Data-Directed Contextual Relevance in the IoT

Colin Maxfield and Christine Julien
The University of Texas at Austin
{colinmaxfield, c.julien}@utexas.edu

***Abstract*—The relevance of data created in or about the IoT has a strong reliance on the *context*, especially spatiotemporal context, of the device and application perceiving it. To ensure that applications perceive data items that are relevant to the current context, it is necessary to restrict when each item is available. To control an application's perceptions of data availability, data items are often put in a group with other similar items, and a static rule is applied to determine when that data can be seen by applications. Such rules are fairly rigid, and the burden is on application developers to manage individual data items and their access policies, including making sure data distribution stays up to date relative to the context that influences data availability. We posit that the development of applications that need to access contextually relevant data can be greatly simplified by enabling a data item itself to control how and when it is available to applications. To realize this simplified programming paradigm, we introduce the *datalet*, an abstraction of a piece of data that understands its contextual relevance and dictates how (i.e., when and where) it is available based on that application's context. The datalet allows the application developer to focus on the application logic that relies on available data, without worrying about how to store, update, and distribute contextually-sensitive data to (distributed) application instances. To show how datalets are used by application developers to construct an application, we create an augmented-reality game that uses datalets to make elements of game play available based on the player's spatiotemporal context. The video of this demonstration is on YouTube at: https://youtu.be/snFhokswWpc**

I. Introduction

With the emergence of the Internet of Things (IoT), everyday physical spaces are embedded with digital capabilities. Context greatly influences what data a user of an application should perceive, given the current time and the user's current location. As this vision of the IoT is becoming a reality, application developers need to be able to respond by taking advantage of the contextual relevance that is inherent in IoT data. Programming abstractions should allow developers to focus on the application logic that consumes contextually relevant data while hiding the details of how data is produced, maintained, and distributed. Existing work in supporting IoT application developers focuses on discovering devices, establishing connections to them, and passing messages across connections. Instead, the above motivates us to focus on the *data* itself. Because of the obvious importance of contextual relevance, we are particularly interested in how data accessibility and availability relate to application context. Concretely, it should be straightforward for application developers (and potentially end users who create data) to provide declarative specifications of the contextual elements of data availability (in particular space and time), delegating to an implementation layer the nuts and bolts of *how* data is made available in a way that matches the application's specification.

We examine three applications, one of which we use throughout and as the driver of our demonstration. First, consider an application to support the safe mobility of children to school, for example in support of *walking school buses* or *bicycle trains*[1]. A walking school bus entails one parent who is responsible for chaperoning several children together to or from school. An application on a parent's smartphone could communicate with small sensors in each child's backpack, ensuring the status of each child relative to the bus. Data about a child should be available only in the vicinity of the child, only during school transit hours, and only to authorized adults. Second, consider a restaurant that uses coupon advertisements to attract customers off the street. These coupons may be shared only with people within walking distance of the restaurant and only during times when the restaurant's capacity is under 60%. Finally, consider an augmented reality game, similar to the wildly popular game Pokémon Go[2]. In this game, players attempt to find (and fight) "nearby" monsters. The particular monsters available to a given player at a given time are dependent on that player's context, most notably on the player's proximity to the monster's location.

Using existing programming abstractions, a developer is focused on how to use available communication links to request and distribute data. Consider the augmented reality game; when the player is active, the game should periodically send the player's location to a backend server, which compares the player's location to the locations of the existing monsters, determining which are sufficiently close to the player. While there is significant work on contextually indexed databases, the data is still fundamentally *static* and unaware of the rules imposed on its consumption. That is, data simply exists, with all control over how it is consumed relegated to the application. Our view is a fundamental shift; our goal is to allow data items themselves to define and execute *policies* that specify how, when, where, and to whom the data item is available given the context. We encapsulate this perspective as a *datalet*, which captures both the information content and the spatiotemporal availability policies associated with it. From this perspective, *a datalet is a spatiotemporal scope that defines the application-level accessibility of a piece of data*. This perspective, described in Section III, is abstract and declarative and relates naturally to the physical world that

[1]http://www.walkingschoolbus.org/
[2]http://pokemongo.nianticlabs.com/en/

978-1-5386-1589-8/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-C.2017.13

the data captures. However, system support requires a more pragmatic and imperative perspective tied to the fact that a datalet must be realized within limitations of real hardware and networks. From this alternative perspective, *a datalet is the combination of a data item and its replicas, and the associated policy governing their placement, replication, migration, and persistence, based on properties of the physical, system, and application context*; we describe this perspective and how it is presented to application developers in Sections IV and V.

Our datalet approach simplifies application development in the following concrete ways:

- Elevating data items and their availability as programming primitives is intuitive for application developers in the IoT.
- It is easy to implement different availability policies for different data items; this is in contrast with current practice, in which application-specific server logic must be aware of and control individual data items' policies.
- Changes to a data item's availability rule are isolated from other availability rules, minimizing the impact of changes.
- The server logic is entirely application-agnostic; datalet programmers can instead focus on (a) defining application-level data via datalets and (b) the application logic for consuming available datalets.

This demonstration showcases the datalet abstraction, showing how applications create datalets and define data availability, and the programming abstractions presented to application consumers of datalets. We also demonstrate how datalets simplifies the backend since executing rules that determine datalet availability is delegated to the data items themselves.

II. Related Work

Other work in mobile and pervasive computing also focuses on data as a programming primitive. This work is largely founded on tuple spaces. However, as with other recent efforts in the IoT, this work focuses primarily on programming abstractions that direct the dissemination of data across devices in an area. TOTAM [9] allows a tuple's propagation rules to generically reference the *context* to determine if a tuple should be propagated before it is transmitted to a neighboring device. This is similar to our approach in that it imbues the data itself with some control, but the focus remains on active dissemination and not simple availability. Further, the allowed context rules only look for the presence of specific other tuples in the local tuple space. This approach has been extended to allow each tuple to determine whether it is in the right context or not [10]. Although similar to our idea of datalets in that the data itself understands its own relevance, the context rules and their implementation are entirely left up to the application developer, requiring the developer to focus explicitly on how *communication* is implemented relative to the context. Instead, we promote abstracting away communication as well. This has the added benefit of allowing diverse underlying communication implementations to be exchanged for one another without impacting the application logic. For instance, the centralized-server based implementation of datalets we describe in this paper can easily be replaced with a peer-to-peer dissemination with no impact on the application implementation.

Modern systems make it straightforward to explicitly search for (or register for notifications about) location-based data. These approaches are driven by applications that *request* location-tagged data. Therefore, while the user's experience appears seamless as the user moves through space and time, the application interaction programmed by the developer is a potentially inefficient series of interactions with some communication substrate. In contrast, adaptive data dissemination preemptively pushes data in opportunistic networks (e.g., among vehicles [5] or wireless sensors [6]) to make it available at the time and place an application requires. Data persistence and replication in mobile networks [2] can adapt to devices' connectivity to each other or to an infrastructure [11]. Mobile bazaars connect "publishers" to "subscribers" [4], [7]. *Floating content* is attached to spatial *anchor zones* [3], [8], and *hovering data clouds* periodically refresh data that "hovers" on co-located devices (e.g., in a traffic jam) [12]. These target areas are defined by the presence of devices or humans, and while they support expressive notions of "context-awareness" [1], they do not put data producers in charge of data availability. In contrast, in *datalets*, data itself drives its availability, defining where, when, and to whom it should be made available.

III. Datalets Language

A datalet consists of: (*i*) a unique *identifier*; (*ii*) a *data item* that can, in principle, be anything; (*iii*) an *owner* that can update the data item; and (*iv*) a spatiotemporal policy that defines the data item's *availability*. A policy includes access control, which may be influenced by the (spatiotemporal) context. A policy's *precondition* controls the datalet's existence (when the precondition is true, the datalet is available), while its *availability* specifies where, when, and to whom the datalet is available. It is intuitive to think about availability from the perspective of an entity that views the data item; the data item is available to an entity if (*i*) the precondition is satisfied and (*ii*) the entity satisfies the availability constraints. These constraints can be based on time, location, or the values of other properties of the entity. Any combination of constraints can be specified using Boolean logic operators.

As an example, Fig. 1 gives a conceptual specification of a datalet that describes a monster. The *monster* datalet is created by the game administrator and exists for some preset amount of time (specified in the precondition). During that time, the data associated with the monster is available to nearby players that are of the specified level. As players move around and their location changes, they might gain or lose access depending on their new location. To illustrate the availability policy, Fig. 2 shows a chupacabra monster and the zone where it is available. Two players are shown in green inside the zone that have access to that monster; the two players in red inside the circle do not have access because they do not meet the level requirement; and two in red outside of the zone do not have access due to not meeting the spatial requirement.

```
/* the monster datalet only exists for a set time and is
      available only to nearby players of the required level */
datalet monster
 dataletID : id
 owner : gameAdminID
 data : name, description, image, type, location
 policy
  precondition : time ∈ [t_start, t_end]
   availability : (within(50m) AND level([7,15]))
```

Fig. 1. Example datalet for a monster

Only the datalet knows its availability policy and how that policy is related to context. In the example above, a player's device only receives data items about monsters that the player is allowed to see, but the player does not know what rules made that possible. The screenshot on the right of Fig. 2 is a view of the indicated player (Joe Smith) from the screenshot on the left. Joe sees two available monsters, but he does not know each monster's region of availability or the level required to view the monster. The datalet model also greatly simplifies the server implementation. Instead of having to understand and enforce application-specific rules, the server can simply ask the datalet if a given user should have access, given that user's context and profile. This allows for a datalet's conditions to be easily changed without affecting other parts of the system. It also makes the backend implementations of applications that rely on datalets completely generic.

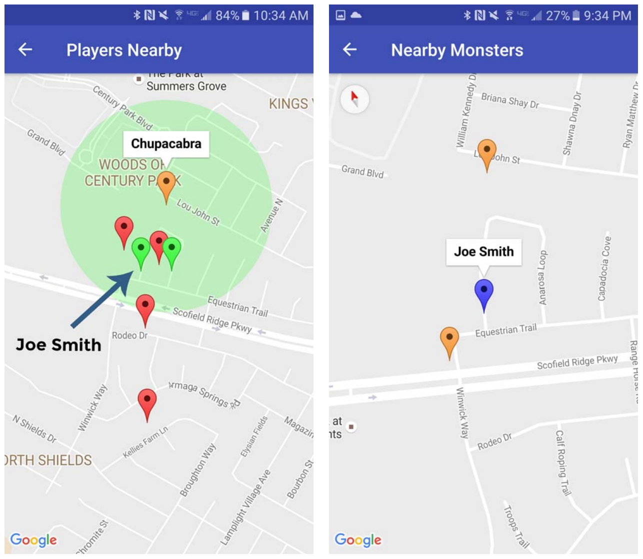

Fig. 2. Monster with nearby players (left); the monster is available to players within the green circle of a certain level. Player view of nearby monsters (right); note that the player cannot see the broader availability of the monsters.

IV. Programming with Datalets

Our programming support for creating and using datalets is built around classes and methods exposed by the datalet API[3]. In this section, we walk through those abstractions, using our augmented reality game on Android as an example. We look first at the steps necessary to create a datalet; then we examine the programming steps involved in using datalets.

The most novel aspect of creating a datalet is defining its precondition and availability policy. These policies have elements of three types: a *schedule*, which constrains the time of availability; a *location*, which constrains the space of availability, and a *profile*, which constrains availability based on aspects of the data consumer. Each type derives from the `Condition` class. To combine these in more complex expressions, an application uses the `Policy` class, which can contain a group of conditions that are evaluated together using a single Boolean operator (i.e., "AND" or "OR"). A `Datalet` contains fields to represent a datalet and allows for the developer to easily interact with the data and its precondition and availability policies. The `User` class encapsulates the profile of an individual user. Fig. 3 shows how an application developer creates datalets representing monsters.

```
/* create a policy that checks a schedule;
    the datalet is available from 6:30-6:45PM
    starting 11/3/2016 and repeats daily
    until 11/30/2016 */
Policy precondition =
 Condition.schedule("<", "15", "6:30 PM",
               new LocalDate(11, 3, 2016))
    .repeatsDaily(new LocalDate(11, 30, 2016))
    .finish();

/* create a policy that checks a user's
    location and profile; the datalet is
    available to players within 50 meters and
    of levels 7-15 */
Policy availability =
 Policy.createAnd(
    Condition.location("<", "50").finish(),
    Condition.profile("range", "7,15",
          "application.beasthunter.level")
       .finish());

Map<String, String> data = new HashMap<>();
data.put("id", "B_125asd90124");
data.put("name", "Chupacabra");
data.put("description", "A goat sucking
        monster that is relatively easy to
        catch");

Datalet monster =
 new Datalet("ADMIN_ID",
          new GPSPoint(30.4441245,
                 108.012490124),
          data, precondition,
          availability);
```

Fig. 3. Application code for creating a monster datalet

To ease application development on the data consumer side, we define a RESTful interface. After a datalet has been created or edited, the application developer can use the provided rest client to retrieve available datalets from the server. Figure 4 shows the major components of this interface. To adhere to the RESTful properties, each method takes a callback that it uses to invoke the appropriate application code once the

[3]App/API Source code can be found at https://gitlab.com/mpc-research/datalets-app and can be used under the MIT license.

necessary communication has finished. The first three methods are used by the datalet *creator*; the last method is used by a data consumer to retrieve datalets that are available to a given user given his current context. Note that we opted to implement a pull mechanism for retrieving available datalets; an alternative implementation would be for the server to push the available datalets based on periodically updated knowledge of the potential consumers' profiles.

```
public class DataletsRestClient {
  void createDatalet(datalet, callback);
  void updateDatalet(datalet, callback);
  void deleteDatalet(dataletID, callback);

  void getAvailableDatalets(userID, callback);
}
```

Fig. 4. API of the provided rest client

V. Realizing Datalets

A major advantage of the datalets approach is that, once datalets are defined, the backend server implementation can remain entirely application-agnostic. Specifically, to enable datalet applications requires a server that stores the datalets and evaluates each datalet's individual precondition and availability policies. The prototype datalet server is written in Java using the Spring Web Framework[4]. To store datalets and users, the server uses MongoDB, which allows individual policies and data items to be easily changed by their owners.

Any application using datalets interacts with the server through the RESTful API in Fig. 4 to create, update, and retrieve datalets. This model allows an application to easily ask for the available datalets for a given user. In our example, when the game application uses the RESTful API to request the datalets for a given user, the server asks each monster datalet to evaluate its availability policy using the player's profile (including the player's location). The datalet simply evaluates whether the player should have access to it; if the result is positive, the server includes the datalet's data item in the result it returns to the application. The server does not know what the datalet is used for, what specific policies the datalet has, or what data is inside the datalet. This model allows the server to handle any datalets for any application without having to be at all aware of the application's implementation.

VI. Proof of Concept and Evaluation Plans

To evaluate datalets as an abstraction, we used them in developing the monster game, which serves to demonstrate that the datalet concept fits very naturally into an application's programming interface. We also demonstrated that the backend can, in fact, remain completely application agnostic.

Since the policies themselves are focused on basic parts of context relevance in the IoT and not just our augmented-reality game, we also plan to evaluate their use in other applications and by other developers. Alongside the proof of concept, we include a few descriptions of applications for the IoT. Potential application developers can use these application descriptions, along with the API of datalets, to write their own datalets to create data items for these applications.

We plan a more extensive controlled empirical study as part of our future research agenda. The goal is to assess how intuitive it is for an application developer to use datalets in the process of writing an application. By providing the abstraction of datalets, we aim to determine if, by assuming a datalet perspective in the design stage of an application, developers can spend the majority of their time on the application logic and user experience instead of on figuring out how to represent the needed data and how to control its availability.

VII. Conclusions and Future Work

We presented datalets as a way to enable applications to leverage data-directed contextual relevance in the IoT. Using current approaches to share contextually relevant data, a developer spends a considerable amount of time writing a system to control data access that results in a set of static rules that are not trivial to adapt with the growth of the application. With datalets, an application developer can focus on the application logic and user experience, while retaining the flexibility to easily generate complex data availability policies that can be different for each data item. Our implementation uses a centralized server to store and maintain user profiles and datalets. In future work we want to explore a hybrid approach that combines centralized storage and processing with local capabilities via cloudlets and device-to-device communication as we explore additional application uses of datalets.

Acknowledgements

The authors thank Gian Pietro Picco and Amy L. Murphy for discussions that formed the genesis of this work.

References

[1] S. Ali et al. A simple approximate analysis of floating content for context-aware applications. In *MobiHoc*, 2013.
[2] D. Gavidia and M. V. Steen. A probabilistic replication and storage scheme for large wireless networks of small devices. In *MASS*, 2008.
[3] E. Hyytia et al. When does content float? characterizing availability of anchored information in opportunistic content sharing. In *Infocom*, 2011.
[4] U. Lee et al. FleaNet: A virtual market place on vehicular networks. *IEEE Trans. on Vehicular Tech.*, 59(1):344–355, 2010.
[5] I. Leontiadis et al. Persistent content-based information dissemination in hybrid vehicular networks. In *PerCom*, 2009.
[6] C. Lu et al. A spatiotemporal query service for mobile users in sensor networks. In *ICDCS*, 2005.
[7] M. Motani et al. PeopleNet: Engineering a wireless virtual social network. In *MobiCom*, 2005.
[8] J. Ott et al. Floating content for probabilistic information sharing. *Pervasive and Mobile Comp.*, 7(6):671–689, December 2011.
[9] C. Scholliers, E. G. Boix, and W. D. Meuter. Totam: Scoped tuples for the ambient. In *Proc. of the CAMPUS Workshop collocated with DisCoTec'09 federated event*, volume 19, pages 19–34. EASST, 2009.
[10] C. Scholliers, E. G. Boix, W. D. Meuter, and T. D'Hondt. Context-aware tuples for the ambient. In *On the Move to Meaningful Internet Systems, OTM 2010*, pages 745–763, 2010.
[11] K. Thilakarathna et al. MobiTribe: Cost efficient distributed user generated content sharing on smartphones. *IEEE Trans. on Mobile Computing*, 13(9):2058–2070, 2014.
[12] A. Wegener et al. Hovering data clouds: A decentralized and self-organizing information system. In *IWSOS*, 2006.

[4]Server source code can be found at https://gitlab.com/mpc-research/datalets-server and can be used under the MIT license.