Mobilytics- An Extensible, Modular and Resilient Mobility Platform

Chinmaya Samal
Vanderbilt University
Nashville, TN, USA
chinmaya.samal.1@vanderbilt.edu

Abhishek Dubey
Vanderbilt University
Nashville, TN, USA
abhishek.dubey@vanderbilt.edu

Lillian J. Ratliff
University of Washington
Seattle, WA USA
ratliffl@uw.edu

***Abstract*—Transportation management platforms provide communities the ability to integrate the available mobility options and localized transportation demand management policies. A central component of a transportation management platform is the mobility planning application. Given the societal relevance of these platforms, it is necessary to ensure that they operate resiliently. Modularity and extensibility are also critical properties that are required for manageability. Modularity allows to isolate faults easily. Extensibility enables update of policies and integration of new mobility modes or new routing algorithms. However, state of the art mobility planning applications like open trip planner, are monolithic applications, which makes it difficult to scale and modify them dynamically. This paper describes a microservices based modular multi-modal mobility platform Mobilytics, that integrates mobility providers, commuters, and community stakeholders. We describe our requirements, architecture, and discuss the resilience challenges, and how our platform functions properly in presence of failure. Conceivably, the patterns and principles manifested in our system can serve as guidelines for current and future practitioners in this field.**

***Index Terms*—Urban Mobility; Microservices; Reliability; Resilient platform; Extensible Platform**

I. Introduction

Emerging trends and challenges. With increasing urban population, moving people from one place to another is becoming an increasingly complex challenge. According to U.S. Census Bureau 2013 survey reports [1], [2], about 86 percent of all commuters commuted to work by personal vehicles, either driving alone or carpooling. The fact that the majority of personal vehicles used by commuters are single occupancy only serves to exacerbate the issue [3]. metropolitan cities are increasing their investment in public transit to provide better mobility options to the residents of the city. However, public transit networks have its limitations because they use existing infrastructure to provide build public transit networks.

Technological innovations have enabled shared mobility options which are increasingly being used by commuters often in lieu of a personal vehicle. To support the demand for such options, companies such as Uber and Lyft are increasingly investing in making shared mobility services readily available to the user on-demand in urban environments. While such services may be deemed more convenient than riding public transit due to their on-demand nature, reports show that they do not necessarily decrease the congestion in major cities [4]. So, there is a need for shared mobility mechanisms in the city, where commuters can judiciously mix multiple modes of transportation for their commute. This problem of route planning involving different modes of transportation is called multi-modal route planning [5].

To fix these problems, many cities in the United States have started implementing Transportation Demand Management programs (TDM) whose goal is to understand how commuters make transportation decisions and to increase the efficiency of the transportation system by providing innovative technology-based services. A mobility platform not only provides routing services to people but also gives access to holistic information which can be utilized by the city officials to understand the mobility patterns of people in the city to improve services and researchers to study and develop efficient algorithms for transportation. Given the societal relevance of this platform, it is necessary to ensure that they operate resiliently. Modularity and extensibility are also critical properties that are required for manageability. Modularity allows to isolate faults easily. Extensibility enables update of policies and integration of new mobility modes or new routing algorithms. However, state of the art mobility planning applications like open trip planner, are monolithic applications, which makes it difficult to scale and modify them dynamically.

Contributions. In this paper, we propose *Mobilytics*- a platform that integrates mobility providers, commuters, and community stakeholders. The novelty of Mobilytics Platform lies in three aspects: (1) a *modular* microservices-based architecture that integrates services provided by multiple stakeholders through loose-coupling of interfaces, (2) an *extensible* platform that can add new data sources and build services on top of it without affecting the entire system (3) a *resilient* platform that has a set of decentralized coordinators for orchestrating services and managing the entire system without failing.

Paper outline. Section II gives an overview of Mobilytics platform. Section III, Section IV, Section V describes how our platform enables modularity, extensibility, resiliency, respectively. Section VI gives an overview of the current literature on mobility platforms. We discuss our approach, future work and present concluding remarks in Section VII

978-1-5386-4705-9/18/$31.00 ©2018 IEEE
DOI 10.1109/SMARTCOMP.2018.00029

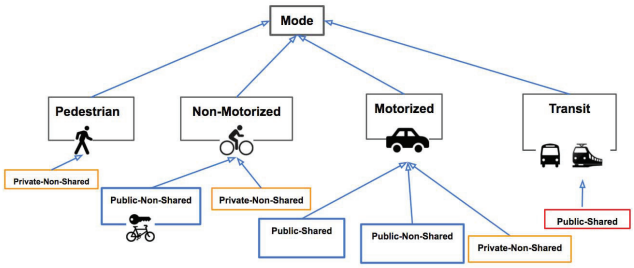

Fig. 1: This figure shows different abstract mode-types that are categorized as per the relation of a user with the mode.

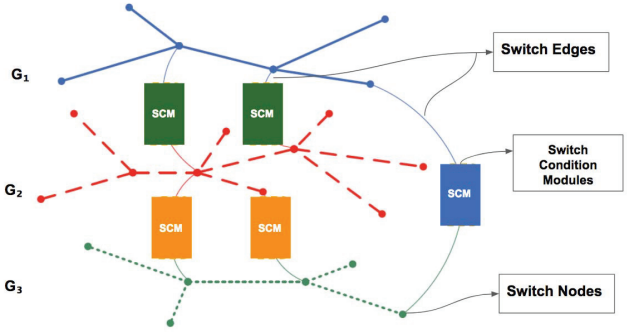

Fig. 2: This figure shows an abstract view of multi-modal graph which can be realized as combination of uni-modal graphs where user can switch from one mode to another using Switch Nodes and edges connecting such switch nodes are called Switch edge.

II. Mobilytics Platform

In this section, we will provide a brief description of *Mobilytics*- a holistic platform that integrates mobility providers, commuters, and community stakeholders. Some key logical layers of our platform are:

A. Database Layers

1) **Geospatial Layer:** This layer contains geospatial information, which contains relationships between the points, lines, and polygons that represent the features of a geographic region. Each geometric feature consists of a unique geographical identification code, topological information such as coordinates and some optional properties of the feature such as buildings, roads, monuments, etc. In case of roads, the properties also contain information on type of road (residential, motorway, highway, transit), number of lanes on a road etc. This layer can be built from different types of Geographic Information System (GIS) file formats such as OpenStreetMap (OSM), GeoJson, Shapefile etc. Some optional user-defined properties can be added too, such as real-time congestion, flow information etc.
2) **Abstract Modal Layer:** This layer abstracts mode-specific information from the *Geospatial Layer* and hence provides a clear separation of static topological data present in *Geospatial Layer*. To make our platform modular and extensible we need different layers of our platform to be mode-independent so that addition and deletion of a mode don't affect other services. So, we need to find common properties between different modes and define a mode-type such that each mode having same properties belong to the same mode-type. Figure 1 shows different modes and the categories they belong to. A mode can be pedestrian, non-motorized, motorized or transit. Mobility services can be categorized as per the relation of a user with the mode. They are **Ownership** and **Control**. Ownership can be *Public* if the mode is not owned by user otherwise it's *Private*. Control can be *shared* if the user is not in control of the mode else it's *Non-shared*. So, based on these categories, Pedestrian is private, non-shared while transit is public, shared. Rental services are public, non-shared because user drives the vehicle but doesn't have ownership of it. Associated with different mode-types are a set of rules and conditions by which they can update the *Geospatial layer*, such as mode-capacity, mode-specific lanes, accessibility of modes, temporal rules for specific modes (no heavy-vehicles in morning), availability of modes at certain nodes etc.
3) **Concrete Modal Layer:** In this layer concrete classes of mode-types mentioned in *Abstract Modal Layer*, are defined. As shown in Figure 1, we can categorize specific modes such as Car, Walk, Bike, B-cycle, Transit, Lyft etc. to their specific mode-types. If a given mode can be mapped to one of the mode-type in *Abstract Modal Layer*, then that mode can be added dynamically by just updating the *Geospatial Layer*. But if we cannot map a new mode, then we have to change the *Abstract Modal Layer* to define new mode-type and rules associated with it. Such mode-specific information needs to be in a standard format for better integration and interoperability. GIS files such as OSM, GeoJson provides some standard rules for Pedestrian, Non-Motorized and Motorized mode-types, while Static General Transit Feed Specification (GTFS) [6] provides some standard rules for Transit mode-type. So, *Geospatial layer* along with *Abstract Modal Layer* and *Concrete Modal Layer* can be visualized as combination of uni-modal graphs where *switch nodes* are candidate nodes where we *merge* and *link* the uni-modal graphs, as shown in Figure 2. As shown in Figure 2, *Switch nodes* are the candidate nodes where user can switch from one mode to another and edges connecting switch nodes are called *Switch edge*. Associated with each switch edge are list of pre-conditions, that needs to be satisfied to traverse that edge. Such conditions are called the Switch condition module (SCM) and depend on the user preferences, geospatial information in *Geospatial Layer* and mode-specific rules, conditions in *Abstract Modal Layer*. Such conditions along with its associated costs are encoded in the switch condition module (SCM) present in each switch node.

B. Real-time Sensor Layer

This layer contains Real-time sensor information. Some sensors are mode-specific and some are not. For example, real-time GTFS updates the *Geospatial layer* for only a specific transit agency with the help of *Abstract Modal layer*. However, some real-time such as sensors such as Traffic sensors give an aggregate information on state of the network and Weather sensors give weather information for a given area. These sensors don't depend on mode-types and thus update the *Geospatial layer* directly.

C. Service Layers

1) **Routing Layer:** This layer contains state of the art algorithms for multi-modal routing. They are variants of goal-directed search methods such as *ALT* [7] and contraction techniques such as *highway hierarchies* [8], to speed-up the shortest path computation. These techniques depend on *Abstract Modal Layer* and *Geospatial layer* to suggest routes to user for their commute. Since this layer uses mode-types defined in *Abstract Modal Layer*, it doesn't depend on concrete modes and hence improves modularity of our platform. These routing algorithms use the abstract mode-types present in *Abstract Modal Layer* and switch junctions present in *Geospatial Layer* in making multi-modal routing decisions.
2) **Analytic Service Layer:** This layer contains analytical services that use historical data and data collected by the platform. These services are used internally by the platform and some are used by various stakeholders to get holistic information. Our traffic speed prediction service gives a time-variant predicted speed information for a given link and is based on our previous works [9], [10].
3) **Simulation Service Layer:** This layer helps in analysis of Urban traffic dynamics such as congestion, vehicular emissions by doing agent-based simulation with help of MATSim [11].

Our platform has a lot of similarity with the services provided by OTP and the algorithms implemented in OTP. However, it only solves part of the problem. Firstly, all the core components in OTP are mode-specific. So, adding or deleting a completely new mode becomes difficult because the code needs to be updated and re-deployed again to add a new mode. This process is one of the reasons why OTP cant be easily deployed in regions it is not properly configured for. Secondly, OTP is a monolithic application. So, integrating services from different stakeholders becomes increasingly complex because each service providers should be independent and not affected by external services. Since a monolithic application like OTP needs to redeployed with each addition of new services, managing and maintaining becomes difficult with scale. So, we need a mobility platform that is **modular**– integrates services provided by multiple stakeholders through loose-coupling of interfaces, **extensible**– can add new data sources and build services on top of it without affecting the entire system and **resilient**– can manage the entire system without failing.

III. Enabling Modularity

A **modular** system can be characterized by functional partitioning into discrete reusable components with rigorous use of well-defined modular interfaces and making use of industry standards for interfaces. We need our platform to be modular so that different stakeholders can integrate their services without being much dependent on other services. There are some dependencies, of course, because modules may need certain features of the core library, but looser coupling of interfaces will help in quickly coordinating such changes. To this end, we follow a microservices-based architecture style [12] where a single application is developed as a suite of small services, each running in its own process and communicating with lightweight mechanisms such as ZeroMQ or HTTP. In a microservices architecture, services should be fine-grained and the benefit of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop and test. It also parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently [12].

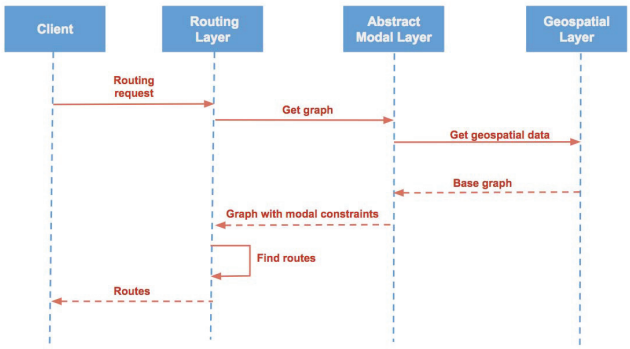

Fig. 3: This sequence diagram shows the interactions between various layers, to serve a routing request from client.

In our platform, each logical layer can be interpreted as an independent module with loose coupling between other layers. So, we can make changes to each layer with only the interfaces provided by each layer. *Geospatial Layer* contains topological information of a region and hence, we can change how this information is actually stored without affecting the upper layers, as long as the interfaces are same. For example, we can use different geospatial database for this layer such as MongoDB, PostGIS etc. *Abstract Modal Layer* abstracts the mode-specific information from the *Geospatial Layer*. Figure 3 shows a sequence diagram that details the interactions between various layers, to serve a routing request from client. Routing algorithms in Routing Layer use the abstract mode-types present in *Abstract Modal Layer* and switch junctions present in *Geospatial Layer* to make multi-modal routing decisions. So, changing Routing algorithms in *Routing Layer* may require update in *Abstract Modal Layer*. Different layers can be discovered dynamically in platform.

IV. Enabling Extensibility

The platform should be designed in such a way that it evolves as the underlying technology and requirements change and should be able to support new standards and services in the future. It should also enable addition and removal of different modes of transportation, services from different stakeholders. For example, new modes can be added by updating the *Concrete Modal Layer* and *Abstract Modal Layer*. New services can be creating new service in *Service Layers*. Figure 4 shows a sequence diagram that details the interactions between various layers, to add a bike mode to our platform. This is initiated by a service provider such as a rental bike owner who wants to add bike mode and integrate it's bike services with the platform, at run-time without any change to the source code.

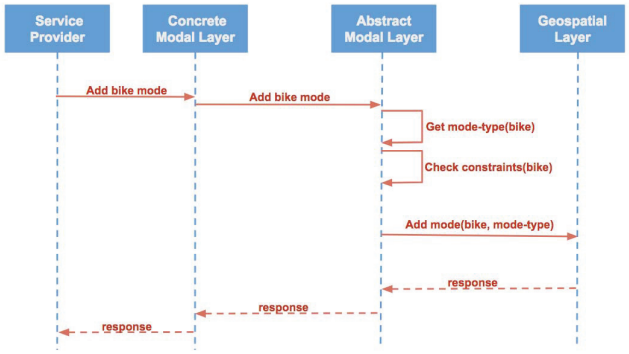

Fig. 4: This sequence diagram shows the interactions between various layers, to add a bike mode to our platform.

Evaluation. For testing code efficiency, we took an example of adding a new mode to the system and then deleting the mode added. For OTP, more than 40 classes and interfaces were changed, but with our platform, less than 5 classes were changed, to build the graph and serve routing requests to clients with new mode. Additionally, for OTP we have to redeploy the application every time code is updated, while the same doesn't happen on our platform. This improved efficiency will make it easier to extend and expand community. Because starting new projects and getting releases out faster should make it easier to join, get started and productive, and thereby lower threshold for participation.

V. Enabling Resilience

The platform should be able to elastically cope with problems and recover itself during failures so that it can provide mobility services to users without any disruptions. Each module of our platform, as mentioned in Section III, can be independently deployed in container such as Docker, LXC etc. Figure 5 shows various components of our platform during deployment. The *Service Providers* contains services from various stakeholders. All the modules are deployed in multiple nodes, in a distributed manner.

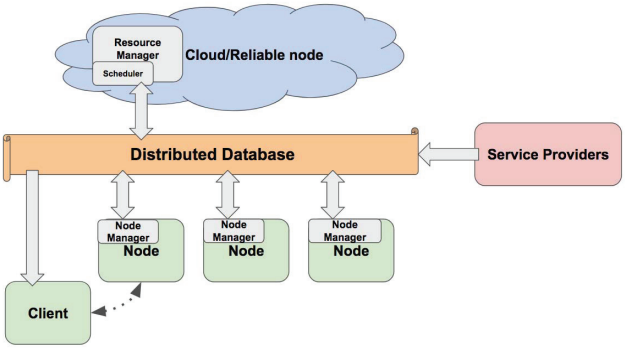

Fig. 5: This figure shows various shows various components of our platform during deployment.

We need our deployments to be reconfigured automatically so that they can deployed quickly in resource-constrained environments. In our prior work on CHARIOT [13], we developed a self-adaptive and resilient Deployment and Configuration (D&C) infrastructure for highly dynamic component-based CPS operating in resource-constrained environments. However this self-reconfiguration approach was policy-based and later in [14] we improved upon our existing work on CHARIOT by proposing a framework for formulating the system reliability as a dependence problem derived from the software component dependencies, functional requirements and physical system dependencies. This framework is codified in a domain-specific modeling language and is used to make reconfiguration decisions at runtime. This makes our deployments resilient and reliable.

Each node has a container which has modules and a *Node Manager* that coordinates communication between various containers. Each node manager has a centralized service called Consul [15] for service discovery and allows for coordination among various services in different layers, through their update managers. We use Consul for (a) Storing globally persistent data, (b) Watching changes and monitoring the system, (c) Get notified whenever a microservice fails and (d) Register and de-register services dynamically.

It should be noted that we have a thin layer of reliable cloud layer, which contains core services such as *Resource Manager* that is responsible for deployment of our modules among various resources available in our platform. When any module fails or any module needs scaling due to a large number of requests, Consul gets notified and it interacts with *Resource manager* to deploy same instance of services in another container. This makes our platform resilient and scales with increasing number of requests. Whenever a failure happens, both platform goes through multiple phases before they are able to process requests again. The phases are (a) **Failure Detection** phase occurs right after a system failure. During this phase Consul detects failure of a system component and identifies the standby service that needs to be started and configured in a new node, (b) **Configure new node** phase occurs after *Failure detection phase*, and in this phase Consul starts and configure the new node so that the identified service

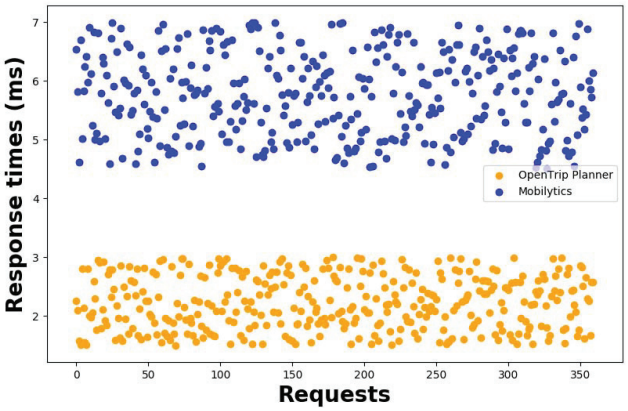

Fig. 6: This figure shows response times of all the requests made to both OTP and Mobilytics platform.

Phase	OpenTrip Planner		Mobilytics	
	mean (min)	sdv (min)	mean (min)	sdv (min)
Failure Detection	1.07	0.3	1.2	0.17
Configure new node	2.24	0.26	2.5	0.21
Restart Service	3.97	0.4	1.47	0.11

TABLE I: This table shows the mean and standard deviation time of various phases during failures. The mean time for *Failure Detection* and *Configure new node* phase is marginally higher for Mobilytics platform than OTP, while there is a significant difference in mean time of *Restart service* phase, between both platforms.

can be deployed, (c) **Restart Service** phase is the final phase which is responsible for starting the service and make it ready for serving web requests.

Evaluation. We setup OTP and Mobilytics services with Nashville OSM data and one GTFS feed of Nashville Metropolitan Transit Authority (MTA). Since OTP is not resilient, we modified OTP so that a new OTP instance starts automatically in presence of failure with the help of Consul. From a client terminal, we queried both the services with a total of 360 requests, sent discretely with 30 requests per 1-minute interval. The requests are generated by combining a range of routing parameters (time of day, maximum walk distance, transportation modes) with endpoints chosen randomly but located within 2km of a transit stop. Response times reported, represent the full round-trip time of a request to the REST API. Figure 6 response times of all the requests made to both OTP and Mobilytics platform. As shown in the figure, the response times of requests made to OTP platform are less than the requests made to our Mobilytics platform, which we expected since we are using a different graph model and routing algorithm for queries. OTP has better performance in routing because of the fast algorithms and optimization they use, which we don't have yet. However, the advantage of resilience and extensibility offsets this minor disadvantage.

Table I shows the mean and standard deviation time of various phases during a failure. As shown in the table, mean time for *Failure Detection* phase of OTP is marginally less than Mobilytics platform. It's because Mobilytics is made up of multiple microservices and distributed among multiple nodes, hence more communication is needed during failure detection than OTP, where there is only one active instance of OTP. The mean time for *Configure new node* phase is marginally higher for Mobilytics platform because of some extra dependencies that were required to be installed on our platform. This result can vary with varying implementations. Most significant difference was noted in mean time of *Restart service* phase, because a new instance for just the *Routing layer* microservice is started in Mobilytics platform which requires less initialization time of 1.47 minutes and memory of 1.4GB, while a new instance is started for OTP which required higher initialization time of 3.97 minutes and memory of 6GB.

VI. Related Research

Architecture The mobilytics platform described in this paper is a cyber-physical system. Such systems are increasingly being used in several domains of technology, engineering, and medicine such as smart power systems, smart buildings, smart public transportation, to name a few. It has been previously established that the design of these systems should be modular with an emphasis on resilience [16]. Component-based software engineering (CBSE) has been accepted as a standard practice to develop robust, modular and maintainable software stacks for embedded systems [17]. The guiding principles of CBSE are interfaces with well defined execution models [18], compositional semantics [19] and model driven analysis [20]. In the past, our group has developed several such frameworks, including ARINC-653 component model [21], [22], which combines the principle of spatial and temporal partitioning with the interaction patterns derived from the CORBA Component Model (CCM) [23]. DREMS (Distributed Real-Time Embedded Managed Systems) component model [24] extended ACM to networked cyber-physical systems that can be used by several concurrent users, by allowing configurable real-time scheduling policies in addition to configurable secure information flow policies. The micoservices based architecture described in this paper is an extension of these embedded system component models towards the enterprise system models. Design of common interfaces and abstractions is a crucial step in enabling this architecture [17].

Resilient System Design Reliability and Resilience are critical properties of computation platforms that need to provide critical service to humans. While reliability is a measure of the likelihood of of a failure, resilience is the measure of overall availability, a ratio of mean time to failure and mean time to recovery [25]. Our concept of using monitors and Consul to reconfigure the system dynamically upon failure is principally similar to the concept of "runtime reconfiguration". In [26], the authors present middleware that supports timely reconfiguration in distributed real-time and embedded systems based on services. At design-time, the schedulability and complexity of a system is analyzed and fine-tuned to bound sources of unpredictability. The resulting *Scheduled Expanded Graph* is used at runtime to determine the *Execution Graph*, which represents the application in execution. Although this approach

is flexible and relies on runtime search of the execution graph for viable reconfiguration solutions, the predictability and schedulability analysis is conducted at design-time, so system resources cannot be modified at runtime. In contrast, our approach supports runtime modification required for systems with dynamic resources.

Dynamic Software Product Lines (DSPLs) have also been suggested for dynamic reconfiguration. In [27], the authors present a survey of the state-of-the-art techniques that attempt to address many challenges of runtime variability mechanisms in the context of DSPLs. The authors also provide a potential solution for runtime checking of feature models for variability management, which motivates the concept of *configuration models*. A configuration model acts as a database that stores a feature model along with all possible valid states of the feature model. Ontology-based reconfiguration work has been presented in [28], [29], where the analytical redundancy of computational components is made explicit. On the basis of this ontology, the system can be reconfigured by identifying suitable substitutes for the failed services. Our architecture relies on the use of consul and goal-based reconfiguration [30].

VII. Conclusion

In this paper, we described a microservice architecture which made our mobility platform modular, resilient, and scalable. We also discussed the advantages of this architecture over a monolith application like OTP. As part of our ongoing work, we are using this platform for developing a socially optimal routing solution to the multi-modal routing problem. To that end, we are following a game-theoretic approach to study the multi-modal routing problem and devise solutions that will benefit society as a whole and not just individual users. We are also designing incentive mechanisms to encourage users to take public transport more often. We are also extending our data to include parking lots, rental vehicles, and ride-share services like Uber and Lyft.

VIII. Acknowledgements

This work is sponsored in part by the National Science Foundation under the award number CNS-1647015 and CNS-1528799 and in part by the Vanderbilt Initiative in Smart City Operations and Research, a trans-institutional initiative funded by the Vanderbilt University.

References

[1] U. C. Bureau, "Biking to work increases 60 percent over last decade, census bureau reports," *American Community Survey Reports*, 2014.
[2] B. McKenzie, "Who drives to work? commuting by automobile in the united states: 2013," *American Community Survey Reports*, 2015.
[3] P. S. Hu and T. R. Reuscher, "Summary of travel trends: 2001 national household travel survey," 2004.
[4] NACTO, "Ride-hailing services: Opportunities and challenges for cities," 2016, accessed: 2017-12-03.
[5] T. Pajor, "Multi-modal route planning," *Universität Karlsruhe*, 2009.
[6] Google, "Gtfs static overview," 2018, [Online; accessed 8-March-2018]. [Online]. Available: \url{https://developers.google.com/transit/gtfs/}
[7] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory," in *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2005, pp. 156–165.
[8] P. Sanders and D. Schultes, "Highway hierarchies hasten exact shortest path queries," in *European Symposium on Algorithms*. Springer, 2005, pp. 568–579.
[9] C. Samal, F. Sun, and A. Dubey, "Speedpro: A predictive multi-model approach for urban traffic speed estimation," in *Smart Computing (SMARTCOMP), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–6.
[10] A. Dubey, J. White *et al.*, "Dxnat-deep neural networks for explaining non-recurring traffic congestion," *arXiv preprint arXiv:1802.00002*, 2018.
[11] A. Horni, K. Nagel, and K. W. Axhausen, *The multi-agent transport simulation MATSim*. Ubiquity Press London, 2016.
[12] M. Flower, "Microservices," 2018, [Online; accessed 8-March-2018]. [Online]. Available: \url{https://martinfowler.com/articles/microservices.html#footnote-etymology}
[13] S. Pradhan, A. Dubey, T. Levendovszky, P. S. Kumar, W. A. Emfinger, D. Balasubramanian, W. Otte, and G. Karsai, "Achieving resilience in distributed software systems via self-reconfiguration," *Journal of Systems and Software*, vol. 122, pp. 344–363, 2016.
[14] S. Nannapaneni, S. Mahadevan, S. Pradhan, and A. Dubey, "Towards reliability-based decision making in cyber-physical systems," in *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.
[15] HashiCorp, "Hashicorp consul: Service discovery and configuration made easy," 2018, [Online; accessed 8-March-2018]. [Online]. Available: \url{https://www.consul.io/}
[16] G. Schirner, D. Erdogmus, K. Chowdhury, and T. Padir, "The future of human-in-the-loop cyber-physical systems," *Computer*, vol. 46, no. 1, pp. 36–45, 2013.
[17] G. T. Heineman and W. T. Councill, "Component-based software engineering," *Putting the pieces together, addison-westley*, p. 5, 2001.
[18] A. Basu, B. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis, "Rigorous component-based system design using the bip framework," *IEEE software*, vol. 28, no. 3, pp. 41–48, 2011.
[19] H. Schmidt, "Trustworthy components—-compositionality and prediction," *Journal of Systems and Software*, vol. 65, no. 3, pp. 215–225, 2003.
[20] P. S. Kumar, A. Dubey, and G. Karsai, "Colored petri net-based modeling and formal analysis of component-based applications." in *MoDeVVa@ MoDELS*. Citeseer, 2014, pp. 79–88.
[21] A. Dubey, G. Karsai, and N. Mahadevan, "Formalization of a component model for real-time systems," 04/2012 2012.
[22] ——, "A Component Model for Hard Real-time Systems: CCM wwith ARINC-653," *Software: Practice and Experience*, vol. 41, no. 12, pp. 1517–1550, 2011. [Online]. Available: http://www.isis.vanderbilt.edu/sites/default/files/Journal_0.pdf
[23] N. Wang, D. C. Schmidt, and C. O'Ryan, "Overview of the corba component model," in *Component-Based Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., 2001, pp. 557–571.
[24] D. Balasubramanian, A. Dubey, W. Otte, T. Levendovszky, A. Gokhale, P. Kumar, W. Emfinger, and G. Karsai, "DREMS ML: A Wide Spectrum Architecture Design Language for Distributed Computing Platform," *Sci. Comput. Program.*, vol. 106, no. C, pp. 3–29, Aug. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.scico.2015.04.002
[25] A. Haldar and S. Mahadevan, *Reliability assessment using stochastic finite element analysis*. John Wiley & Sons, 2000.
[26] M. G. Valls, I. R. López, and L. F. Villar, "iland: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 228–236, 2013.
[27] R. Capilla, J. Bosch, P. Trinidad, A. Ruiz-Cortés, and M. Hinchey, "An overview of dynamic software product line architectures and techniques: Observations from research and industry," *Journal of Systems and Software*, vol. 91, pp. 3–23, 2014.
[28] O. Höftberger and R. Obermaisser, "Runtime evaluation of ontology-based reconfiguration of distributed embedded real-time systems," in *Industrial Informatics (INDIN), 2014 12th IEEE International Conference on*. IEEE, 2014, pp. 538–544.
[29] A. Shaukat, G. Burroughes, and Y. Gao, "Self-reconfigurable robotics architecture utilising fuzzy and deliberative reasoning," in *SAI Intelligent Systems Conference (IntelliSys), 2015*, 2015.
[30] S. Pradhan, A. Dubey, S. Khare, S. Nannapaneni, A. Gokhale, S. Mahadevan, D. C. Schmidt, and M. Lehofer, "Chariot: Goal-driven orchestration middleware for resilient iot systems," *ACM Transactions on Cyber-Physical Systems (TCPS)*, 2017.