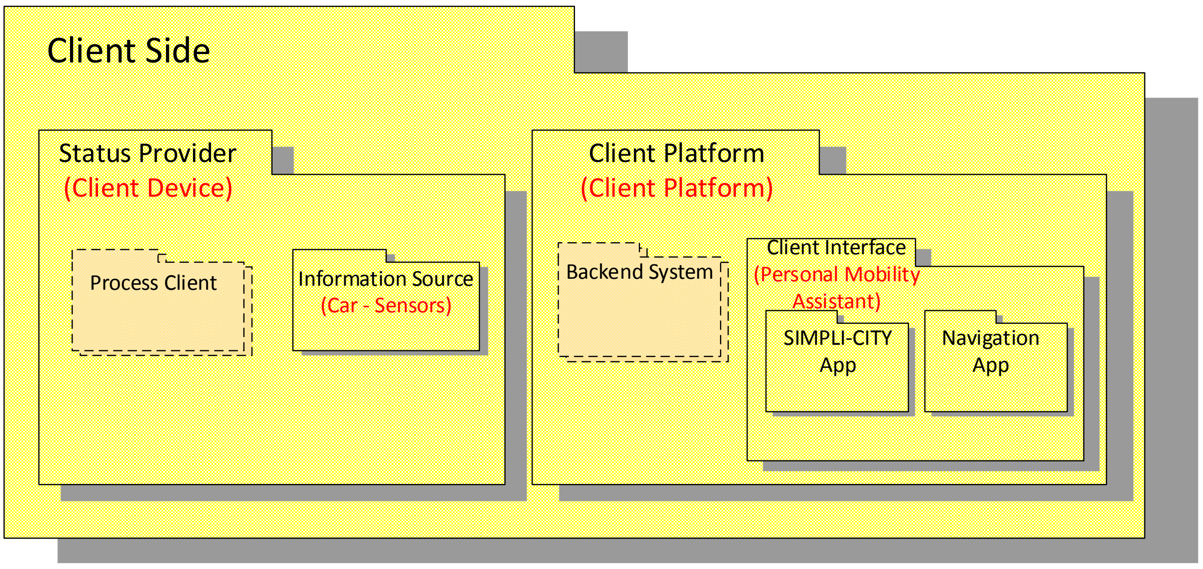


Figure 25: Client Side - SIMPLI-CITY projection - Aggregation level 4

Furthermore, the Client Platform and the SIMPLI-CITY app can be more decomposed into other components as it was presented in Figure 7. These components combined provide the functionality of the SIMPLI-CITY app, so the two next figures (Figure 26 and Figure 27) present a more technical/detailed perception of the Client Side architecture but do not change anything about the use of the Client Side from the reference architecture. As a result in Figure 26, aggregation level 5 is presented:

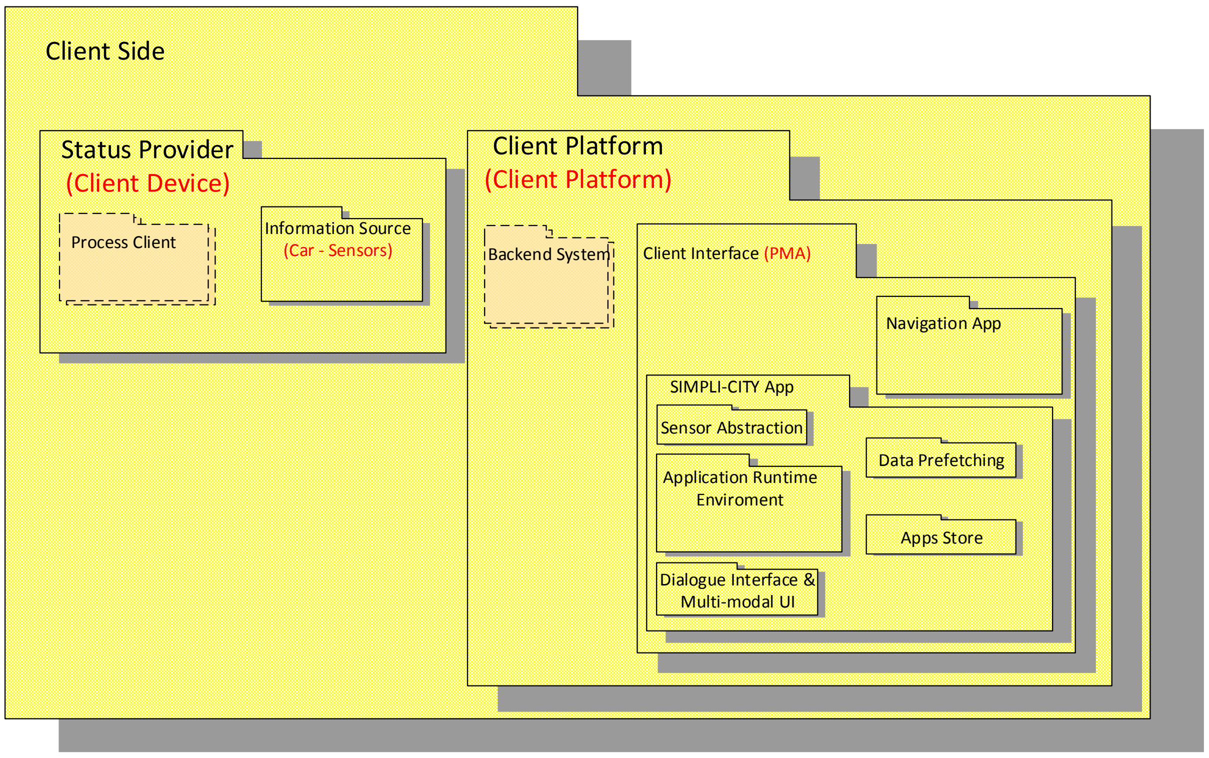


Figure 26: Client Side - SIMPLI-CITY projection - Aggregation level 5

In the following figure (Figure 27) aggregation level 6 of the Client Side component of the SIMPLI-CITY is presented. This level of aggregation is the last level that we can reach as it is the same level that it is provided by the latest SIMPLI-CITY project documents. In that figure all the components of the Vehicle & PMA component (See Section 4.2.1) are mentioned.

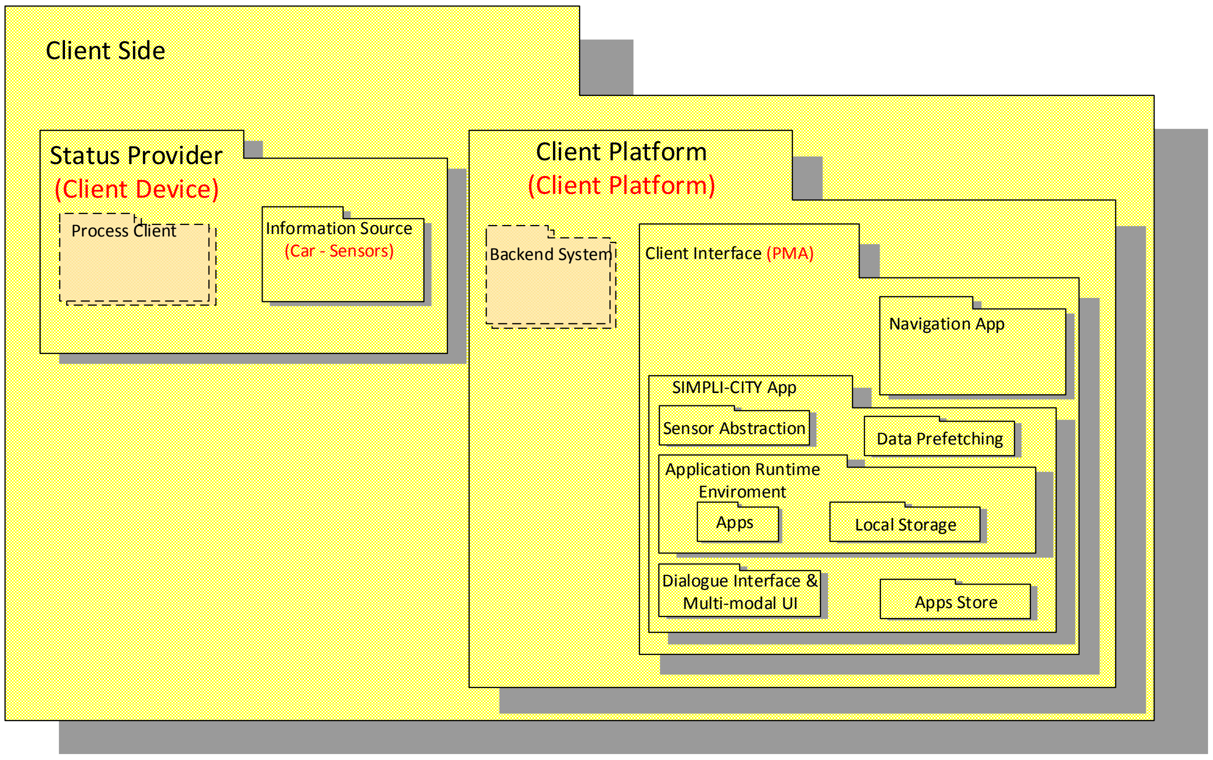


Figure 27: Client Side - SIMPLI-CITY projection - Aggregation level 6

7.2 Server Side

7.2.1 Server Side - GET Service

Projecting the GET Service architecture onto the reference architecture, two components of GET Service, the Extended GET Service Platform and the Core GET Service Platform can be considered to comprise the Server side. Both components play a fundamental role in service architecture and are equally responsible for handling service execution.

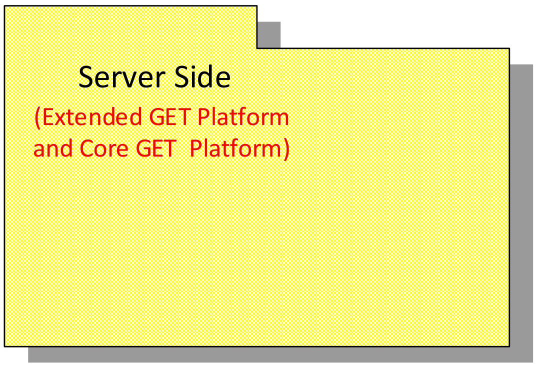


Figure 28: Server Side - GET Service projection

Moving to a lower level of projection, the Extended GET Service Platform can be seen as the Logic Platform while the Core GET Service Platform completely covers the Data Platform and partly covers the Developer Console as explained in Section 6.2. Graphical representation of it can be seen in Figure 29.

The definition of Extended GET Service Platform (Section 3.2.5) fully corresponds to the more general definition of Logic platform from the reference architecture (Section 6.2). This is because like the Logic Platform, the Extended GET Service Platform is responsible for final data processing, logic and execution of services for the end user.

As mentioned in Section 3.2.4, the Core GET Service platform correlates and aggregates external events from multiple Infra and Client platforms. It also contains an Information warehouse which stores static (schedules, master data) and dynamic (capacity) data. The aggregated events are published or can be retrieved by Client platforms or the Extended GET Service Platform. This definition complies with the definition of Data Platform whose main function is to collect data or events, process them and store them (Section 3.2.5). The Core GET Service Platform also includes some elements which belong to Developer Console of the reference architecture. These elements and the reasons why they belong to Developer Console are described later when we discuss a more aggregated level of architecture.

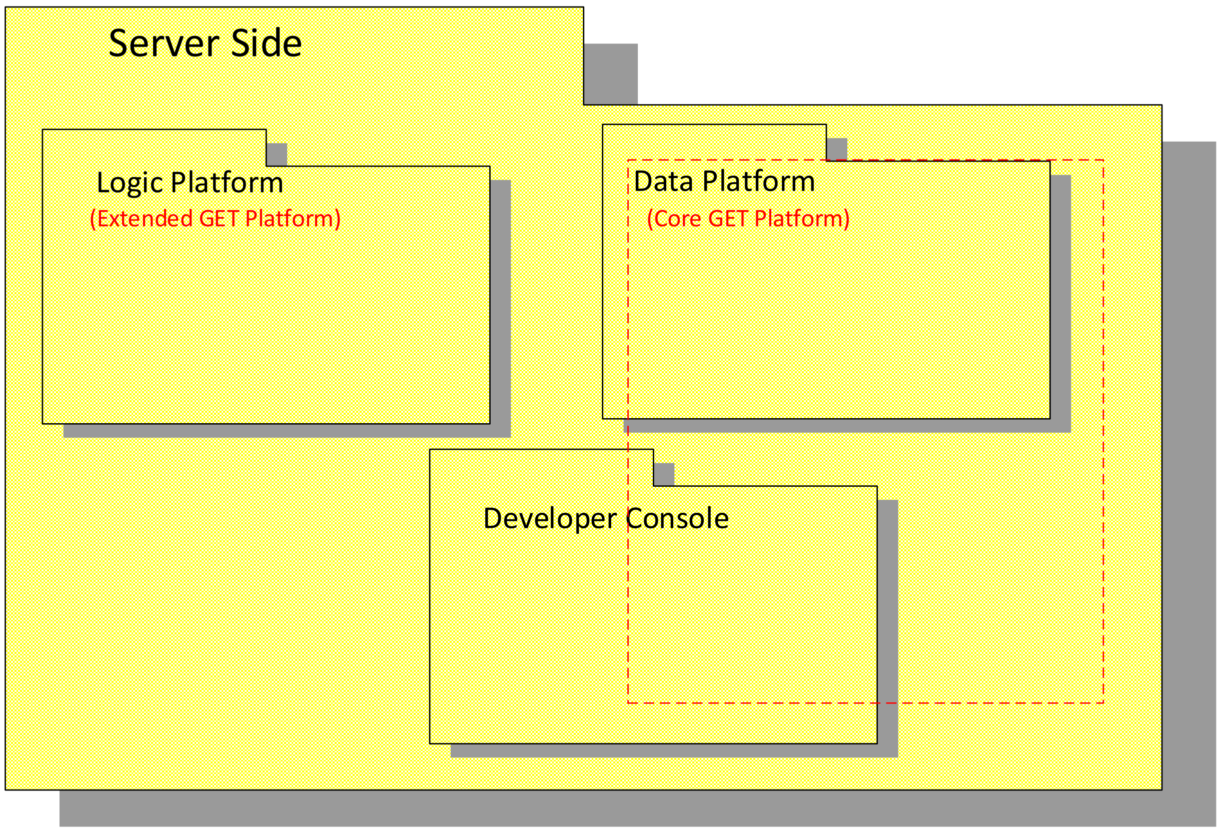


Figure 29: Server Side - GET Service projection - Aggregation level 2

As seen in Figure 30, three components of Extended GET Service Platform, which are the Orchestration Engine, the Proposer and the Process Store, correspond to Execution Engine, Services and Process Store of Logic Platform respectively.

Based on the definition of Orchestration Engine, it is used to orchestrate tasks for a running process. External events can influence the process path and result in a new/updated transport plan. Dynamic instance data is persisted for each running process by means of Process Store. After that the

Proposer proposes a (recalculated) transport plan to the Client platform/end user. Since the functions of the Orchestration Engine, Process Store and Proposer are contained by the Execution Engine, Dynamic Data Handling and Services of the Reference Architecture respectively, they can be considered to be particular cases of these components of the reference architecture.

Process Development Environment is an environment in which the user of GET Service can draw the process model. This process model then goes to the Orchestration Engine where it is read and executed. The Process Development Environment is not included in the reference architecture because it is very specific for GET Service. Thus, it only appears in the projection for GET Service.

Further we project the components of Core GET Service Platform to reference architecture. Subscription store, Event store and Information store can be seen as one component of reference architecture which is Data Storage. This is because the main function of Data Storage is to aggregate and store different types of data that can be classified and organized according to functional requirements and different criteria. The Event Channel corresponds to the component Data Channel which is more broadly defined. As it aims to receive and normalize events from Event sources through subscribe interfaces, the Event Channel represents itself as a particular case of Data Channel. Data Processing, whose main function is to filter data and/or events, correlate, merge, summarize and split them, will be covered by Event Correlator and Event Aggregator when considering Core GET Service Platform. Finally, Community Passport manager is a component which aims to provide service interfaces for registration and authentication to all GET Service components. It also responsible for a centralized passport store.

The difference in GET Service architecture and reference architecture occurs when we project Service Registry and Log Manager to reference architecture as discussed in Section 6.2. The main functions of these two components fall under the functions of Service Registry and Monitoring of Developer Console respectively. The decision to build the third block such as Developer Console was taken after analysing the architectures of both GET Service and SIMPLI-CITY services; it was decided to include it as an additional component that comprises specific functionalities as defined in Section 6.4.

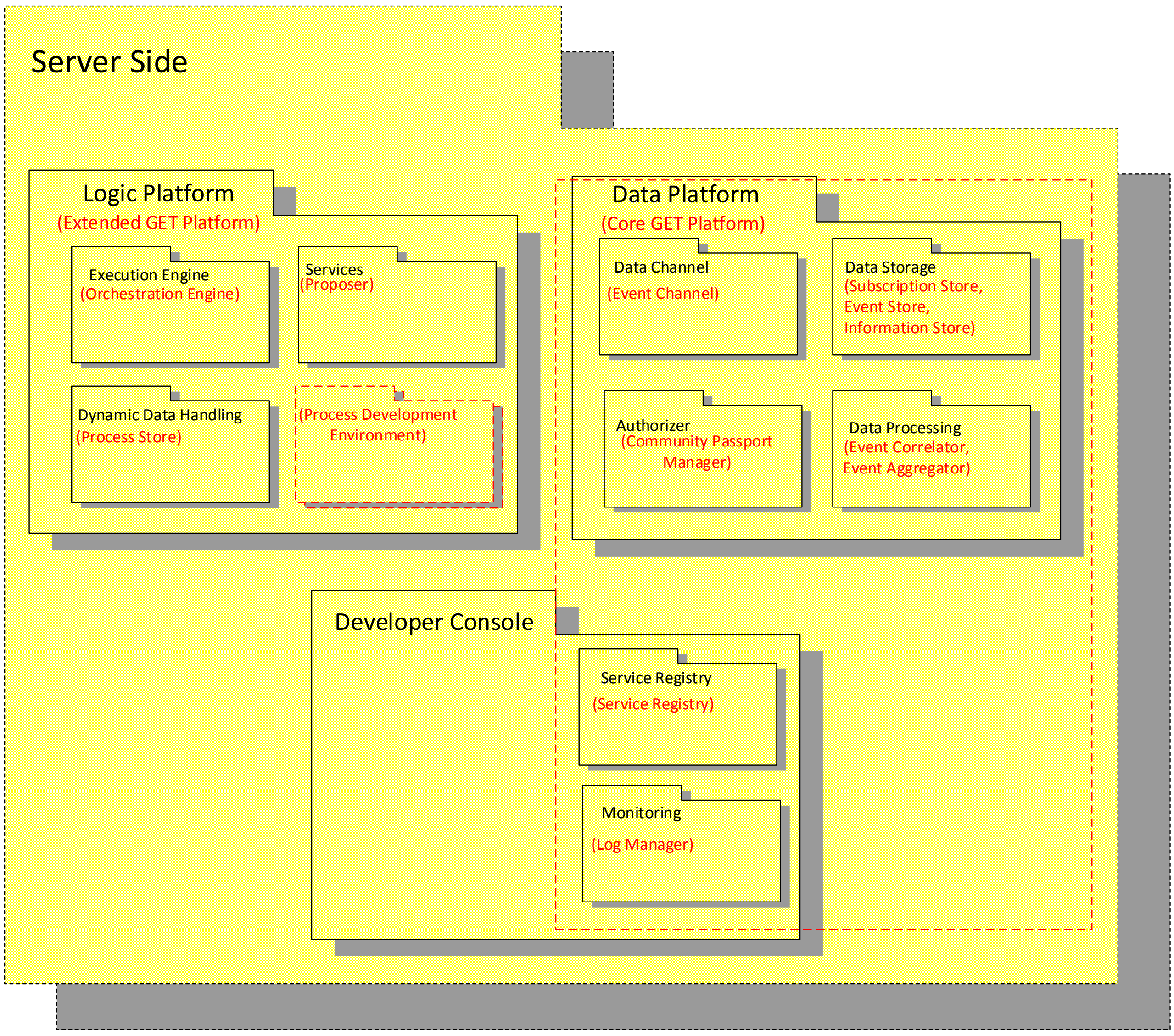


Figure 30: Server Side - GET Service projection - Aggregation level 3

7.2.2 Server Side - Simplicity

Considering SIMPLI-CITY architecture and projecting it to the reference architecture, SIMPLI-CITY Server Side fully corresponds to Server Side component of the reference architecture (Figure 31).

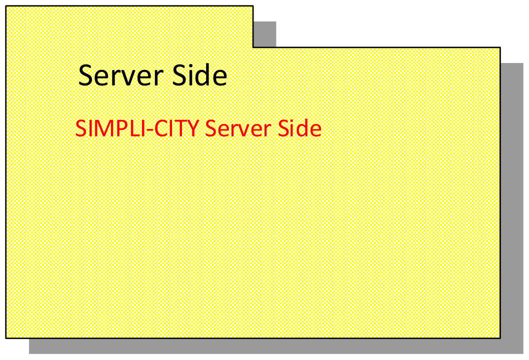


Figure 31: Server Side – SIMPLI-CITY projection

The lower level projection of the SIMPLI-CITY Server Side to reference architecture is depicted in Figure 32

The Service Runtime Environment is the fundamental server-side backend component in SIMPLI-CITY. It provides a deployment and execution framework for services, which offer the business logic for end user apps in SIMPLI-CITY, and provides additional features that are required in conjunction with the execution of services. Thus, it can be seen as the Logic Platform of the reference architecture.

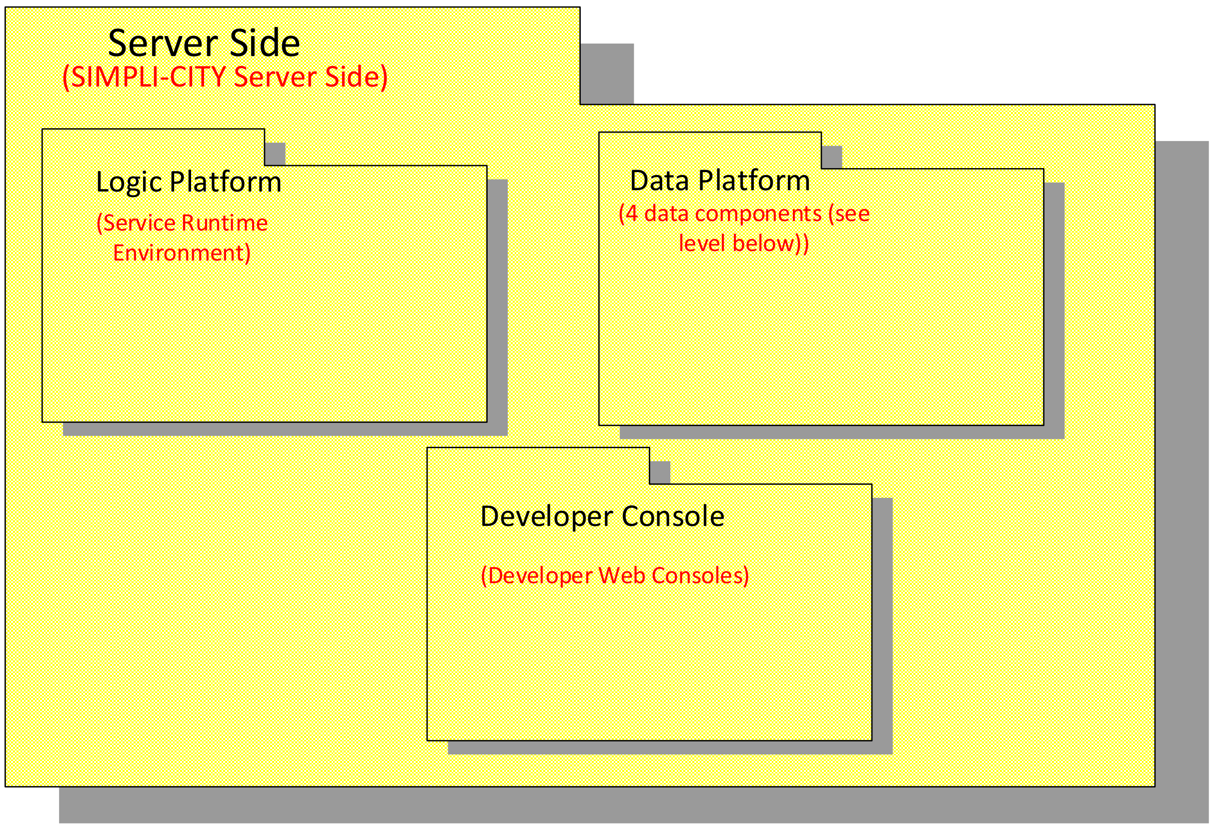

Server Side
(SIMPLI-CITY Server Side)
Logic Platform
(Service Runtime Environment)
Data Platform
(4 data components (see level below))
Developer Console
(Developer Web Consoles)

Figure 32: Server Side – SIMPLI-CITY projection - Aggregation level 2

The functions of the Developer Web Consoles in SIMPLI-CITY coincide with those of Developer Console in the reference architecture. This is because the Developer Web Consoles makes it easier for developers to access information from SIMPLI-CITY including the monitoring of services as well as submission of bundles to the marketplace.

There are a number of components in the SIMPLI-CITY Server Side that function together and coincide with the Data Platform component of the reference architecture. Each of these components will be described in the discussion of the next level of architecture which is depicted in Figure 33.

After analysing the structure of Service Runtime Environment we establish that Deployed Services corresponds to Execution Engine of the reference architecture. This is because like Execution Engine, the Deployed Services is responsible for logical grouping of services deployed in service runtime environment.

In the reference architecture, the Services component consists of services which are hosted and controlled by the server. Services can range in number and will have different functions. In SIMPLI-CITY, the Services component structures and bundles potential services that could deliver data from various sources to road user information systems. Services will provide backend logic which may be consumed by apps. Thus, the Services component in SIMPLI-CITY is similar to Services component in the reference architecture.

Two components of SIMPLY-CITY, Context-based Service Personalization and Data Prefetching Logic, can be considered similar to Dynamic Data Handling of the reference architecture. This is because the main function of the Dynamic Data Handling is to persist dynamic instance data. It also makes use of context specific data to come up with options or services which could prove helpful to the user. In a similar manner, Context-based Service Personalization makes use of context information in order to realize personalized services for a particular user. Based on the context, the Data Prefetching component will allow media prefetching and service prefetching. Service prefetching will invoke services which the user is likely to use in the next time.

The Data Platform of the reference architecture is decomposed into four components - Data Channel, Data Storage, Data Processing and Authorizer. User-Centric & Open Data Access and Sensor Abstraction & Interoperability Interfaces of SIMPLI-CITY can be mapped to Data Channel. Both components collect different types of data from the Client Side and/or External Data Sources and feed this data to the Data Processing component. Data Processing component is responsible for semantic description and processing in the form of filtering, correlation, merging, fusion, summarization and splitting.

The cloud based information of SIMPLI-CITY can be compared to Data Storage of reference architecture. Like Data Storage, the cloud based infrastructure will act as a service which is dedicated into managing different types of data in a persistent, scalable and efficient storage. The cloud based storage will be fed with information from apps (e.g. to store data from users) or from external data sources such as sensors or user centric data.

Authorizer is not presented as an independent component in SIMPLI-CITY architecture. However, there is an Authorization Service which has similar functions as that of Authorizer.

In SIMPLI-CITY, there is a Developer Web Console which consists of 4 components - Service Registry, Monitoring, Apps Store and Service Marketplace. The Service Registry and Monitoring can be mapped to Service Registry and Monitoring components of reference architecture respectively while Apps Store and Service Marketplace are specific for SIMPLI-CITY. Thus, Apps Store and Service Marketplace are not in reference architecture but only appear in the projection.

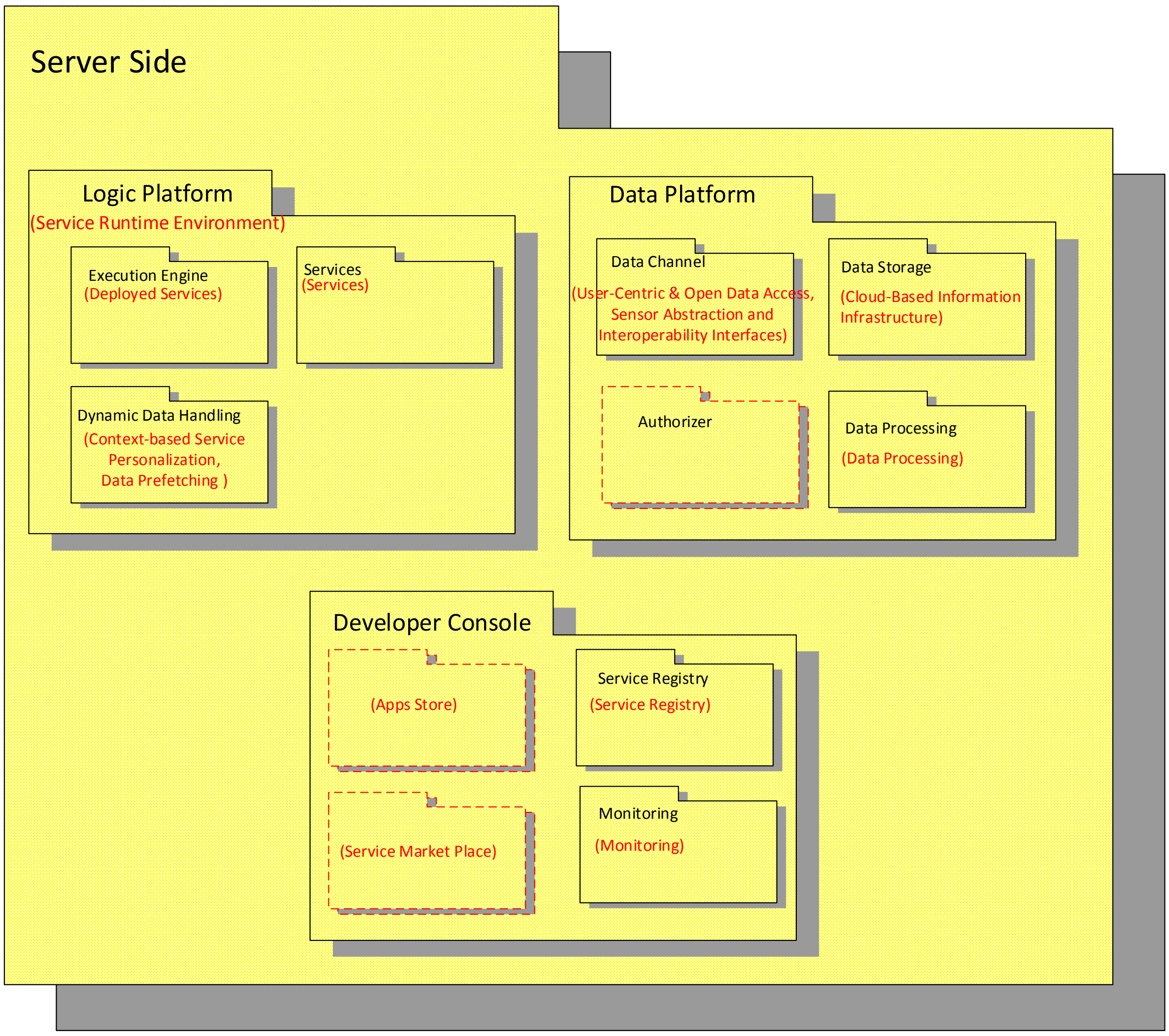


Figure 33: Server Side – SIMPLI-CITY projection - Aggregation level 3

7.3 External Sources

7.3.1 External Sources – GET Service

The Infra Platform serves as the External Data Sources for GET Service. This is because the Infra Platform distributes information on traffic, tides, bridges and locks. The Core GET Service platform can subscribe to events that contain infrastructure status information.

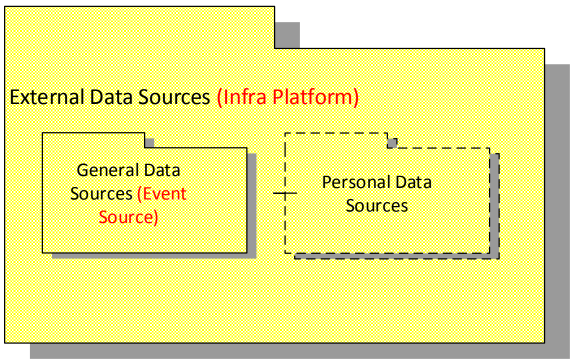

External Data Sources (Infra Platform)
General Data Sources (Event Source)
Personal Data Sources

Figure 34: External Sources – GET Service projection

The Infra Platform contains the Event Source. As mentioned in Section 3.2.3, the Event Source translates infrastructure data (static/dynamic) into events and publishes the events to the Core GET Service platform. It can thus be considered to be similar to General Data Sources. There are no Personal Data Sources in GET Service.

7.3.2 External Sources - Simplicity

In Simplicity there is a component known as External Data Sources. This component consists of Personal Data Sources, sensors, proprietary sensor database, static data source and historical data source. The Gmail calendar of the user and his accounts on social media like Facebook and YouTube are Personal Data Sources. Sensors, proprietary sensor database, static data source and historical data source are General Data Sources as shown in Figure 35.

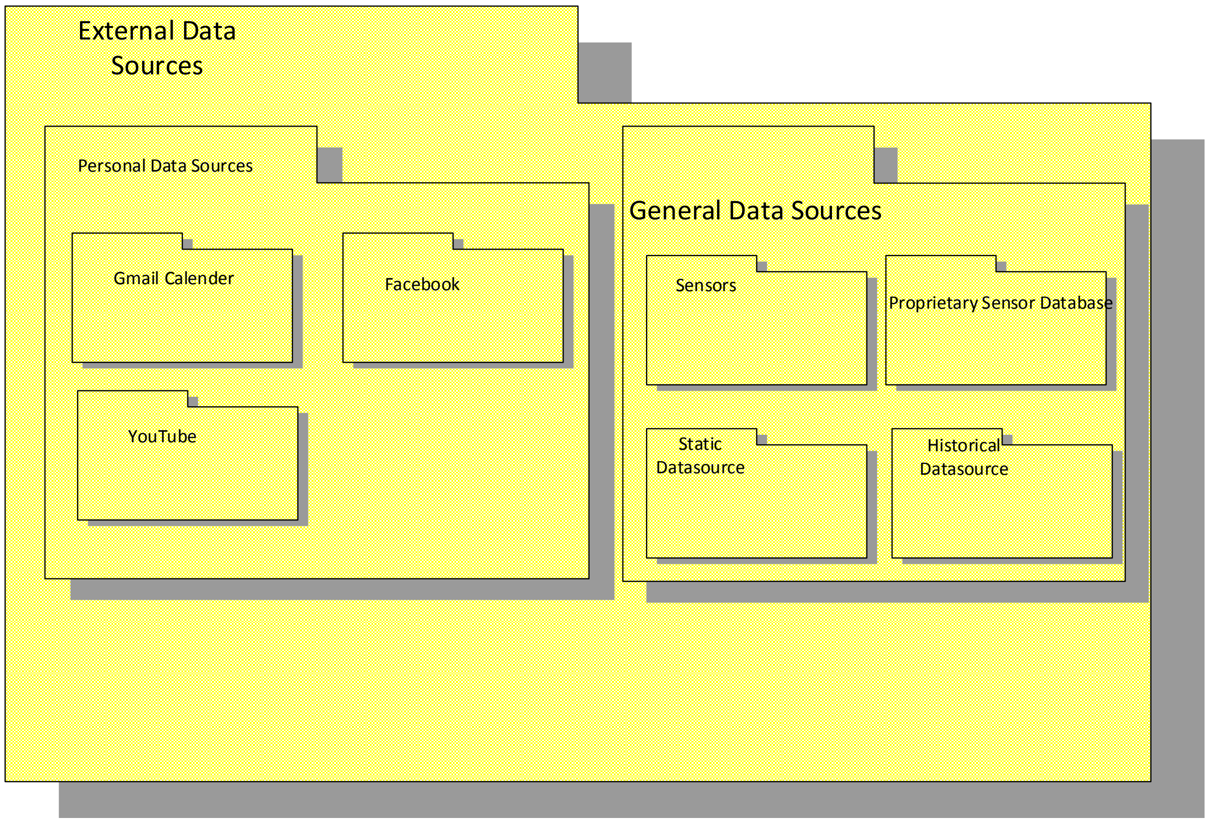


Figure 35: External Sources – SIMPLI-CITY projection

7.4 Developer Support

7.4.1 Developer Support - GET Service

There is no such component in GET Service so there is no projection for GET Service.

7.4.2 Developer Support - Simplicity

In Simplicity there is a component known as Developer Support. Developer Support consists of App Bundle, Service Bundle, App Design Studio, Service Development API and Materials (like Best Practices). The App Bundle is linked to the App Design Studio and the Service Bundle is linked to the Service Development API.

The Application Design Studio offers an appropriate step-by-step procedure to app development, assisting the app developer during the entire development process. It delivers everything that a developer needs to prepare an app for the usage within SIMPLI-CITY.

The Service Development API is a component aimed at third party developers that allows them to create and configure their own services on the SIMPLI-CITY platform. These services will be later used within end-user applications and will be responsible for the provision of the external data needed during their execution.

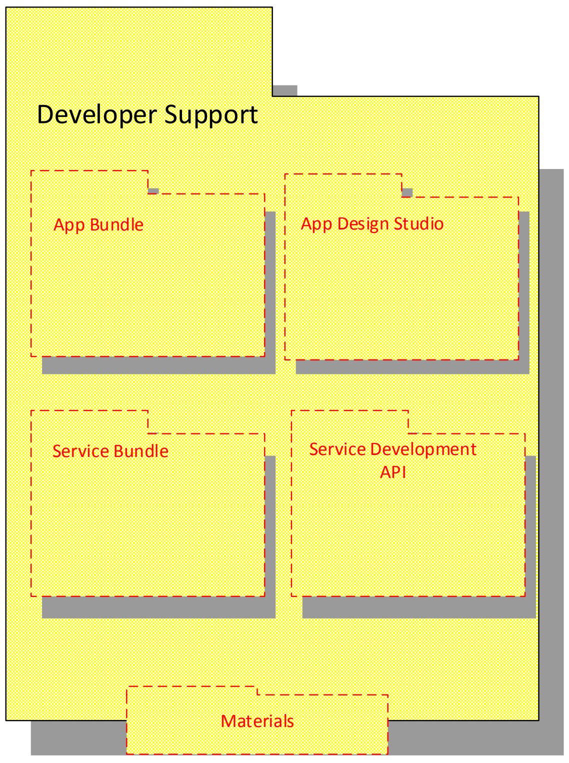


Figure 36: Developer Support – SIMPLI-CITY projection

8 Related work

There are several related efforts in which a reference architecture for Intelligent Transportation Systems (ITS) is developed. Most of these efforts are initiated by industry. In the discussion, we adopt the following terminology about reference architectures (Angelov, Grefen, & Greefhorst, 2012). A classical reference architecture can be implemented using current, tested technologies while a preliminary reference architecture depends on technologies that have been partially developed but need further development to become useful in practical systems. Next, the goal of a reference architecture can be standardization among multiple organizations, aiming at system interoperability, or facilitation of the design of concrete architectures, aiming to provide guidelines for the design of concrete systems.

In the Netherlands, Dutch Integrated Testsite Cooperative Mobility (DITCM) has developed a reference architecture for ITS applications in the Netherlands (Sambeek, et al., 2015). The DITCM reference architecture is based on existing ITS research projects in the Netherlands. It is descriptive, preliminary - for instance technologies for realizing cooperative driving are still under development - and supports both standardization and facilitation. The standardization goal is important to enable interoperability between ITS solutions developed in the Netherlands. The facilitation goal is important to support individual system development projects in their design efforts. The reference architecture covers both roadside and traffic management systems, intelligent vehicles, as well as end user applications. The DITCM reference architecture consists of a system architecture and a description of the business aspects of an ecosystem with stakeholders from public and private parties in the Netherlands.

The European Telecommunication Standards Institute (ETSI) has standardized an architecture for ITS stations (ETSI, 2010). An ITS station is a collaborative functional component in an ITS architecture. The ETSI reference architecture consists of applications, facilities, communication, interfaces, security and management entities. Not all parts have to be present in an ITS Station. Examples of ITS stations are Vehicle ITS, Roadside ITS, Central ITS and Personal ITS. The ETSI ITS architecture is a classical standardization architecture.

The US Department of Transportation leads a project to develop a standardization reference architecture for connected vehicles and their environment, also taking into account roadside infrastructure (ITERIS, 2015). The architecture considers multiple viewpoints: enterprise, functional, physical and communications. The aim of the project is to define a classical reference architecture, though some technologies are still preliminary.

From the academic side, a recent initiative discusses a preliminary reference architecture for an ICT-based intelligent infrastructure based on a collaborative network of ITS stakeholder (Osório, Afsarmanesh, & Camarinha-Matos, 2010). The reference architecture has two common layers, shared among the collaborating partners. The first layer is a collaboration-oriented road mobility infrastructure layer, which supports a network of stakeholders that collaborate to offer services for users. Next, an intelligent road mobility infrastructure layer supports communication between vehicles and the first layer. Both layers map to the server layer of the reference architecture proposed in this report. The architecture supports facilitation.

9 Conclusion

The goal of this work has been to design a reference architecture for mobility-related services (both freight and travellers) by analysing the architectures developed in two European projects - GET Service and SIMPLI-CITY. The reference architecture can be used as a blueprint that facilitates the development of new mobility-related services in the future. After an in-depth analysis of the GET Service and SIMPLI-CITY architectures, the similarities and differences between the architectures of the two projects were considered. Keeping these in mind we propose a design for a reference architecture (an overview is presented in Appendix A) which is flexible and clearly defined.

The proposed reference architecture has been further evaluated by investigating its usage in the context of the re-development of two architectures. The two architectures used to evaluate the reference architecture were the architecture of GET Service and SIMPLI-CITY which can be seen as use cases/projections. Using the reference architecture as a pattern, we have re-built the existing architectures of GET Service and SIMPLI-CITY. This shows the usability of the reference architecture and shows that it can be beneficial for developers who want to design architectures for mobility-related services.

The proposed reference architecture is approached in a mostly intuitive way but with a clearly structured background. We therefore believe that this work will contribute to the better understanding of the domain of reference architectures and to the design of more successful architectures for mobility-related services.

Aknowledgments

We thank Stefan Schulte of Vienna University of Technology and Paul Saraber of Portbase B.V., Rotterdam for their support during the execution of the project leading to this report.

10 Bibliography

Abels, D. S. (2013). *WP3 - Architecture, Functional & Technical Specification, Security & Privacy Concet, Integration.* Simpli-City.

Angelov, S., Grefen, P., & Greefhorst, D. (2012). A Framework for Analysis and Design of Software Reference Architectures. *Information and Software Technology, 54*(4), 417-431.

Arne-Jorgen Berre, et al. (March 24, 2006). *Component and Model-based development Methodology.* SINTEF ICT.

ETSI. (2010). *Intelligent Transport Systems (ITS): Communications Architecture.* ETSI.

Grefen, P. (Spring 2014). *Business Information System Architecture.* Eindhoven.

Grefen, P., & Vries, R. R. (1998). A reference architecture for workflow management systems. *Data & Knowledge Engineering*, 31-57.

ITERIS. (2015). *Connected Vehicle Reference Implementation Architecture*. Retrieved from Connected Vehicle Reference Implementation Architecture: http://www.iteris.com/cvria/

Kruchten, P. (2000). *The rational Unified Process: An Introduction, 2nd Edition.* Addison-Wesley.

Kruchter, P. (1995). Architectural Blueprints - The "4+1" View Model of Software Achitecture. *IEEE Software 12 (6)*, 42-50.

Osório, A., Afsarmanesh, H., & Camarinha-Matos, L. (2010). Towards a Reference Architecture for a Collaborative Intelligent Transport System Infrastructure. *PRO-VE* (pp. 469-477). Springer.

Reed, P. (2002, September 15). *Reference Architecture: The best of best practices*. Retrieved from developerWorks: http://www.ibm.com/developerworks/rational/library/2774.html

Sambeek, M., Ophelders, F., Bijlsma, T., Kluit, B. v., Turetken, O., Eshuis, R., . . . Grefen, P. (2015). *Towards an Architecture for Cooperative ITS Applications in the Netherlands.*

Saraber, P. (November 22, 2013). *Standardized Interface Definitions (Deliverable 2.2.2).* GET Service.

Treitl, Stefan, et al. (May 31, 2013). *Requirements analysis (Deliverable D1.2).* GET service.

Velde, M. v., Saraber, P., Grefen, P., & Ernst, A. C. (2013). *GET Architecure Definition (Deliverale D2.2.1).* GET Service.

Wikipedia. (2014, June 9). Reference architecture. Retrieved from http://en.wikipedia.org/wiki/Reference_architecture#cite_note-3

11 Appendices

Appendix A

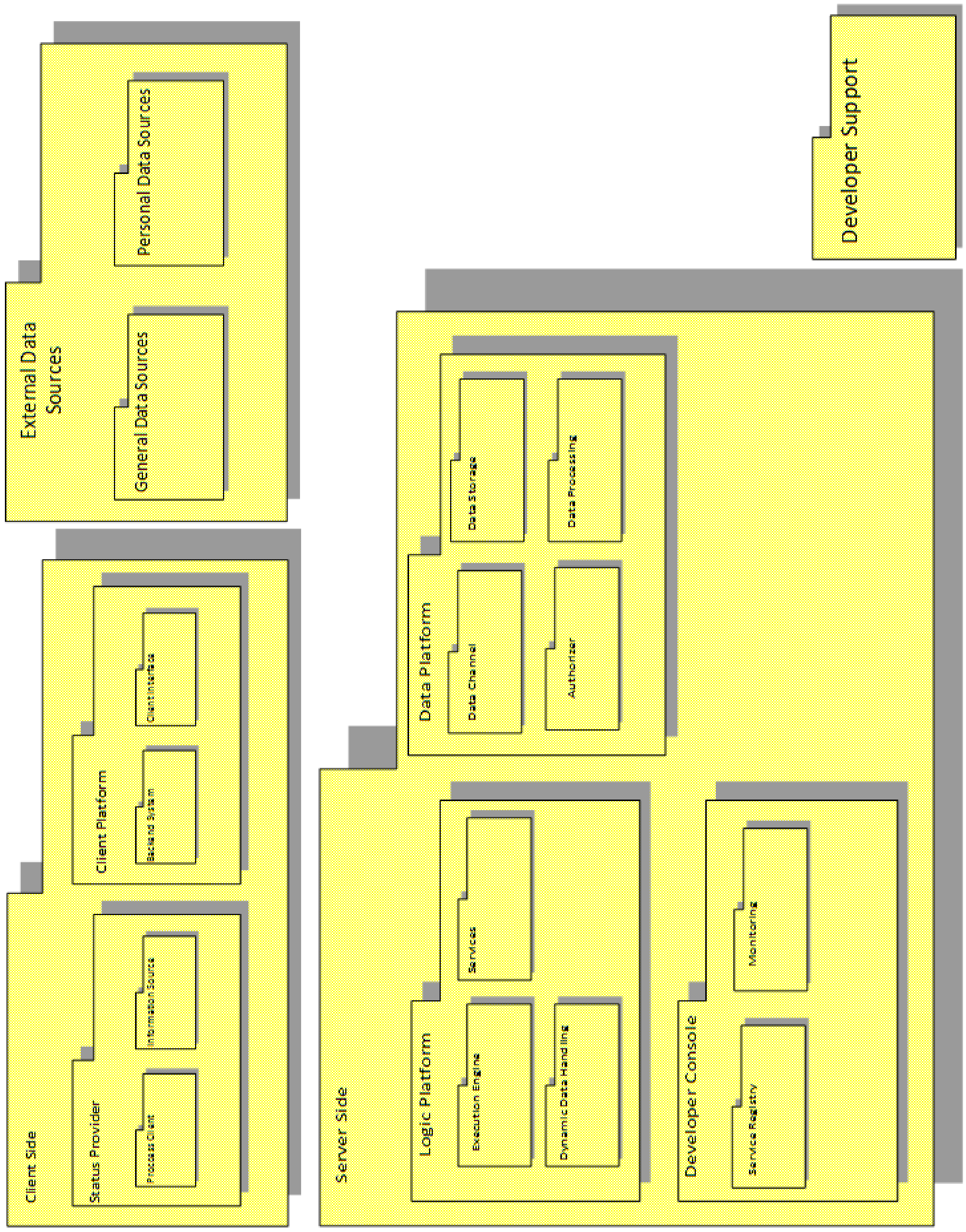


Figure 37: Reference Architecture

Appendix B

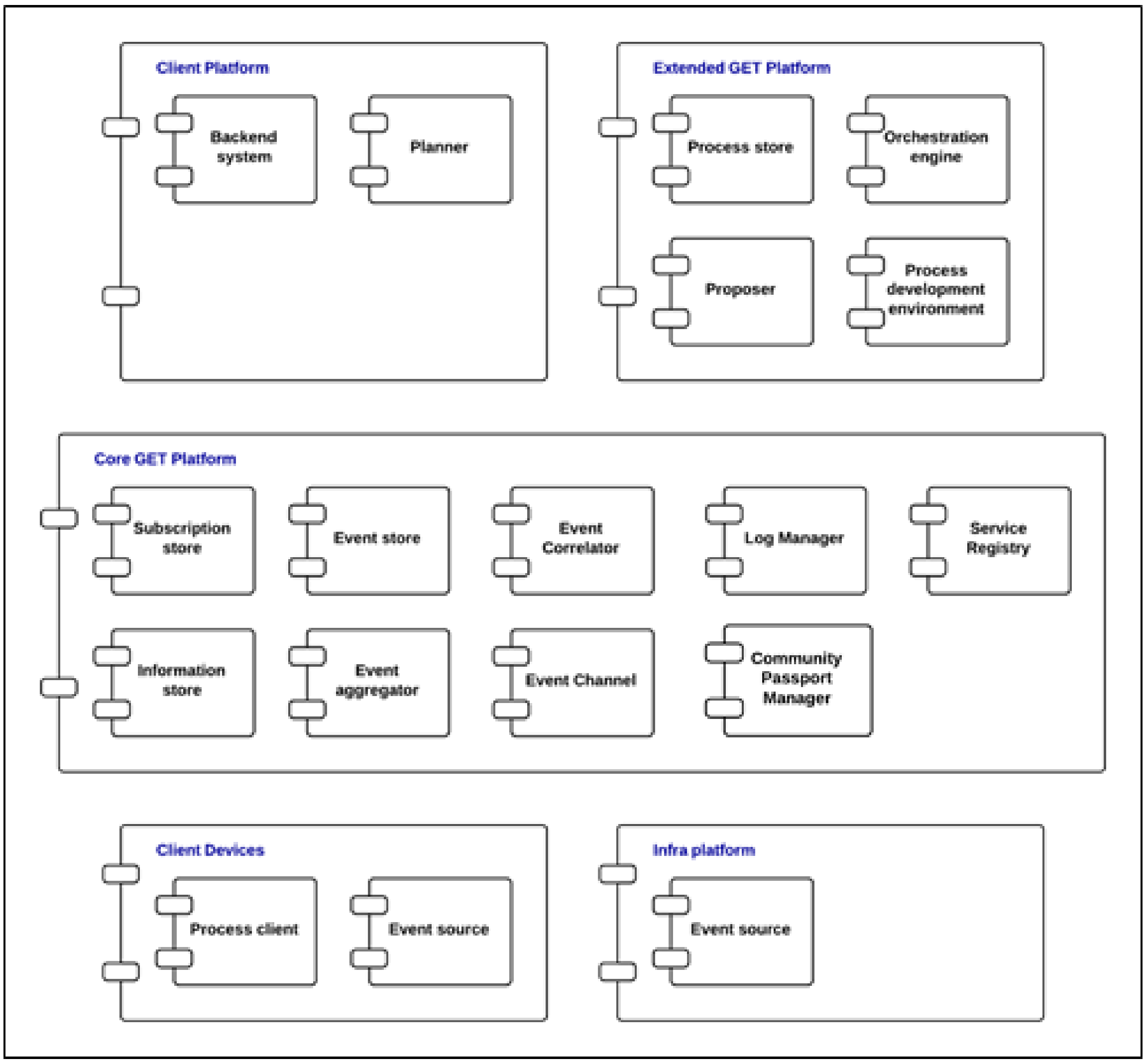


Figure 38: Subsystem Components of the GET Service architecture

Appendix C

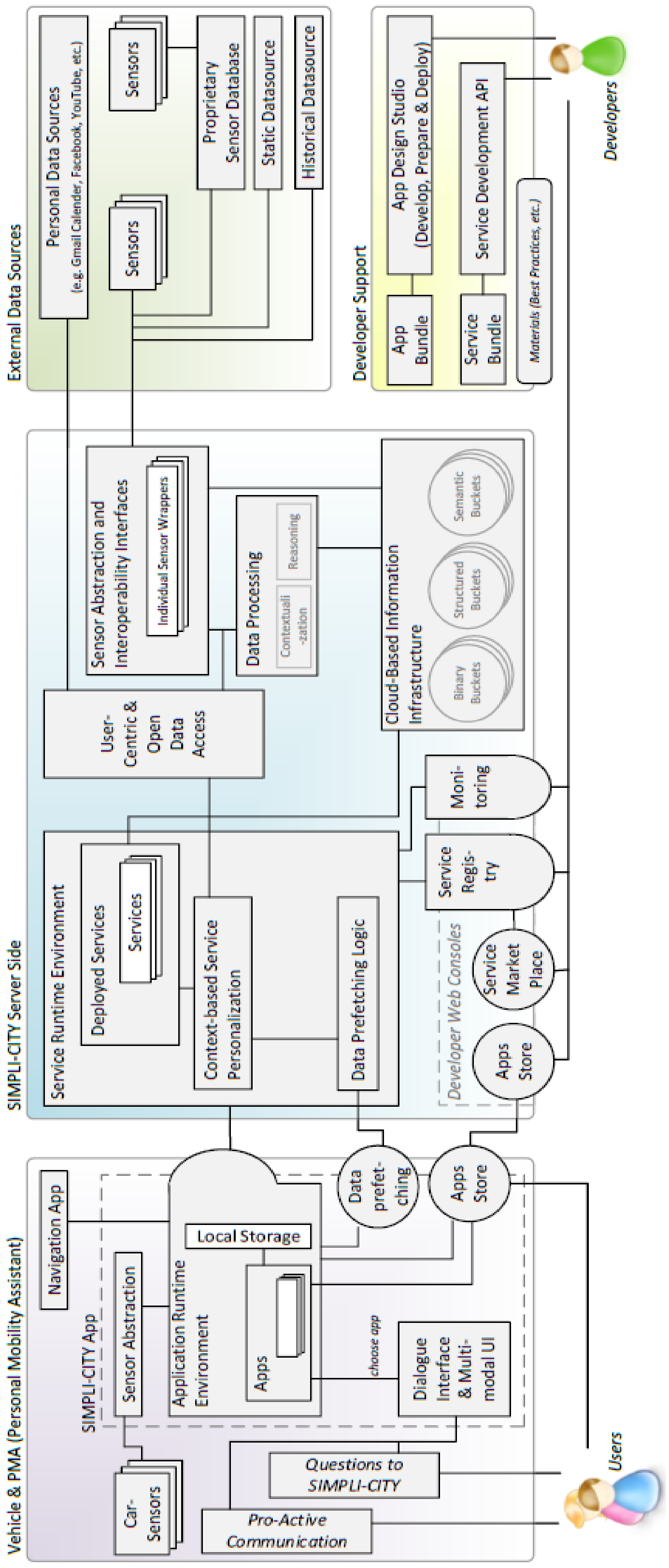


Figure 39: Subsystem Components of the SIMPLI-CITY architecture