To Appear: IEEE Inteligent Vehicles Symposium (IV), Paris, June 9-12, 2019.

Service Discovery for The Connected Car with Semantic Accessors

Matthew Weber[1], Ravi Akella[2] and Edward A. Lee[3]

Abstract— **Connected cars have the potential to transform a vehicle from a transportation platform to a platform for integrating humans with a city. To that end we introduce semantic accessors (actor based local proxies for remote services) as a novel, and powerful discovery mechanism for connected vehicles that bridges the domains of Internet of Things (IoT) composition frameworks and the semantic web of things. The primary components of this approach include a local semantic repository used for maintaining the vehicle's perspective of its real-world context, accessors for querying and dynamically updating the repository to match evolving vehicular context information, accessors for services (such as parking) linked to a service ontology, and a swarmlet controller responsible for managing the above in accordance with user input. We demonstrate this semantic accessor architecture with a prototype Dashboard display that downloads accessors for new services as they become available and dynamically renders their self-described user interface components.**

I. Introduction

With global automotive sales on a downward trajectory, car manufacturers are seeing enormous opportunities in the areas of electrification, autonomy, vehicle ownership and a gamut of connected services [1] [2]. Connected cars have the potential to transform a vehicle from a transportation platform to a platform for integrating humans with a city. Traffic lights and signage (static and dynamic) have served this role in the past, but these are impossible to personalize and difficult to make sufficiently adaptive and dynamic. Here is a story from the connected city:

> As I am approaching downtown Berkeley, I want to know more than that it is safe to drive through the next intersection (a green light), but also that the garage I intended to park at is full, and that I can still make my 2PM meeting on time if I head for an alternate garage and take a tram one stop. When the battery charge on my car cannot get me to my destination, I want alternatives, where if I'm flexible on timing I can recharge while eating lunch, or if I'm not, I can swap batteries for a higher fee. If I'm getting drowsy, I want to know where to get a good macchiato or a brisk walk around a pond.

Simple solutions like putting a smartphone into the vehicle's dashboard miss enormous opportunities. The car has context that a phone does not, and its solutions can seamlessly move from one user to another, a feature that may become increasingly important as personal ownership of vehicles declines.

We identify and propose solutions to three main challenges from the vehicle's perspective in making this vision a reality. 1) Once a driver has established the intention of seeking out the service (eg. parking), how does the connected vehicle select the most contextually appropriate service from a service library? 2) How does the vehicle communicate with this service, with which it has never before interacted, using the correct radio, protocol, and API? 3) How does the vehicle coordinate its use of the service with on-board sensors or other similarly discovered services?

We propose a semantic accessor architecture for connected car integration to address these challenges. This semantic accessor architecture is a novel combination of semantic technologies, as commonly used in the Semantic Web, with the Accessors project, an open source platform for remote service communication and composition. The primary components of this approach include a local semantic repository used for maintaining the vehicle's perspective of its real-world context, accessors for querying and dynamically updating the repository to match evolving vehicular context information, accessors for services (such as parking) linked to a core service ontology, and a swarmlet controller responsible for managing the above in accordance with user input. Our prototype of this system is available on the iCyPhy repository (https://github.com/icyphy/accessors) under a BSD license.

In particular this paper's contributions include:

- An ontology for the accessor and vehicular service subject domains;
- New semantic accessor components which integrate semantic technologies with the accessors platform;
- A proof of concept implementation of the semantic accessor architecture; and
- A self-describing user interface paradigm for accessors leveraging web components to generate a dynamic interface.

Taken together, these contributions tell an end-to-end story through which dynamic vehicular context information is obtained and used for service discovery, the service's self-describing user interface is rendered to the vehicle's Head Up Display (HUD) or In-Vehicle Infotainment (IVI) displays (jointly referred as Dashboard), and interaction with the remote vehicular service is commenced in accordance to its arbitrary third-party API. By combining semantic technologies with accessors we have developed a novel, and powerful discovery mechanism. We begin in Section II with an overview of semantic technologies and their applications to automobiles, present our work on integrating semantic technologies with the Accessors project in Section III, demonstrate the Semantic Accessor Architecture in Section IV, and conclude in Section V.

II. Background and Related Work

Future connected and autonomous vehicle application scenarios span a multitude of domains including but not limited to Intelligent Transportation Systems (ITS), smart cities, smart ecosystems, Internet of Things, and telecommunications. Several competing networking paradigms have been proposed to realize efficient vehicle to everything (vehicles, pedestrians, infrastructure, user devices, networks) applications. A good survey on the state of the art for intravehicular connectivity (eg. Control Area Network (CAN)) and extravehicular networking (eg. 5G) can be found in [3] [4]. The automotive, as part of a rapidly evolving connected

[1]EECS, UC Berkeley `matt.weber@berkeley.edu`
[2]DENSO Int'l America, Inc., `ravi_akella@denso-diam.com`
[3]EECS, UC Berkeley `eal@berkeley.edu`

society, needs a semantic foundation to deal with heterogeneity in standards, communication protocols, data formats, application contexts and business players.

The semantic web was proposed by Tim Berners-Lee in 2001 as an extension of the World Wide Web that would link data to its subject matter similarly to how a web page links to another web page. These relationships can be expressed in RDF, an abstract model for semantic data as sentence-like statements about the world in triples of subject, predicate, object. For example: the sentence "A cow" (subject) "is a" (predicate) "farm animal" (object), or "The mall parking lot" (subject) "has the number of free spaces" (predicate) "45" (object). As hinted at by these examples, triples can express both abstract information about classes (cows and farm animals) as well as facts about specific instances (the mall parking lot) and raw data values (45). A database designed and optimized for RDF data is known as a semantic repository or alternatively a triple store. The W3C SPARQL Protocol and RDF Query Language (SPARQL) recommendation [5] defines both a protocol and a query language for performing SQL-like operations on a semantic repository such as queries, insertions, updates, and deletes.

In this paper we are interested in leveraging the semantic web stack toward facilitating smart thing interaction with a vehicle. The Semantic Sensor network ontology [6] and updated Sensors Observations, Samples, and Actuators (SOSA) ontology [7] codify domain knowledge regarding the abstract relationships of sensor network entities such as sensors, actuators, and the phenomena being sensed. When different semantic applications standardize on such an ontology, their knowledge bases become compatible in the sense that information may effortlessly be shared from one system to another. Pfisterer et al.'s Spitfire project coined the term "Semantic Web of Things" [8] to describe their approach in leveraging Linked Open Data [9] for the IoT. A good survey of progress to date in the Semantic Web of Things can be found in Barnaghi's report [10].

Researchers have applied semantic technologies to vehicles, but not to the best of our knowledge toward the goal of universal connectivity promised by the semantic web of things. W3C's Automotive Ontology working group[1] is developing shared vocabularies based on web ontologies for data interoperability in the automotive industry primarily for the purposes of standardizing vehicle information used for car rentals and sales. Schema.org now hosts these shared vocabularies at auto.schema.org as metadata markup for semantic web search.

Several ontology frameworks are being explored for use in autonomous vehicle technology such as Advanced Driver Assistance Systems (ADAS) to facilitate self-driving functions involving sensor actuation and control [11]. In [12], ontological knowledge systems were employed to interoperate with extra-vehicular infrastructure such as IoT devices or traffic lights to enhance traffic safety and driving experience. Our proposal is distinguished by our integration of semantic technologies with the accessors platform, designed to abstract away these communication difficulties. A connected car can just download an accessor (Section III) for a sensor or actuator without worrying about the protocol or API the accessor uses internally.

III. Semantic Accessors

In this section we provide background on existing work in the Accessors project and present our work on new semantic accessors for connected vehicles.

A. The Accessors Platform

An accessor is a downloadable chunk of JavaScript implementing a *local proxy* for a remote service, first proposed in [13]. An accessor encapsulates the complexities of communication with that service (i.e. protocol, network, API, etc.), exposing to the programmer a uniform actor interface for sensors, actuators, local machine resources, or remote web resources. Fig. 2 gives an example of the SemanticRepositoryQuery accessor's external actor interface and Fig. 3 shows a fragment of its internal JavaScript implementation. In the typical pattern of accessor usage, a programmer initiates interaction with the resource represented by an accessor by providing an input to the local accessor's input port and receives a response back from the resource on the accessor's output port. Interaction between remote systems can be locally coordinated by linking the input of an accessor to the output of another. Such a network of accessors is called a swarmlet[2], an example of which can be seen in Fig. 5.

The accessor platform is intended to be for the IoT what a web browser is for the internet. In the IoT it is common for each service to require its own service-specific application, complicating interaction and composition across services. But on the web, the browser is a single flexible platform that is compatible with all kinds of services. For example, when a user wants to make a web transaction with a bank, the user can navigate to the bank in their browser and request a web page that tells the browser what to display and how to perform the transaction over the internet. The web page acts as a *local proxy for the bank's remote service*.

Like the bank's transaction web page, an accessor is a downloaded component which tells the swarmlet how to perform a particular interaction with a remote service. Also like a browser, the same swarmlet can be compatible with many different remote services: it just downloads and uses each service's accessor. This accessor component architecture was proposed by Brooks et al. in [15] to implement an augmented reality demo which interacts with multiple remote services.

Similar actor oriented platforms for IoT composition are in development, notably Calvin [16] and Node Red [17]. Compared to these alternatives, accessors bring deterministic models of computation from the timing-critical world of embedded systems [18] and interface theory [19]. Additionally, accessors have a greater focus on dynamic discovered component behavior via the mutable accessor [15] (see Section IV-A.3) and, of course, integration with semantic technologies discussed in this paper.

B. Accessor and Service Ontology

The environment in which accessors and swarmlets execute is known as an accessor host. As of this writing, the Accessors project has mature hosts that run on Node.js, in standard browsers, and in Java using Nashorn (Cape Code), as well as experimental hosts Cordova and Duktape for mobile and embedded devices respectively. To allow accessors to be platform independent, these hosts are responsible for providing native implementations for common environmental functions and modules. However not every host has the hardware to support every accessor. A Node.js host running in an intelligent vehicle may, for example, not support the cameras module if the vehicle has no cameras. Before a

[1]https://www.w3.org/community/gao/

[2]The name swarmlet is in reference to the "Swarm at the Edge of the Cloud" [14]

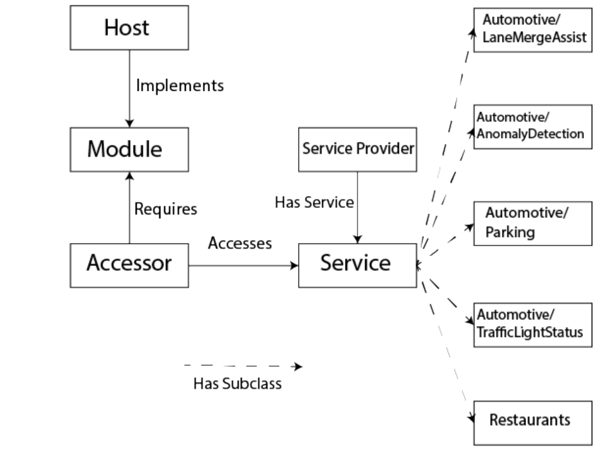


Fig. 1: Diagram of accessor and service ontologies.

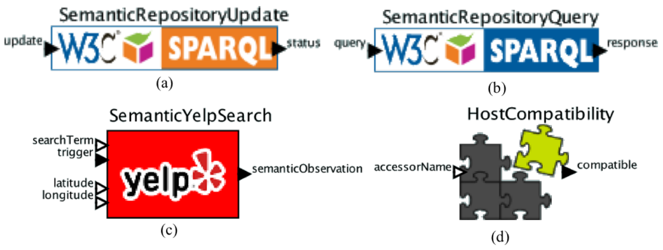


Fig. 2: Actor interfaces to the accessors developed in this work

newly discovered accessor can be selected, it is important to know whether this is the case.

To address this problem we developed an accessor ontology, a core excerpt[3] of which is presented in Fig. 1. We developed scripts that iterate through all known accessor and host libraries to populate the ontology with individual data for specific accessors, hosts, and modules. To be consistent with the principles of Linked Open Data [9], the name of an accessor in this generated ontology is a URI linking to a file containing its JavaScript implementation.

The ontology in Fig. 1 shows that an accessor may have an additional link to the service it accesses. For example, a parking service accessor ought to access a particular real-world parking service (an instance of Automotive/Parking) as the remote resource it proxies. Such a parking service is linked to its service provider through the Has Service relationship.

The combined Accessor and Service ontology is a powerful resource for semantic service discovery, and methods for its effective use are the core proposal of this paper. We present a scheme for managing these dynamic updates and service discovery in Section IV-A.1. It is worth noting these updates can include numeric, string, and other basic datatypes. It is a common misconception that ontologies cannot link basic data to concepts.

C. *Semantic Accessors*

In this research we have developed a new variety of accessor that interacts with semantic technologies: a semantic

[3]The full accessor ontology is considerably more complex with concepts for Accessors, Accessor Interfaces, Inputs, Outputs, Parameters, Modules, Hosts, Services, and relationships for interface implementation, inheritance, and contained subaccessors among others. However, as the majority of these concepts are not related to service discovery, we have omitted them from Fig. 1.

```
exports.handleResponse = function(message){
  var writer = N3.Writer({ prefixes:
   { sosa: 'http://www.w3.org/ns/sosa/',
     rdf: 'http://www.w3.org/1999/02/22-rdf-syntax-
         ns#',
     xsd: 'http://www.w3.org/2001/XMLSchema#',
     schema: 'http://schema.org/'
    }});
  var obsNode = store.createBlankNode('
     SemanticYelpSearchObservation');
    ...
}
```

Fig. 3: Partial JavaScript code listing for SemanticYelpSearch

accessor. We illustrate the actor interface of four of these new accessors in Fig. 2 and elucidate their behavior in this section.

SemanticRepositoryUpdate and SemanticRepositoryQuery in Figs 2(a) and 2(b) proxy a SPARQL compatible semantic repository over HTTPS—it is irrelevant whether the repository runs locally or remotely. A SPARQL delete or insert command sent to the update port of the SemanticRepositoryUpdate will return with the success or failure of the operation on the accessor's status output. Similarly a SPARQL select, construct, ask, or describe query sent to the SemanticRepositoryQuery accessor will return with the query results on the accessor's response output.

Semantic repositories as conventionally used in the Semantic Web are relatively static entities used to maintain information in abstract domains such as medicine, government, linguistics, or media. We believe semantic repositories have enormous potential for use as a dynamic knowledge base for the immediate real-world context of a vehicle. Encapsulating the SPARQL protocol in these accessors integrates semantic repositories with the Accessor platform, where SemanticRepository Update and Query can be connected to other accessors in a swarmlet and used to bring the semantic repository into a dynamic control loop with sensors and actuators.

The SemanticYelpSearch and HostCompatability accessors depicted in Figs. 2(c) and 2(d) represent more advanced semantic capabilities. When HostCompatability receives the name of an accessor as input, it checks the current state of the accessor's ontology (described in Section III-B and depicted in Fig. 1) to determine whether the host it's running on is capable of providing that accessor's required modules. When the query is complete, the Boolean answer to the compatibility question is produced on its output.

The HostCompatability accessor works by querying this ontology for the list of modules supported by the current host and compares them against the list of modules required by the given accessor. Some services may be known in advance and hence statically encoded into the ontology before deployment, but others may be dynamically discovered (for example, nearby Restaurant services via SemanticYelpSearch) and inserted into the ontology with SemanticRepositoryUpdate.

The SemanticYelpSearch accessor is an example of a general class of *semantic sensor* accessors. These accessors produce as output semantic observations in the form of Turtle syntax ontologies. The name "semantic sensor" is not an accident: the output ontology uses concepts from the W3C SOSA[4] ontology for sensor observations. SemanticYelpSearch takes a location and search topic as input and when triggered, interacts with the Yelp API to produce as

[4]SOSA is a rethinking of the classic *Semantic Sensor* Network ontology.

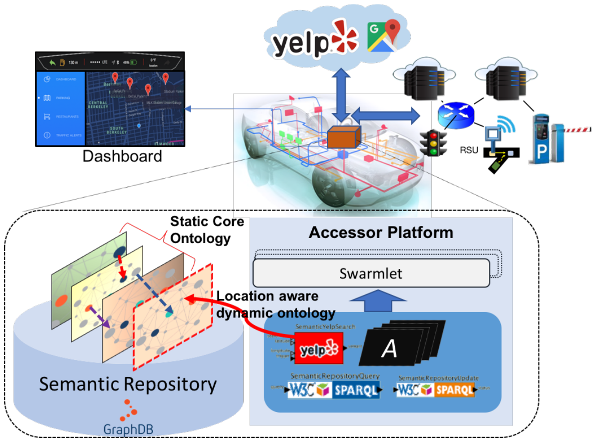


Fig. 4: Elements of the Semantic Accessor Architecture.

output a semantic observation of nearby local businesses.

Since the Yelp API does not use a semantic representation for its results, SemanticYelpSearch uses the N3 library to translate Yelp results into schema.org's LocalBusiness ontology. In fact, the ease of *encapsulating an arbitrary non-semantic API with an accessor to translate its results into a semantic output* is one of the chief advantages of semantic sensor accessors. Mapping data to standard SOSA and schema.org ontologies facilitates automatic integration of data obtained from a semantic sensor accessor with any other SOSA and schema.org compatible ontologies, such as the service ontology described in Section III-B and depicted in Fig. 1.

IV. Semantic Accessor Architecture

In this section we present an architecture for semantic accessors that enables semantic service discovery and a dynamic user interface for service interaction. The main system components of this approach are diagrammed in Fig. 4, and the swarmlet controller that coordinates these components is given in Fig. 5. The controller has four interconnected parts: the semantic repository maintenance swarmlet, which manages information about the world; a service selection user dialogue, which determines user intent for service interaction; a mutable controller for web component accessors, which downloads and sets up communication with the selected service; and an accessor for the dynamic user interface itself. As shown in Fig. 4, the controller and semantic repository run locally on the vehicle and display a user interface on the vehicular dashboard. SemanticSensor accessors in the controller proxy remote services in the cloud such as Yelp, and also nearby fog resources such as traffic lights and roadside units. When service discovery is complete, the user can interact with cloud and fog resources (eg., paying for parking) through UIComponent accessors (Section IV-A.3).

A. Swarmlet Controller

We address the aspects of the swarmlet controller[5] depicted in Fig. 5.

1) Semantic Repository Maintenance: Knowledge in the vehicle's local semantic repository comes from two sources: the static core ontology initialized into the repository with basic information about service concepts (or universally relevant services) and the contextually relevant dynamic ontology provided by SemanticSensor accessors. Directed by user interest from the dashboard, the semantic repository maintenance swarmlet periodically acquires new location-specific information from semantic sensors (such as SemanticYelpSearch) and writes the data into the dynamic ontology. This use of a semantic repository is analogous to a streaming database like Apache Kafka or AWS Kinesis but comes with the advantages discussed at length in sections II and III.

To avoid endless maintenance of stale data in the SemanticRepository,[6] a SPARQL delete command also runs periodically to delete old semantic observations. A semantic repository that supports the GeoSPARQL standard for geographic data (like GraphDB) can also be set up with a SPARQL delete command to periodically delete old observations by geofencing and deleting observations spatially located far from the vicinity of the vehicle.

2) Service Selection Dialogue: With the semantic repository kept fresh with contextually relevant data, the controller is ready to respond to user requests for service discovery. In response to a service topic from the user interface (eg. parking services), the service selection dialogue finds relevant host-compatible services using the SemanticRepositoryQuery and HostCompatability accessors respectively. We developed a prototype user interface for the resulting IVI display, shown in Fig. 6. The point isn't to recreate Yelp or Google Maps, but to present the user a fusion of contextual information from the repository they couldn't get from any individual cloud service.

3) Mutable Controller and Dynamic User Interface: Once a service is identified by the service selection dialogue, its accessor is downloaded by the mutable controller and sent to a mutable accessor. A mutable accessor is a special higher-order accessor, presented in [15], which can be thought of as an open hole in the swarmlet that fills itself with the accessor code it receives on its "accessor" input. In this way the downloaded accessor is *reified* (i.e. made real and plugged into the model) in place of the mutable, as shown in Fig. 7.

This particular mutable accessor requires that the accessor it reifies for a connected car service implement the UIComponent accessor interface. Upon reification in a mutable, a UIComponent produces a definition for an interactive user interface component on its UIComponent output in the form of a web component[7], which will be rendered as part of the user interface. In this way a UIComponent is self-describing!

Our prototype of the dynamic user interface is a React.js app[8]. The prototype communicates with the swarmlet controller over web socket. Web apps have been rendered in IVIs (for example, using application frameworks provided by Automotive Grade Linux (AGL)). When the React app receives a web component from the mutable controller, the component is rendered as an interactive card, as shown in Fig. 8 for a parking component. User interaction with such

[5]For clarity of presentation, this diagram omits slots for multiple semantic sensors, multiple service selection dialogues, multiple downloaded accessors, and the message routing system which ensures in-swarmlet communication is directed to the correct slot.

[6]Plus, it is against the terms of Yelp's API usage to store their data for more than 24 hours and "build another Yelp". The same is likely true with other service providers.

[7]Web components are a newly released standard for creating encapsulated bundles of HTML, CSS, and JavaScript, which are reusable as interface components across web apps. For more information see: https://www.webcomponents.org/introduction.

[8]Built off the Black Dashboard React template by Creative Tim (MIT Licensed)

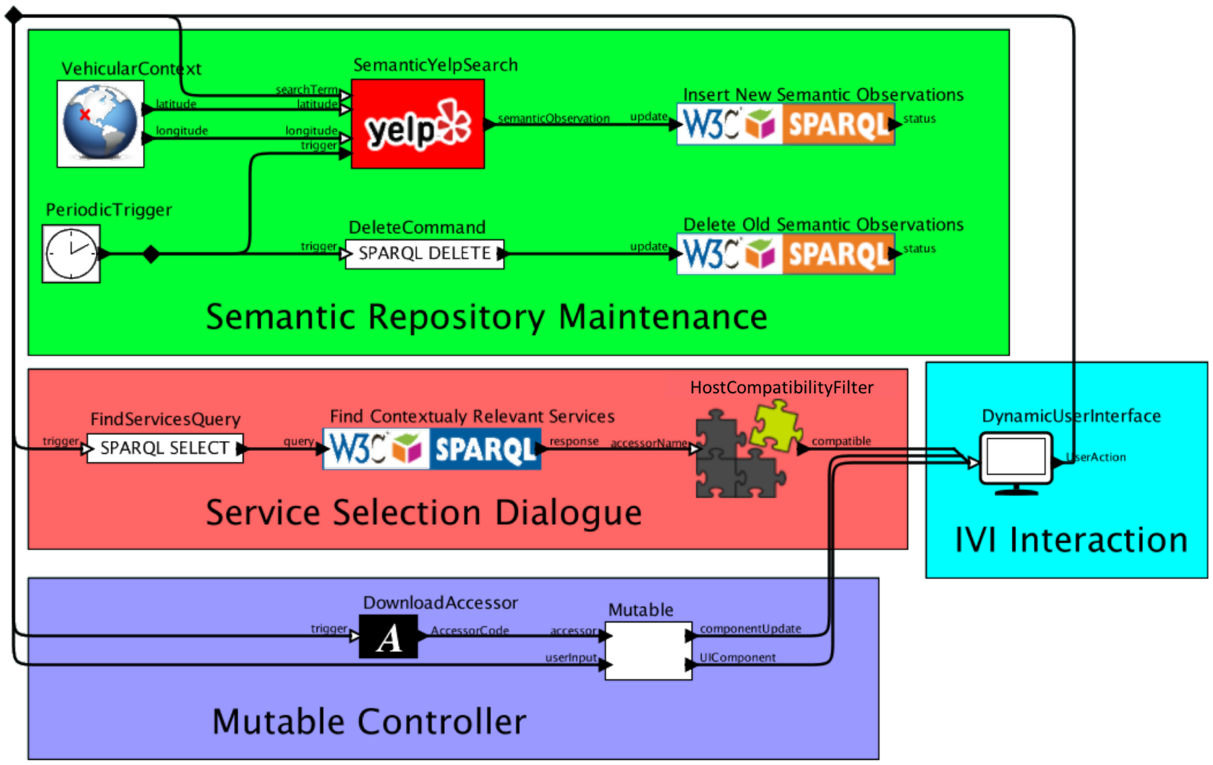
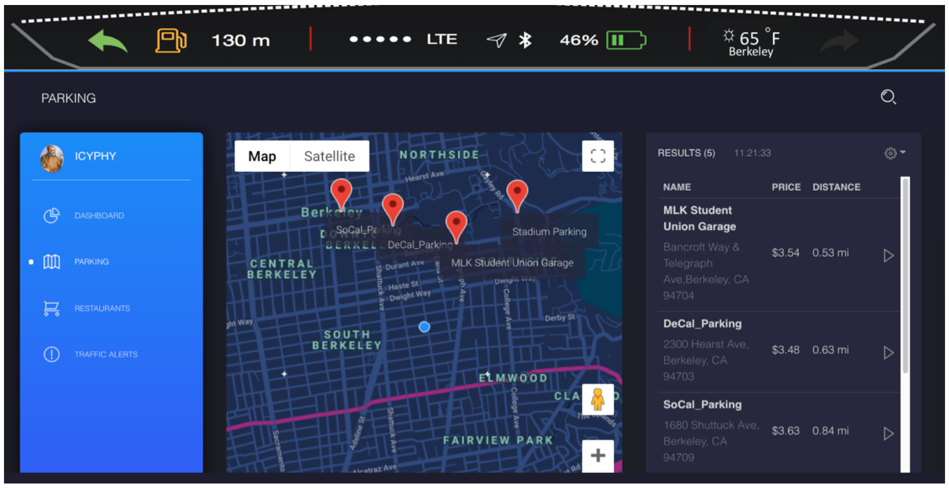


Fig. 5: Illustration of the Semantic Accessor Architecture's swarmlet controller



NAME	PRICE	DISTANCE
MLK Student Union Garage Bancroft Way & Telegraph Ave.Berkeley, CA 94704	$3.54	0.53 mi
DeCal_Parking 2300 Hearst Ave, Berkeley, CA 94703	$3.48	0.63 mi
SoCal_Parking 1680 Shuttuck Ave, Berkeley, CA 94709	$3.63	0.84 mi

Fig. 6: Screen capture of our user interface for a parking selection dialogue. Parking options are displayed in both a Google maps component and a table sorted by price or distance. Selecting a marker in the map or a row of the table triggers the mutable controller to download the accessor for that particular parking service. Marker locations and distance are dynamically generated from the semantic repository.

a card is directed back to the mutable controller and the Mutable's userInput port, providing the reified accessor (and the service it proxies) the opportunity to respond to user interface events. The reified accessor may update the UI component in the React app by producing an output over componentUpdate.

For example when a parking service is selected from the parking dialogue in Fig. 6, the selected parking accessor is downloaded and reified in the mutable. It produces the parking web component shown in Fig. 8 on its UIComponent output. When the parking component is first rendered in the React app, it doesn't know about the current status of parking spots so it initiates communication back to the mutable controller and the refied parking accessor's userInput port. The parking accessor communicates with the parking service it proxies to obtain this information and sends it back to the instantiated component via the componentUpdate output. The parking component receives the data and renders the current parking information. Similar to the parking component we also prototyped a video component intended to show a clear view of traffic from an intersection when the driver's view is obstructed, and a restaurant menu component for ordering from a fictional restaurant.

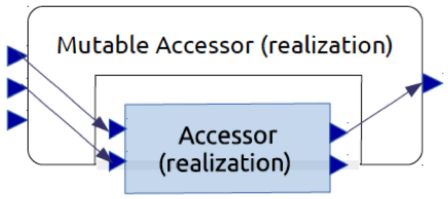
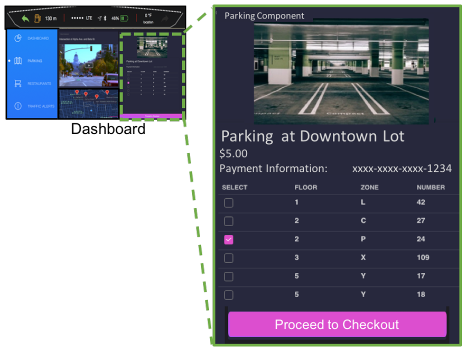


Fig. 7: A mutable's behavior can change during the lifetime of a swarmlet



Parking at Downtown Lot
$5.00
Payment Information: xxxx-xxxx-xxxx-1234

SELECT	FLOOR	ZONE	NUMBER
☐	1	L	42
☐	2	C	27
✓	2	P	24
☐	3	X	109
☐	5	Y	17
☐	5	Y	18

Fig. 8: Prototype implementation of a user interface web component for an example parking service. This web component, and our other prototype components were built with React.

V. Conclusion

From traffic infrastructure that helps you make the light from a mile away and avoid an accident, to restaurants that let you order before you arrive and save you a parking spot, the future holds exciting possibilities for connected cars. But for these capabilities to be fully realized, vehicles must acquire dynamic knowledge of their world. They must learn about connected services in the environment, how the services relate to each other, and how to interface with a service's intelligent capabilities. We proposed Semantic Accessors as a fusion of the technologies of the Semantic Web with the IoT interaction of the Accessors project. We presented an ontology for accessors and vehicular services that enables the HostCompatability accessor to enhance swarmlets with the power of semantics and semantic sensors like Semantic-YelpSearch to bring the flexibility and composition potential of accessors to bear on semantic repositories. The Semantic Accessor Architecture we prototyped alongside a dynamic user interface for Dashboard displays demonstrates how these components can be combined to build flexible and contextually intelligent applications.

In future work, we intend to investigate logical and machine learning inference to enhance the knowledge stored in a semantic repository. For example, a faulty street address for a restaurant might be identified by inferring a discrepancy between the advertised geolocation, street address, and geolocation where the vehicle actually parks. It would also be interesting to enhance the semantic repository maintenance swarmlet with mutable slots for discovered semantic sensors. In this way, a semantic repository might accumulate data sources as it progressively infers newly relevant sensors. And finally, enhancing our dynamic interface prototype with security and authentication methods from Kim's locally centralized, globally distributed authentication agents is a prudent avenue for future investigation [20].

Acknowledgements

The work in this paper was supported in part by the National Science Foundation (NSF), award #CNS-1836601 (Reconciling Safety with the Internet) and the iCyPhy Research Center (Industrial Cyber-Physical Systems, supported by Avast, Camozzi Industries, DENSO International America, Inc., Ford, Siemens, and Toyota.

References

[1] Springer India-New Delhi, "Automotive revolution & perspective towards 2030," *Auto Tech Review*, vol. 5, no. 4, pp. 20–25, Apr 2016
[2] P. Bansal and K. M. Kockelman, "Forecasting americans long-term adoption of connected and autonomous vehicle technologies," *Transportation Research Part A: Policy and Practice*, vol. 95, pp. 49 – 63, 2017
[3] J. E. Siegel, D. C. Erb, and S. E. Sarma, "A survey of the connected vehicle landscape—architectures, enabling technologies, applications, and development areas," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 8, pp. 2391–2406, Aug 2018.
[4] S. Mumtaz, K. M. S. Huq, M. I. Ashraf, J. Rodriguez, V. Monteiro, and C. Politis, "Cognitive vehicular communication for 5g," *IEEE Communications Magazine*, vol. 53, no. 7, pp. 109–117, July 2015.
[5] S. Harris, A. Seaborne, and E. Prud'hommeaux, "SPARQL 1.1 Query Language," Mar. 2013
[6] M. Compton, P. Barnaghi, L. Bermudez, R. García-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W. D. Kelsey, D. Le Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, and K. Taylor, "The SSN ontology of the W3c semantic sensor network incubator group," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 17, pp. 25–32, Dec. 2012
[7] K. Janowicz, A. Haller, S. J. D. Cox, D. L. Phuoc, and M. Lefrancois, "SOSA: A Lightweight Ontology for Sensors, Observations, Samples, and Actuators," *arXiv:1805.09979 [cs]*, May 2018, arXiv: 1805.09979
[8] D. Pfisterer, K. Romer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, A. Kröller, M. Pagel, M. Hauswirth, M. Karnstedt, M. Leggieri, A. Passant, and R. Richardson, "SPITFIRE: toward a semantic web of things," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 40–48, Nov. 2011
[9] T. Berners-Lee, "Linked Data - Design Issues," July 2006
[10] P. M. Barnaghi, W. Wang, C. A. Henson, and K. L. Taylor, "Semantics for the internet of things: Early progress and back to the future," *Int. J. Semantic Web Inf. Syst.*, vol. 8, pp. 1–21, 2012.
[11] A. Armand, D. Filliat, and J. Ibañez-Guzman, "Ontology-Based Context Awareness for Driving Assistance Systems," in *IEEE Intelligent Vehicles Symposium (IV)*, Dearborn, United States, June 2014, pp. 1–6
[12] S. Fernandez, R. Hadfi, T. Ito, I. Marsa-Maestre, and J. R. Velasco, "Ontology-based architecture for intelligent transportation systems using a traffic sensor network," *Sensors*, vol. 16, no. 8, 2016
[13] E. Latronico, E. Lee, M. Lohstroh, C. Shaver, A. Wasicek, M. Weber, and others, "A Vision of Swarmlets," *Internet Computing, IEEE*, vol. 19, no. 2, pp. 20–28, 2015
[14] E. A. Lee, B. Hartmann, J. Kubiatowicz, T. Simunic Rosing, J. Wawrzynek, D. Wessel, J. Rabaey, K. Pister, A. Sangiovanni-Vincentelli, S. A. Seshia, D. Blaauw, P. Dutta, K. Fu, C. Guestrin, B. Taskar, R. Jafari, D. Jones, V. Kumar, R. Mangharam, G. J. Pappas, R. M. Murray, and A. Rowe, "The Swarm at the Edge of the Cloud," *IEEE Design & Test*, vol. 31, no. 3, pp. 8–20, June 2014
[15] C. Brooks, C. Jerad, H. Kim, E. A. Lee, M. Lohstroh, V. Nouvelletz, B. Osyk, and M. Weber, "A Component Architecture for the Internet of Things," *Proceedings of the IEEE*, vol. 106, no. 9, pp. 1527–1542, Sept. 2018
[16] P. Persson and O. Angelsmark, "Calvin — Merging Cloud and IoT," *Procedia Computer Science*, vol. 52, pp. 210–217, 2015
[17] N. K. Giang, M. Blackstock, R. Lea, and V. C. Leung, "Developing IoT applications in the Fog: A Distributed Dataflow approach," in *2015 5th International Conference on the Internet of Things (IOT)*. Seoul, South Korea: IEEE, Oct. 2015, pp. 155–162
[18] C. Jerad and E. A. Lee, "Deterministic Timing for the Industrial Internet of Things," in *2018 IEEE International Conference on Industrial Internet (ICII)*. Seattle, WA, USA: IEEE, Oct. 2018, pp. 13–22
[19] M. Lohstroh and E. A. Lee, "An interface theory for the internet of things," in *Software Engineering and Formal Methods*. Springer, 2015, pp. 20–34
[20] H. Kim and E. A. Lee, "Authentication and Authorization for the Internet of Things," *IT Professional*, vol. 19, no. 5, pp. 27–33, 2017