Future Generation Computer Systems 28 (2012) 619–632

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Supporting ubiquitous sensor-cloudlets and context-cloudlets: Programming compositions of context-aware systems for mobile users

Seng W. Loke

Department of Computer Science and Computer Engineering, La Trobe University, Victoria 3086, Australia

ARTICLE INFO

Article history:
Received 28 April 2011
Received in revised form
7 September 2011
Accepted 17 September 2011
Available online 7 October 2011

Keywords:
Composition operators
Context-aware pervasive systems
Sensor-cloudlets
Context-cloudlets

ABSTRACT

Increasing widespread use of sensor and networking technologies are yielding ubiquitous sensors and applications that pervade daily life. At the same time, context-aware pervasive computing has experienced tremendous developments in terms of context modelling and reasoning, and applications. Such developments coupled with a cloud computing model are yielding sensor-cloudlets and context-cloudlets based on sensors and applications deployed as services that can be harnessed in applications on-demand, ad-hoc and on a pay-per-use model. Sensor-cloudlets and context-cloudlets depend on and adapt to the available resources at the time, and involve context-aware systems (including sensors) that need to be dynamically composed as needed. This paper first outlines current trends and key issues and challenges in sensor-cloudlets and context-cloudlets. We then present a key contribution of this paper, which is an application of an abstract model of context-aware systems for specifying compositions of context-aware systems used in sensor-cloudlets and context-cloudlets. We show how expressions in our formalism can be embedded into a programming language (which we show via an example extending the logic programming language Prolog). We then present numerous examples illustrating applications expressed in our extended Prolog language. We also show how compositions specified in our formalism supports estimating the reliability and cost of using such compositions of resources in computations, in a well-defined semantics. Finally, we describe meta-level control operators on evaluation of queries posed to compositions of resources and specify a service-based interface on context-aware systems. We conclude with issues to be tackled in the future.

1. Introduction

For an application to adapt to the physical environment in which it inhabits, there must be a means to acquire sensory and context information. Numerous sensor network applications have emerged over the last decade, from habitat monitoring to structural monitoring, indoors and outdoors [1–4]. But the range of sensor based applications are moving from specialized narrow domains to domains useful for the general public, from hazard detection, local real-time traffic jam information, to crowd sensing. As sensors increase in variety and daily applications, sensing will become ubiquitous, providing unprecedented real-time knowledge of the physical world [5].[1] In the same way that one might utilize IT resources, and even more so, individuals might utilize such sensing information or context knowledge. Indeed, the notion of sensor grids [6] and sensor Webs [7,8] have emerged where users should be able to link into sensor network infrastructures anywhere and anytime as needed.

Sensor data often needs to be analyzed and interpreted. From raw sensor data, we can obtain context information. According to Dey [9], context is information about the situation of entities, including an individual, object or place. The essential element of such context is sensory information. Data from sensors must be interpreted and fused in order to provide useful context. Such interpretation of sensor data yields meaningful context, perhaps represented in some ontology [10,11]. In context-aware computing, a general definition of sensors is hence, any device or mechanism that can provide context information [12]. There has been work on general infrastructures for providing context information (e.g., [13,14]), where users can tap into such infrastructures to acquire context information. Reasoning with context [15] then yields further context or knowledge about situations [12,16,17], or aggregated context. Two different systems can reason with the same sensor data and provide slightly different context information, or the same contexts may be inferred from different sensors and different sensor data [18]. Also, in response to the same context or situation inferred, different actions might be taken.

E-mail address: s.loke@latrobe.edu.au.

[1] As an example, the goal of Hewlett Packard Labs' Central Nervous System for the Earth, or CeNSE, project is to "build a planetwide sensing network using billions of tiny, cheap, tough and exquisitely sensitive detectors". http://www.hpl.hp.com/news/2009/oct-dec/cense.html.

0167-739X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2011.09.004

A mobile device can utilize sensors on itself or sensors from the surrounding infrastructure. Similarly, a mobile device can reason about sensor data and context using its own resources [19,20], or query infrastructure context sources (and other mobile sources) for such information (e.g., by abstracting context as data items in a distributed tuple space model implemented over a network of peers [21,22]). In fact, resources and infrastructure around the mobile user and his device(s) may be utilized on an on-demand, ad-hoc basis, and mixed-and-matched to suit the mobile user's need at the time. This notion of combining resources as needed and having resources available as-a-service on a pay-per-use basis relates to the notion of cloud computing.

There are many definitions of cloud computing [23] but they share similarities. From [24], cloud computing refers to "a pay-per-use model for enabling available, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". Commercial offerings of platform or applications as a service are already available, such as EC2 and S3 from Amazon [25], Google's App Engine,[2] Microsoft's Azure[3] and Manjrasoft's Aneka [26]. The idea of resource pooling as needed is an important elasticity characteristic of cloud computing, and then the resources are mixed and matched to meet end-user needs.

Generally, a context-aware system employs sensors, reasoning (or thinking) components, and components mapping context to actions. Sensing and reasoning resources might be mixed and matched in an ad-hoc and on-demand manner for an end-user application. To recognize or detect a given situation, a set of sensors might be selected and consulted on demand, and composed with appropriate context and situation reasoning components, to provide a context-aware application for the end-user. Such resources form a cloud of resources which have been called *sensor clouds* [27] comprising a collection of sensors pooled together and governed to provide a service directly to end-users or application providers, and if the cloud also includes components to fuse sensor data into context or higher level situation knowledge, we call such combinations of sensors and reasoning components *context clouds*. Hence, a *context cloud* may utilize a *sensor cloud*. In sensor clouds and context clouds for context-aware applications formed in an ad-hoc and on-demand manner, resources needed may include not only storage and computational resources, but also sensors and context reasoning components. Such sensors and reasoning components might be exposed as services (e.g., with REST or SOAP interfaces).

In recent times, the notion of *cloudlets* has been proposed that relates more to the mobile user's perspective, where a cloudlet is "a trusted, resource-rich computer or cluster of computers that's well-connected to the Internet and available for use by nearby mobile devices" [28]. In this paper, when such a resource includes primarily sensors (e.g., a sensor network, or even positioning technology infrastructure), we call such cloudlets, *sensor-cloudlets*, and when surrounding context sources or servers of context information, context reasoning engines, and situation inference applications, are packaged with the cluster of computers available for use by a mobile device, we call these *context-cloudlets*. Our notion of cloudlet refers to the set of resources (which could include clusters of computers nearby a mobile device and other sensor-enhanced mobile devices) used in an application by a mobile device. Both the notions of sensor-cloudlets and context-cloudlets relate to SaaS and might be thought of as a subset of SaaS where sensors and context are provided as services. As noted in [28], a cloudlet is different from a cloud in that a cloudlet tends to be (1) transient, (2) localized to the current environment of the mobile user (which could even be a business premise or coffee shop), as opposed to some air-conditioned machine room, (3) ownership is decentralized (e.g., owned by local business or current site such as a museum), rather than centralized ownership by a large provider, (4) network is via LAN and WLAN instead of Internet scale, and (5) there are fewer users at a time, instead of thousands. Also, in our case, as the name implies, a sensor or context cloudlet makes use of a relatively small set of (typically local) sensors, context servers, or or context-aware systems, compared to the large (Internet-scale) set of resources as typical in a cloud model.

[2] http://code.google.com/appengine/.

[3] http://www.microsoft.com/windowsazure/.

This paper discusses the notion of sensor-cloudlets and context-cloudlets, and outline challenges associated with the widespread usage of such a cloudlet model of context-aware systems. In tackling one of the challenges, we also highlight a formalism for composition of sensors and key components supporting a mix-and-match approach for assembling resources to build a context-aware system - such "elasticity", and ad-hoc mix-and-match of selected resources are typical of the cloud computing model.

The key point of this paper is that a language specifying sensor-cloudlets and context-cloudlets greatly helps the use and composition of resources because

- compositions of resources can be described declaratively using expressions at a high level of abstraction, useful for programming applications;
- the declarative descriptions of composed resources facilitates the analysis of the cost and reliability of compositions; and
- meta-level operators are useful for managing the execution of expressions (representing compositions of resources).

We demonstrate all of the above features in a language we propose, and outline how this language of operators for sensor/context-cloudlets can be implemented via a service-oriented interface.

In the rest of this paper, we first provide an overview of trends towards the emerging infrastructure of ubiquitous sensors in Section 2, and in Section 3, we discuss scenarios and challenges with sensor and context cloudlet computing. In Section 4, we then provide a formalism for mix-and-match of resources specialized for context-aware cloudlet applications, giving their operational semantics, with examples of compositions forming cloudlets; we also describe meta-level control operators over computations involving compositions of resources and their semantics, and describe a service-based interface for context-aware systems for our model. Section 5 concludes with future work.

2. Background and related work

This section provides background for our work and introduces related work.

2.1. The emerging infrastructure for sensor-cloudlets and context-cloudlets: ubiquitous sensing

There is a growing infrastructure of sensors, some of which will become publicly available for use and integrated into applications.

Sensor grids. The notion of the sensor-grid was proposed in [29,30] which integrated sensor networks with grid computing, so that computational resources from the grid can be employed to process sensor data real-time. This idea will become extremely important as sensor networks become more widespread. Services can be deployed over such combined infrastructures over Web standards, such as the NICTA Sensor Web architecture, which provides stateful Sensor Web services to clients [31]. With the development of cloud platforms [32], the integration of sensing infrastructures with cloud resources to offer services via Web/Internet standards is the next step (e.g., sensor clouds noted earlier [27]), and for example as already demonstrated in the ECG data analysis service in [33]. Indeed, such a paradigm for large scale sensing with cloud-based processing will continue to be important. Our work here complements such work by allowing descriptions of compositions of resources in sensor-cloudlets and context-cloudlets in specific applications.

Sensing on the mobile. Many mobile devices (e.g., smartphones or on-car computers) now are equipped with sensors (or can be augmented with sensors) and an API to access sensor outputs, and we imagine greater sophistication and variety of sensors on mobile devices on the user as well as on cars, buses, trams and trains (as we describe below). There has been work on acquiring and using context information on mobile platforms [34,35,19,20,36,37], and a trend towards sharing such aggregated context information via what can be called sensor mashups [38] and context mashups [39–41], where information from geo-tagged sensors are aggregated and "pushed out" to the Web to be visible.

Urban sensing. There have been a number of Web sites providing map-based visualization of real-time data acquired via sensors at different geographical locations, many of which have been contributed by the public and are community-based. There are others with more specific information such as real-time information about bus arrivals in the Cambridge TIME project,[4] and in the Melbourne trams tracking project.[5] Some sites provide visualizations of GPS tracked cycling and walking routes from voluntary contributors, and car routes. Other projects examine traffic loads in real-time[6] making such information available from mobile devices.

In addition, urban sensors including air pollution monitors (e.g., as mentioned in mobiscopes [42,43]), noise sensors, traffic monitoring services, and construction work detection cameras, around the city might be able to help pedestrians choose the best and most pleasant path through a part of the city, without causing anyone to miss the bus. The urban sensors might be queried by users on an on-demand basis, at reasonable costs (yet this can be significantly profitable with thousands of users per day).

Citizen science. An emerging development where individuals can make a difference is what has been termed citizen science where "networks of volunteers perform or manage research-related tasks such as observation, measurement or computation". In citizen science, according to Paulos et al. [44], "we need to expand our perceptions of our mobile phone as simply a communication tool and celebrate them in their new role as personal measurement instruments capable of sensing our natural environment and empowering collective action". Urban sensing [45–48], where urban environments are probed and tracked using a participatory "Web 2.0"-style collaboration can be facilitated by the right mobile tools. There are also research projects that use electronic street signs to communicate air quality outdoors [49] and sensors to monitor air quality indoors [50], which can be made accessible to mobile devices. Cyclists allergic to pollens might employ (and maybe pay for) pollen sensors in the environment to help them avoid such hazards.

Social network sensing. Social networks of sensors (on mobile devices of networked users) might also be employed to detect one another's presence. The project that links Bluetooth proximity sensing to facebook social networks,[7] and Bluetooth presence sensing [51,52] and friend location detection provides another kind of context information for users. Activity recognition via sensor data gathered using mobile devices can be shared as done in [34] and used in applications, whether it is to compete with one another to encourage fitness or to provide a greater sense of closeness, groups of sensors and contexts can be utilized for such social enhancement purposes. Such peer-to-peer tracking might on-demand, on a pay-per-use basis, tap in to the current positioning infrastructure of an environment, or utilize wide-area GPS-based solutions, depending on indoor and outdoor features.

Bubble sensing. In bubble-sensing [53], a mobile device initiates a sensing task and the presses a button on the phone to affix the task to a location, which persists until a timeout. The mobile device can register the sensing task with a bubble server. Sensors at that location or other mobile phones (with sensors) might be employed to service the sensing task. Sensed data from phones servicing the sensing task can be uploaded via the cellular network or WiFi. The notions of sensor-cloudlets and context-cloudlets have similarities with the notion of *bubble-sensing*. However, learning from the ContextToolkit,[8] a context-aware application comprises sensors, an encapsulation of sensors as context widgets, context aggregators and context interpreters, and so, our proposal involves heterogeneous resources (not only sensors, but also sensor data interpreters, reasoning modules, etc.) composed on-demand for a user's context-aware application. Moreover, in our approach, while we assume appropriate resources can be discovered as needed, we also provide operators to compose resources and show how to embed these compositions within the full expressive power of a programming language. Also, with our cloudlets, a billing model is required.

Possibilities. Hence, mobile devices act as sensor devices and become portals to access sensory information from sensors in the environment, acting as aggregation points. Mobile devices act as sensors in two ways: via sensors on those devices uploading sensor data or interpreted context information, or users of those devices manually providing such data (e.g., uploading video feeds, or annotations of events and spots for tourists). There will be an increasing amount of information available at a particular location, which users might want to be aware of, or which can be analyzed to infer semantic context or knowledge about situations.

Within the cloudlet computing model, intermediaries might work with third-party sensor network administrators, context information providers and application developers, and package context-aware applications that end-users can employ on a pay-per-use basis. A key idea is that such sensor-intensive context-aware applications can be configured and delivered as a service to end-users on an ad hoc and on-demand basis. The opportunities lie in the multiple categories that services can be provided. In the typical cloud computing model, we may have storage, database, information, process, application, platform, integration, security, management or governance, testing and infrastructure as-a-service [24]. In selling usage of context-aware applications, a question is what might be offered as a marketable or chargeable service?

Here, we might be more specific and a sensor, a whole sensor network, context information, context reasoner, context-models/context ontology, situation reasoner, situation models/situation ontology can be as-a-service, and also a packaged context-aware application can be provided as-a-service. Even more specifically, we might have positioning-as-a-service where an environment might sell its tracking infrastructure on a pay-as-you-use basis. In fact, multiple parties might be involved in the delivery as-a-service of a particular context-aware application for a user.

4 http://www.cambridgeshire.gov.uk/transport/around/buses/-realtime.htm.

5 http://tramtracker.yarratrams.com.au/.

6 For example, the iPhone App TrafficAU provides real-time traffic information in Australia.

7 http://news.bbc.co.uk/2/hi/technology/6949473.stm.

8 http://contexttoolkit.sourceforge.net/documentation/UserGuide.html.

3. Scenarios and challenges

In this section, we describe scenarios for the cloudlet computing model we are focusing on in this paper, and then outline challenges.

3.1. Example scenarios

In shopping malls, indoor positioning will become increasingly commonplace, paving the way for context-aware advertising and allowing the user to connect to ambient infrastructure sensors and product information databases on-the-fly. A cloud of sensors, context providers and database resources can be pooled together, for example, to support a father's shopping task to find something for his daughter's birthday. Such a cloudlet formed with the current sensors and computers around the user is transient, adapted to the current available resources and personalized for the user.

Consider a "Christmas social shopping" assistant application for Mary and her friends shopping in Taipei during the week before Christmas. With this application as a service, the locations of friends, navigational maps within the shopping areas, location-based advertisements, crowd and traffic real-time data, weather reports (including rain, temperature are included) are integrated and provided as a package. Different parties might be aggregated to provide such a service, including some delivering raw sensor data as a service, others positioning and people tracking services, and others retail information. A cloudlet of resources is formed to provide the whole-packaged service to the end-user. There are issues of privacy when context information, such as location of users, are shared. In fact, there may be an incentive model for users to share their context information, in exchange for monetary or other kinds of benefits such as free access to other services, essentially being not just a consumer but a provider when joining a cloudlet. Mary and her friends might choose to opt-in to have their locations tracked, yet maintaining a privacy policy with the company. There could also be a case where competitors are present, each providing a similar kind of service, where end-users then need a way to choose from a selection of service providers. For example, there could be two or more providers for a "Christmas social shopping" assistant (each with their own points of differentiation), who may utilize the same positioning infrastructure (delivered as a pay-per-use service by another party).

Consider another example of integration-as-a-service in the sensor-cloudlets context. Wider area hostpots-based networking and positioning might be provided by means of an overlay over a collection of WiFi hotspots. Several parties provide hotspot access and another party provides a layer of integration and transparency of a service to end-users that spans multiple hotspots. Sensors connected to different hotspots might be accessed by the end-user uniformly via this layer.

There are numerous sensor-based context-aware applications that can be delivered as services to end-users. Consider event awareness and delivery of video feed for events. When a local event happens (e.g., a Japanese culture dance in an open air mall in association with a cultural festive season), certain providers might be able to provide live video feeds accessible via hotspots to nearby users or beyond. There could be multiple providers able to provide such a service, some just a plain video feed and others with running commentary (to add value), such feeds sold as a service to end-users viewable from their mobile devices on a pay-per-use basis. Some end-users, willing to spend more, might be able to get multiple video feeds of the same event at the same time, delivered via multiple providers, thus having a better (but more expensive) experience of multiple perspectives concurrently—the video feed providers might be companies or even individuals (who simply managed to get a good seat in front of the open air stage for the performance and decided to actually capitalize on that).

The combination of sensors on mobile devices (e.g., smartphones, cars and bicycles) and in the surrounding infrastructure, together with reasoning algorithms (e.g., data mining or knowledge-based), will yield a platform that can be tapped-in by mobile users on an on-demand and ad-hoc (and even pay-per-use) basis. Markets can form where resources from multiple providers are integrated as needed or multiple providers offering similar services over an area might compete.

3.2. Challenges for sensor-cloudlets and context-cloudlets

Similar challenges as those found in normal cloud computing arise with sensor-cloudlets and context-cloudlets. When providers collaborate to provide sensory and context information, and context-aware applications-as-a-service, issues arising include pooling of resources when needed (including dynamic discovery and reservations of dynamically provisioned resources possibly from multiple parties), agility and adaptations to available resources and changing user needs, costs and (static and dynamic) pricing structures and appropriate business models, billing, resource metering, accessibility from different platforms, scalability, security, trust, privacy of usage, sustainability, maintenance, governance of services, QoS negotiation and guarantees, reliability of services, service level agreements, application development complexity, composability of clouds and services, and user experiences.

In particular, from the perspective of assembling a set of resources (e.g., sensors and context reasoning engines, and computers on which to do processing) and in order to discover appropriate context-aware systems, whether they contain all three components of sensors, interpreters and situation reasoners or only some, there is a need for

- representation of sensors, sensor networks, sensor data, context information, context ontologies/models, situation knowledge, and situation ontologies to facilitate their discovery and usage;
- representation of components that reason with and interpret sensor data and processes and components that aggregate contexts to infer situations;
- mechanisms for composition or mix-and-match combinations of sensors and components that yield a particular context-aware application as a service to end-users, and the associated representations of compositions, creation of compositions together with adaptations and recomposition mechanisms;
- search and discovery infrastructure for components from sensors, context providers, to context reasoners; and
- a system for governance of services delivered (be it delivering sensor data as a service, context information delivery as a service, situation reasoning as a service, or a whole context-aware application-as-a-service).

In the next section, we present a formal abstract model of context-aware systems, and a formalism for representing compositions of context-aware systems, sensors and applications. We show how our model maps to the service-oriented computing paradigm. Note that we do not focus on resource discovery in this paper but assume that there is an infrastructure for finding (appropriately described) resources required for a composition.

4. Composition for sensor-cloudlets and context-cloudlets

The above challenges are numerous. In this section, we address one aspect of the above challenges, that of providing a mechanism to compose sensors and other components to create a context-aware application (to be delivered as a cloudlet service to end-users). More specifically, we focus on representations of such compositions in a formal way which allows reasoning with such compositions, changes to such compositions, and cost analysis, as we demonstrate below.

A formalism for representing compositions was first introduced in [54], but not used in cloudlets. Here, in this paper, and not in previous work:

- we provide a subset of the operators and demonstrate how we can extend the formalism to make location explicit and to perform cost analysis; and
- we show how to embed compositions into a programming language and provide numerous examples of compositions for use in cloudlets. Our operators are not merely composing services[9] but composing context-aware systems, each of which may expose its functionalities as a set of services.

4.1. A general model of a context-aware system

Here, we use *context-aware system* as a general term to refer to three types of systems:

- systems comprising only of sensors,
- systems comprising only of context interpretation and reasoning components, and
- systems with sensors and context interpretation and reasoning components.

The general model of a context-aware system R is of the form (Σ, Π, Θ). Σ is a finite set of sensors where each sensor σ_i is a function which senses a part of the world (in which the sensor is situated at the time) $W \in \mathbf{W}$ and produces a set of sensor readings $G \in \mathbf{G}$, i.e. $\sigma_i : \mathbf{W} \rightarrow \mathbf{G}$. G is represented in the form of equalities or inequalities (i.e., ranges) on sensor readings, and we use the function symbol $reading(\sigma, T)$ to denote the reading of a sensor σ at some time T (or simply, $reading(\sigma)$ if time is implicit). We let $\mathbf{W}$ and $\mathbf{G}$ remain opaque as our discussions do not require their details, but explicit and precise definitions could be given for them; for example, W can be defined as a three-dimensional volume of spherical space (and $\mathbf{W}$ as a set of such spaces) at some location of a prescribed size where the sensor is contained, and G can be a sensor stream comprising a set of timestamped data samples, and $\mathbf{G}$ a set of such streams of readings.

The interpreter Π can then be defined as a mapping from sensor readings $\mathbf{G}$ to some context (e.g., a symbolic location such as a room number) $C \in \mathbf{C}$ (which we assume are concepts grounded in some ontology such as SOUPA[10] and CONON [57], or other ontologies as surveyed in [58,59,10]), i.e., $\Pi \subseteq (\mathbf{G} \times \mathbf{C})$. So, given $W \in \mathbf{W}$, and suppose $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, and $\sigma_1(W) = G_1, \ldots, \sigma_n(W) = G_n$ (for $G_i \in \mathbf{G}$), then Π can be applied to interpret each G_i to obtain a set of contexts $\{C_1, \ldots, C_n\}$, denoted by $\Pi(G_i) = C_i$ (taken here to mean $(G_i, C_i) \in \Pi$). The situation reasoner Θ is a pair of relations (Θ_c, Θ_s) with Θ_c, mapping sets of contexts to situations, and Θ_s, mapping sets of situations to other situations, i.e.

$$\Theta_c \subseteq (\wp(\mathbf{C}) \times \mathbf{S})$$

where $\mathbf{S}$ is a set of situations (again, which we assume are grounded in some ontology), and

$$\Theta_s \subseteq (\wp(\mathbf{S}) \times \mathbf{S}).$$

Examples of aggregating context to infer situations can be found in the literature [60,10,61–63]. Our model makes the distinction between the notions of context and situation following the work such as [57,61,64,12,9], where as mentioned earlier, in Dey's definition [9], context information is used to "characterize the situation of an entity", modeled via Θ_c. While we do not commit to any ontology here, in practice, we assume that systems do return inferred contexts or situations using concepts from a "well-known" ontology in order to facilitate interoperability.

Moreover, we define the relation denoted by $\vdash$ between systems and pairs of the form (W, S) where $W \in \mathbf{W}$ and S is some situation, such that $R \vdash (W, S)$ if and only if R recognizes S when sensing part of the world W. A situation that is recognized by a system is then either computed from contexts (recognized by some sensors and the interpreter) via Θ_c or aggregated via Θ_s from other recognized situations (it could be from both context and situations as appropriately modeled). This meaning of the relation $\vdash$ can be expressed recursively as follows in a rule written in the form $\frac{premises}{conclusion}$:

$$\frac{\begin{array}{l} \text{Where } \Theta = (\Theta_c, \Theta_s), \quad \text{either} \\ \text{(i) } [(\{C_1, \ldots, C_m\}, S) \in \Theta_c \\ \text{for some } m, \text{ where for each } C_i, i \in \{1, \ldots, m\}, \\ \Pi(\sigma_j(W)) = C_i, \text{ for some } j \text{ and } \sigma_j \in \Sigma], \\ \text{or} \\ \text{(ii) } [(\{S_1, \ldots, S_k\}, S) \in \Theta_s \text{ for some } k, \\ \text{where for each } S_i, i \in \{1, \ldots, k\}, (\Sigma, \Pi, \Theta) \vdash (W, S_i)] \end{array}}{(\Sigma, \Pi, \Theta) \vdash (W, S)}$$

(one-system).

Similarly, we can represent context recognition by a system R, i.e., $R \vdash (W, C)$ for some context C, where $\Pi(\sigma(W)) = C$, for some $\sigma \in \Sigma$. And one could generalize the notion of recognition to include not just situations but also contexts.

4.2. Syntax of expressions and operational semantics of operators

We consider a language of expressions representing compositions of context-aware systems involving three binary operators we call union, intersection and tight-union.

A relevant question here is why an operator approach? We adopt an operator approach in the spirit of and drawing inspiration from software engineering for large logic programs [65]. Moreover, each operator provides abstraction, encapsulating a pattern of interaction or cooperation among the composed systems, and has a well-defined semantics. Such an approach is also extensible, e.g., by introducing new operators to the language, each with its own meaning. Composing context-aware systems (abstracted as triples as shown above) is different from composing individual services, and hence, we defined operators on systems rather than services. We show later that such operators would map to specific ways of invoking services, with each context-aware system having functionality exposed as services.

We can define a set of composition expressions in EBNF:

$$Q ::= R \mid Q + Q$$
$$E ::= Q \mid E \oplus E \mid E \otimes E.$$

Informally, the meaning of the operators are as follows, extending our relation $\vdash$ given above. The union of two context-aware systems used in attempting to recognize C, denoted by $R_1 \oplus R_2 \vdash C$, means that context C is recognized either using R_1 or R_2, succeeding if either succeeds. Note that this can be nondeterministic but one could employ short-circuit evaluation in practice, that is, try R_1 first, and only on failure try with R_2.

The intersection of two context-aware systems used in attempting to recognize C, denoted by $R_1 \otimes R_2 \vdash C$, means that context C is recognized using both R_1 and R_2, succeeding if both succeed.

The tight-union of two context-aware systems used in attempting to recognize C, denoted by $R_1 \otimes R_2 \vdash C$, means that context C

[9] There is much work on service composition such as [55,56].

[10] http://pervasive.semanticweb.org/soupa-2004-06.html.

is recognized using both R_1 and R_2 working in a tightly coupled co-operative manner (as we define below).

Here, we extend these expressions, beyond those in [54] by prefixing the systems with a location designator and incorporating evaluation costs. The new EBNF of expressions is as follows.

$$Q ::= [L :]R \mid Q + Q$$
$$E ::= Q \mid E \oplus E \mid E \otimes E.$$

We assume that L is a label for a possible location of a context-aware system, an element of a predefined set of location labels. The set of location labels would be application dependent; the following discussions make no commitments to this set, maintaining generality. The location prefix "L :" is optional in expressions, and so enclosed in square brackets. By default, location would be the local host on which the expressions are evaluated. For example, the expression $m : R_1 + d : R_2$ could denote that system R_1 resides on the mobile m and R_2 resides on a desktop d. If the expression is evaluated on the mobile device m, then it is equivalent to $R_1 + d : R_2$, leaving out the prefix "m :" for R_1.

The rules in Fig. 1 in Plotkin-style operational semantics [66] define the modified relation $\vdash_\delta$ more precisely, where evaluation returns δ as the evaluation cost. Even if the situation or context is not recognized, the cost is still incurred, i.e. we write $\nvdash_\delta$ to denote that evaluation has failed but cost of δ has been incurred.

In the rule for tight-union (tu), we can observe a tighter coupling of the three components of sensors, interpreter and situation reasoner compared to union, in that a tight-union composition behaves as an integrated system at all three levels (white-box integration), whereas the union employs multiple systems but each as a blackbox. While the rule specifies that the union of the corresponding components (union of sensors, union of interpreters and union of situation reasoners) are employed in recognizing a situation (or context), this happens conceptually, and does not necessarily imply the actual textual integration of software components. Tight-union represents a type of interaction among two systems. The generalized form of tight-union to n-ary is given by the rule (gtu). Note that (gtu) with two operands is equivalent to (tu).

The two rules of union rely on the success of one or the other - operationally, ($union1$) would first be used for evaluations. If rule ($union1$) succeeds, then ($union2$) is not needed. Note that costs are added up for evaluations. The rules ($relocate$) and (gtu, $relocate$) count the cost of using remote system(s) R, which is modeled via a function `cost`. Such a cost may be monetary and/or in terms of network connection time. The rule (gtu, $relocate$) reduces the evaluation to (gtu) after counting the costs (here, we count the costs only once for a consultation, though more fine-grained costing might be possible).

The rules above can be used in a backward-chaining fashion to determine if a particular context or situation currently holds. For example, $(R_1 + R_2) \vdash (W, S)$ determines if situation S holds with respect to the composition (R_1+R_2), by pattern matching. The rules can also be used in as a query to ask what contexts or situations currently hold (or are happening), if S is a variable instead of a given situation. For example, $(R_1 + R_2) \vdash (W, S?)$ asks what situations may be happening with respect to $(R_1 + R_2)$, resulting in one or more possible instantiations for S? (we denote a variable by an identifier post-fixed by a "?").

Note that compositions of systems using the above operators provide as output contexts and situations inferred, and not the actions to be taken. In discussions which follow, we use the symbol $\alpha_{\langle description\rangle}$ to represent systems (or services) that take contexts (e.g., location of a user) and/or situations, as well as possibly other input (e.g., user profile), as input and perform one or more action(s) (e.g., sending advertisements).

We also assume that the above expressions can be embedded within some existing programming language, whether imperative or declarative, and hence, in effect, the context-aware systems can be queried from within the logic of a program. In the discussions which follow, for readability, we will use an embedding within a simple logic programming language with typical Prolog-like constructs, with rules of the form $A : -G$, where A is a Prolog term (which may be a predicate representing a service invocation), I is an identifier which is constant symbol, and G is a goal of the EBNF form

$$G ::= [L :]A \mid E \vdash I[?] \mid G, G$$

where I is an identifier, with an operational semantics that is a simple extension of Prolog's [67], with the additional type of goal $E \vdash I$, or for variables $E \vdash I?$, evaluated using the rules given earlier.

4.3. Example compositions

4.3.1. Example 1

Consider a scenario of a mobile device user in an open air shopping area that aims to utilize a sensor-cloudlet in order to understand the surrounding environment, in terms of air quality, substances that can trigger allergies, crowd and vehicle traffic conditions, and in terms of shops of interest which are in close proximity (say within 20 m), using two types of positioning technologies, via WiFi signal triangulation and GPS. The WiFi positioning mechanism is used just for the time the user is in the area, and in fact, not all the available access points in an area needs to be used for positioning (at least three say if a system such as Ekahau[11] is being used). When the user moves from one area to another a different set of access points might then be employed to provide positioning, i.e. there is a need for resource reservation, utilization and then release, resources used in one area is released and another set of resources are then employed in another area. Moreover, the mobile user may be connected to air quality monitors and crowd, traffic and event monitors in one area, and then the corresponding local monitors of another area. There may be a component downloaded onto the mobile device (perhaps on demand) for discovery and connection to the local sensor-cloudlet for this purpose. Each use of sensors and/or collections of sensors might then incur a charge on the user. The changing resources employed in such a scenario corresponds to what Satyanarayanan in [68] calls *localized scalability* where density of interactions, and so, utilization of local resources, with a place reduces as one moves away from a place and increases with the place one moves into. In other words, the context- or sensor-cloudlet that was formed initially might need to adapt as the user moves, with resources released and other resources bound.

We provide a description of this application as follows. Let R_a^{air}, R_b^{air}, and R_c^{air} represent air quality sensor systems in three adjacent areas a, b and c, respectively, R_a^{crowd}, R_b^{crowd}, and R_c^{crowd} represent corresponding crowd monitoring systems, R_a^{wpos} and R_c^{wpos} represent two positioning systems that can return the location of the user using WiFi triangulation in two areas a and c, assuming there is no WiFi positioning coverage for area b. Now, suppose that the user's mobile device has a local GPS system providing the user's location R^{gpos}, and a system $R^{adsOptIn}$ that infers whether the user is in a suitable situation to receive an ad. Now, we assume that the user is moving around within the adjacent areas a, b and c. We assume that the remote (with respect to the user's mobile device) sensor systems for area a is at location L_a (identified

11 http://www.ekahau.com.

$$\frac{\begin{array}{l}\Big[\text{Where}\quad \Theta = (\Theta_c, \Theta_s), \text{either} \\ (i)\quad [(\{C_1, \ldots, C_m\}, S) \in \Theta_c \\ \text{for some } m, \text{ where for each } C_i,\ i \in \{1, \ldots, m\}, \\ \Pi(\sigma_j(W)) = C_i, \text{ for some } j \text{ and } \sigma_j \in \Sigma], \\ \quad \text{or} \\ (ii)\quad [(\{S_1, \ldots, S_k\}, S) \in \Theta_s \text{for some } k, \\ \text{where for each } S_i,\ i \in \{1, \ldots, k\},\ (\Sigma, \Pi, \Theta) \vdash (W, S_i)]\Big]\end{array}}{(\Sigma, \Pi, \Theta) \vdash (W, S)}\quad (one-system, local)$$

$$\frac{\begin{array}{l}\text{Where } \Theta = (\Theta_c, \Theta_s), \text{ and } \Theta' = (\Theta'_c, \Theta'_s), \text{ either} \\ (i)\quad [(\{C_1, \ldots, C_m\}, S) \in \Theta_c \quad or \quad (\{C_1, \ldots, C_m\}, S) \in \Theta'_c \\ \text{for some } m, \text{ where for each } C_i,\ i \in \{1, \ldots, m\}, \\ \Pi(\sigma(W)) = C_i, \text{ or } \Pi'(\sigma(W)) = C_i, \text{ for some } \sigma \in (\Sigma \cup \Sigma')], \\ \quad \text{or} \\ (ii)\quad [(\{S_1, \ldots, S_k\}, S) \in \Theta_s \quad or \quad (\{S_1, \ldots, S_k\}, S) \in \Theta'_s \text{ for some } k, \\ \text{where for each } S_i,\ i \in \{1, \ldots, k\},\ ((\Sigma, \Pi, \Theta) + (\Sigma', \Pi', \Theta')) \vdash (W, S_i)]\end{array}}{((\Sigma, \Pi, \Theta) + (\Sigma', \Pi', \Theta')) \vdash (W, S)}\quad (tu)$$

$$\frac{(\bigcup_{i=1}^{n} \Sigma_i, \bigcup_{i=1}^{n} \Pi_i, (\bigcup_{i=1}^{n} \Theta_{ic}, \bigcup_{i=1}^{n} \Theta_{is})) \vdash_\delta (W, S)}{Q \vdash (W, S)}\quad (gtu)$$

where $n > 1$ and $Q = (\Sigma_1, \Pi_1, (\Theta_{1c}, \Theta_{1s})) + \ldots + (\Sigma_n, \Pi_n, (\Theta_{nc}, \Theta_{ns}))$

$$\frac{R \ \vdash\ (W, S)\ \wedge\ L\ not\ local\ \wedge\ \delta = \mathtt{cost}(L : R)}{L : R\ \vdash_\delta\ (W, S)}\quad (relocate)$$

$$\frac{R_1 + \ldots + R_n\ \vdash\ (W, S)\ \wedge\ \delta = \sum_{L_i\ not\ local} \mathtt{cost}(L_i : R_i)}{L_1 : R_1 + \ldots + L_n : R_n\ \vdash_\delta\ (W, S)}\quad (gtu, relocate)$$

$$\frac{E\ \vdash_\delta\ (W, S)}{(E \oplus E')\ \vdash_\delta\ (W, S)}\quad (union1)$$

$$\frac{E\ \not\vdash_\delta\ (W, S)\ \ and\ \ E'\ \vdash_{\delta'}\ (W, S)}{(E \oplus E')\ \vdash_{\delta+\delta'}\ (W, S)}\quad (union2)$$

$$\frac{E\ \vdash_\delta\ (W, S)\ \ and\ \ E'\ \vdash_{\delta'}\ (W, S)}{(E \otimes E')\ \vdash_{\delta+\delta'}\ (W, S)}\quad (intersection)$$

Fig. 1. Overview of operators and associated rules.

via a URL, say), for area *b* is at location L_b, and for area *c* is at location L_c. The `good_to_go/1` predicate returns a location that is good to go if air quality is good and there is no crowd:

`good_to_go(`a`) :-`
$L_a : R_a^{air} \vdash good\_air\_quality,\ \ L_a : R_a^{crowd} \vdash no\_crowd.$
`good_to_go(`b`) :-`
$L_b : R_b^{air} \vdash good\_air\_quality,\ \ L_b : R_b^{crowd} \vdash no\_crowd.$
`good_to_go(`c`) :-`
$L_c : R_c^{air} \vdash good\_air\_quality,\ \ L_c : R_c^{crowd} \vdash no\_crowd.$

The following rule returns appropriate ads for the user, assuming a predicate `display/1` that shows ads to the user, α_{ads} is a system (or service) located at $L_{adserver}$ that, given a user's location, returns an *ad* that is most relevant to the user's location, *user_loc*? is a variable which will be instantiated with the user's location, and *ad*? is a variable which will be instantiated with an advertisement:

`see_ad :-`
$R^{adsOptIn} \vdash ok\_for\_ads$, `// check user in situation to receive ads`
$(L_a : R_a^{wpos}\ \oplus\ L_c : R_c^{wpos}\ \oplus\ R^{gpos}) \vdash user\_loc?$, `//get user location`
$L_{adserver} : \alpha_{ads}(user\_loc?, ad?)$, `// retrieve relevant ads`
`display(`$ad?$`).`

We assume such rules being invoked on the user's mobile device, which in turn will query remote context-aware systems. Note that the above rule uses the union composition of the three positioning systems to get the user's location; due to the semantics of union, any one system returning a user location will be acceptable. The use of our intersection operator on multiple positioning systems can be used to model confirmation of locations by different systems, as suggested in [69], where redundant positioning can help to prevent errors. We have assumed, for simplicity, that the systems return location information in the same format, based on some ontology as we noted earlier, and that the argument for α_{ads} accepts this format as well; if this is not the case, an additional intermediate step of format conversion is required.

The identifiers *good_air_quality*, *no_crowd*, and *ok_for_ads* are situations recognized via the respective systems, and can be concepts from some ontology, i.e. there could different systems recognizing the situations specified in an ontology. For example, using the Semantic Web OWL style of encoding ontologies, the identifiers are then concepts within an ontology, and we could reference these ontologies via prefixing the identifiers with the URI identifying the ontology.

Note that the remote context-aware systems and their locations L_a, L_b and L_c can be preprogrammed, or discovered via a system discovery phase (not shown above), and the local (on-mobile) systems used in `good_to_go/1` and `see_ad/0` can also be discovered via a local registry. Details of sensor and system descriptions for purposes of discovery will not be discussed further in this paper.

From the rules, we can see that a set of resources (or context-aware systems) $\{R_a^{air}, R_b^{air}, R_c^{air}, R_a^{crowd}, R_b^{crowd}, R_c^{crowd}\}$ are required for `good_to_go/1` and another set of context-aware systems $\{R^{adsOptIn}, R_a^{wpos}, R_c^{wpos}, R^{gpos}, \alpha_{ads}\}$ for `see_ad/0`. These two sets of resources form the cloudlets for the two predicates, and collectively form the cloudlet for the mobile device supporting these two predicates. These resources can be obtained on-demand during goal evaluation, when a query to a context-aware system is to be issued, or all reserved before the goal evaluation begins, and the cost is computed at run-time using the rules in Fig. 1, by considering each subgoal.

4.3.2. Example 2

In recent times, Bluetooth sensing has been considered for tracking, with many people leaving their Bluetooth enabled mobile phones in discoverable mode [70,71].[12] Consider another example of utilizing Bluetooth sensing infrastructure.

We have two cafes $c1$ and $c2$, each with a device capable of Bluetooth discovery—we call each of these devices a *Bluetooth scanner*, denoted by σ_{c1} and σ_{c1}. Given the limited range of today's Bluetooth of around 10 m, each scanner scans periodically (every 10 s, say) to discover the Bluetooth addresses of devices in the cafe. This is a rough guide as to the number of people in the cafe at different times of the day since not everyone will have a Bluetooth device, and have it on discoverable mode. Suppose each scanner runs daily, over several months, a database of Bluetooth addresses (time-stamped with the time of scan) is created. While the identity of a person is not generally integrated with a particular Bluetooth address so that anonymity can be preserved, the same Bluetooth address might be detected in more than one scan (e.g., when the person is in the cafe for more than 20 s) or the person is a regular visitor to the cafe (so that the person's device is detected often over the span of several months). Let R_{c1}^{bt} and R_{c2}^{bt} be systems of the form $(\{\sigma_{c1}\}, \emptyset, \emptyset)$ and $(\{\sigma_{c2}\}, \emptyset, \emptyset)$, respectively, where each of the Bluetooth scanners, in effect, returns a stream of time-stamped Bluetooth addresses.

Case 1. Now suppose that a user's mobile device has a system with an interpreter that can map his/her friends' Bluetooth addresses into names (viewed as context information indicating that the friend was present) and recognize certain groups of people, denoted by $R^{grp} = (\emptyset, \Pi, (\Theta_c, \emptyset))$, where given a set of friends' Bluetooth addresses Bt and friends' names $NamesPresent$, $\Pi : Bt \to NamesPresent$, and a mapping Θ that maps names of family members present (say, Dav, May, Mary and Tim) to the situation *family_here*:

$$((DavPresent, MayPresent, MaryPresent, , TimPresent), \\ family\_here) \in \Theta$$

and maps the presence of Sam, Zoe, and Joe to the situation *bestfriends_here*:

$$((SamPresent, ZoePresent, JoePresent), bestfriends\_here) \in \Theta.$$

The composition $(L_{c1} : R_{c1}^{bt} + R^{grp})$ forms a context-aware system where the sensors are the Bluetooth scanners from cafe $c1$ (assumed at remote location L_{c1}), and the interpretation and rules to map people present to situations are provided by the system R^{grp} on the mobile device.

So, we can consider a scenario where the mobile user steps into $c1$, and then runs the query $(L_{c1} : R_{c1}^{bt} + R^{grp}) \vdash family\_here$, which will be true if family is present in cafe $c1$, and the query $(L_{c1} : R_{c1}^{bt} + R^{grp}) \vdash group\_here?$, depending on who is present, will yield the variable *group_here?* instantiated accordingly, to *family_here* if all the family members are present, and to *bestfriends_here* if the best friends are present, or none, if the query fails. Now if the user steps into cafe $c2$, the system might adapt to a different composition using the current local resources, i.e., the query $(L_{c2} : R_{c2}^{bt} + R^{grp}) \vdash family\_here$ is used instead, which will be true if family is present in cafe $c2$.

Note the use of the tight-union operator "+" here which would combine the different components from the systems as though there was one system with all the components (as specified in rule (*gtu*) in Fig. 1).

Case 2. Now, a cafe may be small enough so that who is present is easily identified. But considering people present in both the two cafes, the query $(L_{c1} : R_{c1}^{bt} + L_{c2} : R_{c2}^{bt} + R^{grp}) \vdash family\_here$ will determine if family is present in the two cafes, and the query $(L_{c1} : R_{c1}^{bt} + L_{c2} : R_{c2}^{bt} + R^{grp}) \vdash group\_here?$ will return some group (family or best friends) present in the two cafes (if any).

The cafe $c1$ owner, who is, away from his/her cafe, may want to know if his/her cafe is currently crowded, that is, has more than 40 people, say, and has a system $L_{comp} : R^{crowded}$ residing on some compute server, which interprets the readings from the Bluetooth scanner in cafe $c1$ to see if the number of unique addresses detected currently is more than 40, and decides to post the query

$$(L_{c1} : R_{c1}^{bt} + L_{comp} : R^{crowded}) \vdash crowded$$

which results in either true or false, depending on the current number of detected addresses. What we aim to illustrate here is that the output of the Bluetooth scanner in $c1$ is used differently by different people, by composing with different systems, with R^{grp} in Case 1 and $L_{comp} : R^{crowded}$ in Case 2.

Case 3. Rather than a cafe, consider a set of n Bluetooth scanners distributed throughout a shopping mall, denoted by $R_{s1}^{bt}, \ldots, R_{sn}^{bt}$, at remote locations L_{si}, the composition

$$(L_{s1} : R_{s1}^{bt} + \cdots + L_{sn} : R_{sn}^{bt} + R^{grp}) \vdash group\_here?$$

determines if the situation of some group (family or best friends) might be present in the shopping mall. Of course, the cost of these queries can be summed up if the use of the scanners $L_{si} : R_{si}^{bt}$ and $L_{ci} : R_{ci}^{bt}$ are made available as services on a pay-per-use basis.

4.3.3. Example 3

Rather than Bluetooth scanners, other types of scanners or tracking sensors can be packaged as services. For example, in retail stores in a shopping mall, there may be a set of RFID readers that track collections of products, denoted by $L_{ri} : R_{ri}^{rfid}$, and the mobile user has a system that maps RFID ids to products in a shopping wish list $R^{wishlist}$, then, a query to stores 1, 5 and 8 such as

$$(L_{r1} : R_{r1}^{rfid} + L_{r5} : R_{r5}^{rfid} + L_{r8} : R_{r8}^{rfid} + R^{wishlist}) \vdash all\_found$$

can be used to determine if all the products in the wish list are currently found just from the combined set of products tracked (via RFID scanning) in stores 1, 5 and 8. Connections from the

[12] Also see Jabberwocky http://www.urban-atmospheres.net/Jabberwocky/ on Bluetooth discovery of familiar strangers.

user's mobile device to the retailers' remote servers L_{r1}, L_{r5} and L_{r8} (forming a cloudlet around the mobile device) may be made by WiFi or Bluetooth and are not specified here. The mobile user may be somehow charged for interrogating the stores' databases in this way, or retailers may make this service available free to users.

Now, stores 1, 5 and 8 might be the nearest relevant ones to the user. As the user moves, a different set of stores might be queried instead. We have the following rule which queries stores 1, 5 and 8 if the user is in area a and stores 2, 7 and 9 if the user is in area c (using Prolog's if-then-else construct):

```
all_found_area(X) :-
  (L_a : R_a^{wpos} ⊕ L_c : R_c^{wpos}) ⊢ user_loc?, // get user location
  ((user_loc? == a) →
   (L_r1 : R_r1^{rfid} + L_r5 : R_r5^{rfid} + L_r8 : R_r8^{rfid} + R^{wishlist}) ⊢ all_found, X = a
  ; // else user in area c
   (L_r2 : R_r2^{rfid} + L_r7 : R_r7^{rfid} + L_r9 : R_r9^{rfid} + R^{wishlist}) ⊢ all_found, X = c
  ).
```

Hence, even the resources to be used can be adapted to the current context (in this case, location) of the user. We have assumed that the *user_loc?* obtained is an area, but it could be of finer granularity, in which case we assume an `in/2` predicate which can determine if the user is currently within a given area:

```
all_found_area(X) :-
  (L_a : R_a^{wpos} ⊕ L_c : R_c^{wpos}) ⊢ user_loc?,
  (in(user_loc?,a) →
   (L_r1 : R_r1^{rfid} + L_r5 : R_r5^{rfid} + L_r8 : R_r8^{rfid} + R^{wishlist}) ⊢ all_found, X = a
  ;
    (in(user_loc?,c) →
      (L_r2 : R_r2^{rfid} + L_r7 : R_r7^{rfid} + L_r9 : R_r9^{rfid} + R^{wishlist}) ⊢ all_found,
      X = c.
    ).
  ).
```

A more dynamic form of this is to query a "location to stores" database, denoted by $L_{stores} : \alpha_{stores}$ which provide the stores given a location. Now, writing $+[L_{r1} : R_{r1}^{rfid}, \ldots, L_{rn} : R_{rn}^{rfid}]$ to mean $(L_{r1} : R_{r1}^{rfid} + \cdots + L_{rn} : R_{rn}^{rfid})$, we have the rule:

```
all_found_location(X) :-
  (L_a : R_a^{wpos} ⊕ L_c : R_c^{wpos}) ⊢ user_loc?,
 L_stores : α_stores(user_loc?, storelist?), //query for stores at
user loc
  + (storelist?) ⊢ all_found, X = user_loc?.
```

A pay-per-use model makes even better sense here since the actual servers queried depends on context.

4.4. Reliability and cost analysis on compositions

An advantage of our programming language approach to declaratively describing compositions of resources is that static analysis can be facilitated.

Below, we demonstrate this by defining reliability and cost operators on composition expressions, and state observations on the reliability and cost trade-off.

We use a simple reliability model. There are a number of reasons why a query to a system might fail, from connection failure to the system crashing. For simplicity, we denote by p_{rel} a function returning the probability of success of using one or a composition of systems, e.g., $p_{rel}(L : R)$ is the probability that $L : R$ can be successfully queried to return a reliable result.

Given the semantics of the operators, we can define p_{rel} inductively as follows, by following the syntax of expressions given earlier:

$p_{rel}(L : R) = known\_probability\_for\_L : R$

$p_{rel}(Q_1 + Q_2) = p_{rel}(Q_1) \cdot p_{rel}(Q_2)$

$p_{rel}(E_1 \oplus E_2) = 0.5 \cdot p_{rel}(E_1) + 0.5 \cdot p_{rel}(E_2)$

$p_{rel}(E_1 \otimes E_2) = p_{rel}(E_1) \cdot p_{rel}(E_2)$.

Basically, we multiply probabilities when both systems are required to be available for a binary composition to work, and for union, we add the probabilities since we employ an "or" semantics, but weigh the probabilities in that, roughly, the chance of connecting to either one is 0.5 (assuming a random choice). The above can be easily generalized to tight-union of n systems.

So, suppose $p_{rel}(L_a : R_a^{wpos}) = 0.6$ and $p_{rel}(L_c : R_c^{wpos}) = 0.9$, then $p_{rel}(L_a : R_a^{wpos} \oplus L_c : R_c^{wpos}) = 0.5 \cdot 0.6 + 0.5 \cdot 0.9$.

One can estimate the probability of success or determine an upper bound for costs before attempting a given goal.

Several implications of the above are observed:

- tighter cooperation (using tight-union or intersection) among more systems tend to decrease the overall probability of success;
- redundancy tends to increase the probability of success—this means that if two systems are available for the same query, it is better to use a union of the two than just one, but with a possible increase in costs if more than one system in the union needs to be consulted (upon failure);
- given a fixed overall cost C, and a union of a set $\mathbf{X}$ of systems from which to choose from, one can maximize the overall success probability by trying systems from a subset $X \subseteq \mathbf{X}$ such that we maximize:

  $$1/|X| \cdot \sum_{R_i \in X} p_{rel}(R_i)$$

  subject to

  $$\sum_{R_i \in X} cost(R_i) \leq C$$

- given a required fixed overall success probability P_{rel}, and a union of a set $\mathbf{X}$ of systems from which to choose from, one can minimize the overall cost by trying systems from a subset $X \subseteq \mathbf{X}$ such that we minimize:

  $$\sum_{R_i \in X} cost(R_i)$$

  subject to

  $$1/|X| \cdot \sum_{R_i \in X} p_{rel}(R_i) \geq P_{rel}$$

- given a Prolog program with embedded compositions, we can estimate an overall cost and probability of success of a goal as follows (assuming no cycles): for a given goal g, assume r is a rule whose head unifies with g, extract all expressions used in the evaluation of subgoals B in the body of the rule (including subgoals in the other rules which may be used), call this set of expressions E_g. Let $subgoals(g)$ denote the set of subgoals in the body of a rule whose head unifies with g and any other subgoals that might be used in rules used to prove the subgoals in the rule, recursively, i.e.

  $$subgoals(g) = B \cup \left(\bigcup_{b \in B} subgoals(b) \right)$$

  where there exists a rule $g : -B$ (if there is more than one rule, choose the first rule as in normal Prolog evaluation).

  $subgoals(g) = \emptyset$

  if there is no rule whose head unifies with g or g is a fact. Then,

  $E_g = \{E \mid (E \vdash I) \in subgoals(g)\}$.

  Then, the probability of success of a goal (as far it depends on the external resources mentioned in $E_g = \{E_1, \ldots, E_k\}$) is then

  $p_{rel}(E_1) \cdot \ldots \cdot p_{rel}(E_k)$.

  If there is more than one rule whose head unifies with g, the probability increases since there are then more possibilities of success, and so, the above is a minimum probability (with order in which Prolog selects rules to use).

4.5. Meta-level control of evaluation in cloudlets

The embedding and evaluation of compositions within a programming language as we have shown for Prolog enables manipulation of the compositions as first-class objects in programs. This means that compositions can be constructed programmatically and then its evaluation controlled programmatically. Moreover, it is possible to add meta-level control operators on goals of type $E \vdash I$ mentioned in Section 4.2; these control operators are viewed as meta-level as they are at a higher level than the compositions themselves.

To represent the evaluation of a goal $E \vdash I$ and the different states of the computation (as we show later), we introduce the notion of an evaluation object (or *eval object* for short). An eval object encapsulates the evaluation of a goal $E \vdash I$, and is operated on by operators (predicates) given below.

We can call the predicate **create/2** which takes a goal of type $E \vdash I$ and returns an eval object, bound to a given variable, i.e. a call **create** $(E \vdash I, X)$ binds X to an eval object representing the goal $X = E \vdash I$. We can then allow control operators to a created eval object X as follows:

- **start**: which starts the evaluation (according to the operational semantics in Fig. 1) asynchronously on a separate thread, so that **start**$(X, X1)$ takes an eval object X and returns a new version of the eval object which has started $X1$,
- **finish**: which waits for an evaluation to complete and returns the results, so that **finish**(X, R) waits for X to complete and returns the results bound to variable R,
- **stop**: which stops a given evaluation without necessary completing it, so that **stop**(X, F) stops evaluation represented by the eval object bound to X and returns a new eval object bound to F representing a stopped evaluation,
- **pause**: which pauses a given evaluation, so that **pause**$(X, X1)$ takes an eval object X (which is running) and returns a new version of the eval object which has paused evaluation in $X1$,
- **resume**: which resumes a given evaluation, so that **resume**$(X, X1)$ takes an eval object X (which has paused) and returns a new version of the eval object which has resumed evaluation in $X1$.

The above control operators are distinguished, extending the syntax of our Prolog programs:

$G ::= [L :]A \mid E \vdash I[?] \mid G, G \mid M$

$M ::= \textbf{create}(X, Y) \mid \textbf{start}(Y, Z) \mid \textbf{finish}(Y, R) \mid$

$\textbf{stop}(Y, Z) \mid \textbf{pause}(Y, Z) \mid \textbf{resume}(Y, Z)$

where in the EBNF above, X denotes a variable bound to a goal of the form $E \vdash I[?]$, R represents a term representing the result of the evaluation which is either true or false, or a term containing the result $R = I?$, and Y and Z denote variables representing an eval object. For example, consider the following program which is a rewriting of an earlier program, this time using the control operators with the new syntax:

```
see_ad :-
   // check user in situation to receive ads
```
create($R^{adsOptIn} \vdash ok\_for\_ads, X$), **start**($X, X1$), **finish**($X1$,true),
```
   //get user location
```
create($(L_a : R_a^{wpos} \oplus L_c : R_c^{wpos} \oplus R^{gpos}) \vdash user\_loc?, Y$), **start**($Y, Y1$),**finish**($Y1, L$),
$L_{adserver} : \alpha_{ads}(L, ad?)$, `// retrieve relevant ads`
`display(`$ad?$`).`

In the above program, two eval objects were created and used, one for the goal $R^{adsOptIn} \vdash ok\_for\_ads$ (bound to variable X), which evaluated to true or false, and another for the goal $(L_a : R_a^{wpos} \oplus L_c : R_c^{wpos} \oplus R^{gpos}) \vdash user\_loc?$ (bound to Y), which evaluated to the user's location (i.e., $L = user\_loc?$). Note that it is possible to start several eval objects at the same time and wait for them to finish later.

The semantics of the control operator can be given as follows as transitions in a state machine representing the lifecycle of the evaluation of a goal of the form $E \vdash I[?]$ and is given in Fig. 2. The states represent the states of an eval object. Any other application of an operator not detailed in the diagram will have no effect, e.g., applying a stop to an eval object in the stop state has no effect, resuming an eval object that is not in the paused state has no effect, and stopping a finished eval object has no effect (and all these applications are not represented in the diagram). Also, some transitions in the Fig. 2 happen automatically such as "return results" or "reserved resources" (transition happens automatically after resources are successfully acquired).

Evaluation can happen in two modes:

- *conservative*: in conservative evaluation, all the resources are acquired before evaluation begins. For example, in the example above, in the goal to find the user's location, the systems $L_a : R_a^{wpos}, L_c : R_c^{wpos}$, and R^{gpos} are all contacted and resources reserved before evaluation begins. Hence, there is no need to use the transition "request resources" in Fig. 2. This transition is dotted in Fig. 2 since it is not taken in the conservative mode.
- *bold*: in bold evaluation, resources are acquired only when needed. For example, in the goal where the resources $L_a : R_a^{wpos}, L_c : R_c^{wpos}$, and R^{gpos} are mentioned but are unioned, evaluation might proceed with querying $L_a : R_a^{wpos}$ first in which case only this resource is acquired first (and the two resources might never be contacted or acquired if this succeeds according to the semantics of union). Hence, bold evaluation might result in the transition "request resources" in Fig. 2 several times as the process switches from evaluating to reserving resources and back again to evaluating.

A small extension with a new parameter to **create** can be used to specify this mode, e.g. **create**$(E \vdash I, \text{bold}, X)$ specifies a bold evaluation for the given goal.

Note that hanging evaluations are possible—the programmer has to make sure they are completed, by invoking the predicate **finish**, for example. Two other "housekeeping" predicates are defined:

- **status**: which takes an eval object and returns its current state (i.e., returns a string denoting one of the states in Fig. 2), i.e. **status**(X, S) returns $S =$ "*paused*" if X is a valid existing eval object in the paused state, and
- **exception_callback**: which register a callback predicate to be invoked in order to handle exceptions when they are encountered, i.e. **exception_callback**(X, `handle_excp`), where
  `handle_excp(Error) :- display_message(Error).`
  where `handle_excp` always has an error object (containing an error code and related information) as an argument—we do not detail the error object here.

4.6. An abstract architecture and service-based interfaces for realizing composition operators

We describe an abstract architecture for a system which can be used to realize the above composition operators. The idea is to have a front-end component (on a mobile device, say) which queries the remote component systems (in the surrounding infrastructure of stationary computers and perhaps other mobiles) according to the semantics of the composition operators. Our architecture is abstract in that our description does not mandate any particular representation for contexts, situations, or sensor values, nor do we mandate the use of any particular programming platform.

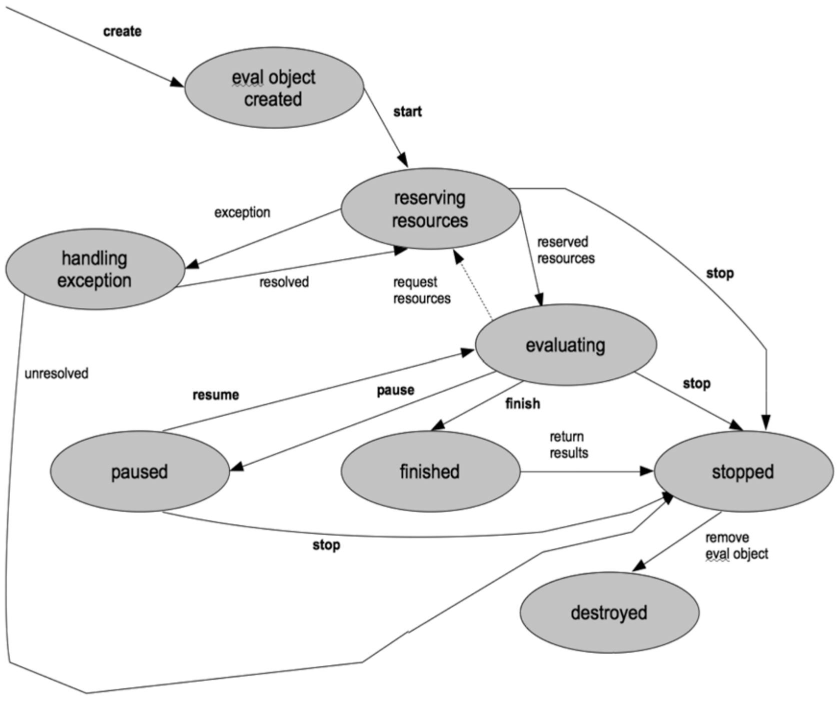

Fig. 2. The states of an eval object, and the corresponding control operators and automatic transitions.

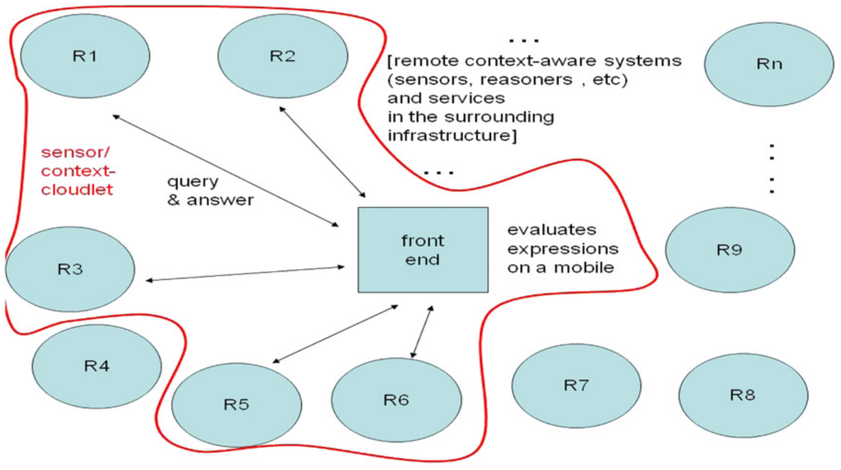
R1
R2
. . .
[remote context-aware systems (sensors, reasoners , etc) and services in the surrounding infrastructure]
Rn
sensor/ context- cloudlet
query & answer
. . .
front end
evaluates expressions on a mobile
R3
R9
R4
R5
R6
R7
R8

Fig. 3. When evaluating a composition on a mobile device, queries might be sent to remote sensors, and services, according to a specified composition expression. A boundary around resources constituting a sensor/context-cloudlet is also illustrated.

We assume an architecture with *n* component systems and a front-end component (typically, on a mobile device). We assume that this is a distributed system with the front-end networked to the remote component systems. From the mobile device, queries are issued to servers that connect to infrastructure sensors. Evaluating a composition then involves sensors on the mobile device and/or sensors from the infrastructure. Fig. 3 illustrates the idea of a mobile device in the midst of local nearby resources and querying these resources as needed as specified in context-aware system compositions.

Computation for evaluating queries can proceed in a backward-chaining manner, resulting in a proof tree if evaluation succeeds. However, computation, can also proceed in a forward-chaining manner where sensor readings are first detected and interpreted, and then context are acquired, and then inferences about situations are made from the contexts acquired and situations recognized. The definitions of the operators allow for such forward chaining reasoning on the rules given earlier.

Each component system in a composition should provide an interface (which we prescribe here) so that the sensor, interpreter and situation reasoner can be accessed by the front-end. The interface can be specified loosely as a set of methods (or services) with arguments and return values as follows (Fig. 3):

```
%%%%% whitebox interface
% retrieving raw sensor data given published sensor
identifiers <Value> reading(<sensor_identifier>)

% interpret(<sensor_data>,<condition>,<Context>)
<Context> fwd_interpret(<sensor_data>,<condition>)
<BooleanValue> bwd_interpret(<Context>)

% situation_reasoner_c(<SetOfContexts>,<Situation>)
<Situation> fwd_situation_reasoner_c(<SetOfContexts>)
<SetOfContexts> bwd_situation_reasoner_c(<Situation>)

% situation_reasoner_s(<SetOfSituations>,<Situation>)
```

```
<Situation> fwd_situation_reasoner_s
(<SetOfSituations>)
<SetOfSituations> bwd_situation_reasoner_s
(<Situation>)


%%%%% blackbox interface
<Situation> fwd_recognize()
<BooleanValue> bwd_recognize(<Situation>)
```

The above methods basically expose the sensors, the interpreter and the situation reasoning capabilities of a context-aware system. Not all context-aware systems will have all the above methods implemented. For example, a system of the structure $(\Sigma, \emptyset, \emptyset)$ with only a non-empty set of sensors Σ without interpreter and situation reasoning capabilities would only have `reading` implemented in its interface.

The first method returns a sensor reading given an identifier for a sensor (realizing $\sigma : \mathbf{W} \rightarrow \mathbf{G}$), assuming that sensor identifiers for a given context-aware system are published and so can be used in arguments.

The next three pairs of methods can be viewed as implementations of reversible predicates.

The `fwd_interpret` method returns context given sensor data, and a condition on the sensor data (realizing $\Pi \subseteq (\mathbf{G} \times \mathbf{C})$), and the `bwd_interpret` method determines if sensors confirm the given context, returning true if so, and false, otherwise. Note that we assumed here that Π is a function rather than a relation, but the method is easily generalizable to return more than one computed context.

The next two methods realize the pair of functions $\Theta = (\Theta_c, \Theta_s)$. The `fwd_situation_reasoner_c` method returns an inferred situation given a set of contexts, and `bwd_situation_reasoner_c` method, given a situation, returns a set of contexts which yields that situation. Note that we assume calls to the reasoners are stateful, that is, for example, if after the first call to `bwd_situation_reasoner_c(S)` with a given situation S, a set of contexts yielding that situation is returned, a subsequent call to the same method might return another set of contexts that also yields that situation. Thereafter, a third call may return another set until an empty set is returned in which no other sets of contexts that yield that situation are found. Similarly, the `fwd_*` methods are similarly stateful. The `fwd_situation_reasoner_s` method returns an inferred situation given a set of situations, and `bwd_situation_reasoner_s` method, given a situation, returns a set of situations which yields that situation. `bwd_situation_reasoner_s(S)` is also assumed to be stateful.

So far, the methods are to allow tight-union composition by exposing the internal components of the system.

The last two methods treat the system as a blackbox. The `fwd_recognize` method asks the system to return a situation it currently recognizes (if any), and the `bwd_recognize` method asks if the system recognizes a given situation. A system implementing only the last two methods cannot participate in tight-union, but will be adequate for union and intersection.

While our abstract architecture does not depend on any particularities of Web services technology, one can imagine these methods being specified as Web services which the front-end can invoke; the prefix L in $L : R$ of a system R can then be a URL.

For instance, consider a query such as $(R_1 + R_2) \vdash (W, S)$ to determine if the situation S is occurring, being evaluated on the front-end component. Each component R_i will provide the above methods. From the definition of tight-union, the sensors, interpreter or situation reasoner of each component are consulted during evaluation, and so, the methods (or services) corresponding to the particular component will be invoked by the front-end. Based on the above interface and rule (*tu*), to evaluate this query, we have the following algorithm (ignoring costs) using backward-chaining reasoning, conveniently expressed in Prolog-like recursive rules, and we assume that calls to `situation_reasoner_s/2` and `situation_reasoner_c/2` map to corresponding `bwd_*` method calls above:

```
evaluate_binary_tight_union(R1,R2,S) :-
     ( R1:situation_reasoner_s(SSet,S),
     ; R2:situation_reasoner_s(SSet,S)
     ),
    foreach T in SSet: evaluate_binary_tight_union(R1,R2,T).
evaluate_binary_tight_union(R1,R2,S) :-
     ( R1:situation_reasoner_c(CSet,S)
     ; R2:situation_reasoner_c(CSet,S)
     ),
     foreach C in CSet: (R1:bwd_interpret(C) ;
     R2:bwd_interpret(C)).
```

For union, based on rules (*union*1) and (*union*2), to evaluate $(R_1 \oplus R_2) \vdash (W, S)$, we would have the following algorithm:

```
evaluate_binary_union(R1,R2,S) :-
    R1:bwd_recognize(S) ; R2:bwd_recognize(S).
```

And for an intersection query $(R_1 \otimes R_2) \vdash (W, S)$, we would have:

```
evaluate_binary_intersection(R1,R2,S) :-
    R1:bwd_recognize(S), R2:bwd_recognize(S).
```

Hence, as long as each context-aware system exposes its components, sensors, interpreters or situation inference rules, they can be interrogated for values. The way in which two systems in a composition are queried are determined by the semantics of the operator. Each operator, in effect, captures a way in which the two systems "cooperate" in answering queries.

5. Conclusion and future directions

This paper has proposed the concept of sensor-cloudlets and context-cloudlets, for providing sensor information and context information on an on-demand (pay-per-use) basis. Because sensor-cloudlets and context-cloudlets essentially involves the composition of context-aware systems (an abstraction over sensors, context interpreters and situation reasoning engines) within applications, we provided a set of operators for this purpose. We started with the basic intuitive operators of union and intersection, which simply follows from "and" and "or" of systems. But we also demonstrated an operator "tight-union" which allows a tighter integration of systems in evaluating queries, which has a declarative semantics in our abstract model, which is the union of the respective internal components.

We do not aim to be exhaustive in the set of operators that one can define, but aimed to provide a starting point for operators that might be employed in composing resources for cloudlets. Our examples also serve to illustrate what can be achieved with only a small set of operators.

A prototype implementation of our system, with the language as operators, is being implemented. We also noted many questions and challenges that are not addressed in this paper relating to sensor-cloudlets and context-cloudlets, summarized below:

- discovery of resources to fulfill a user's need, expressed in a program with embedded compositions: this will involve appropriate (preferably standardized) descriptions of resources and algorithms to match such descriptions with the user's needs; algorithms to bind available sensors to that specified in a composition is required;
- managed execution of evaluations when compositions are being used, and user program's are executed: resources discovered must be reserved and allocated for the user for a given period, and during execution, unexpected failures and

metering of resource usage must be included; on the resource provider side, scheduling of resources to meet multiple user needs is required;

- there is a need for trust and security models, in that handling of context information of users must be privacy-respecting and context information provided must be reliable and trustworthy;
- creation of a database of useful user programs, and useful compositions, so that users who are not technical can immediately benefit;
- wrapping of provider resources according to a resource description standard, in order to be made available for user programs and compositions;
- billing and service level agreements framework, that is efficient enough for transient usage (e.g., some resources may only be used for several minutes while the user is in the area); our cost model integrated into our operational semantics rules is coarse-grained especially for tight-union—finer-grained cost models are needed that could even bill for each individual service invocation;
- the need for a regulatory framework for the emergence of markets involving multiple resource providers;
- ontology support for interoperability when sharing context and situation knowledge; for example, given a composition of resources, different resources might come from different providers, and as seen in the examples earlier, the context information from one provider might be used as input to services or as premises in the rules of a context-aware system belonging to a different provider, or context information from different providers need to be combined; we envision that context ontologies that have already been developed can be employed, and new ontologies developed as new types of context information and situation knowledge are provided to users;
- mechanisms for adaptations of compositions can be investigated—for example, when one resource is lost or becomes out of range, a replacement can be found, or a better alternative for a resource (e.g., a better positioning technology is found when the user walks into a new area) might be employed; there is a need to release resources and bind to newly discovered resources, with a composition expression acting as a specification of what resources are needed and how they are obtained (e.g., rather than having a composition bind to specific sensors and systems at program-specified locations, a composition could bind to resources discovered at evaluation time as needed). Indeed dynamic context-aware resource binding for mobile applications between clients and services has been considered previously (e.g., [72]), though not in this context.

References

[1] I. Akyildiz, S. Weilian, Y. Sankarasubramaniam, E. Cayirci, A survey on sensor networks, IEEE Communications Magazine 40 (8) (2002) 102–114. doi:10.1109/LCN.2005.123.

[2] F. Zhao, L. Guibas, Wireless Sensor Networks: An Information Processing Approach, Morgan Kaufmann, 2004.

[3] K. Martinez, J. Hart, R. Ong, Environmental sensor networks, IEEE Computer 37 (8) (2004) 50–56.

[4] W.S. Conner, J. Chhabra, M. Yarvis, L. Krishnamurthy, Experimental evaluation of topology control and synchronization for in-building sensor network applications, Mobile Networks and Applications 10 (4) (2005) 545–562. doi:10.1145/1160162.1160177.

[5] G. Borriello, K. Farkas, F. Reynolds, F. Zhao, Special issue on building a sensor-rich World, IEEE Pervasive 6 (2) (2007).

[6] H.B. Lim, Y.M. Teo, P. Mukherjee, V.T. Lam, W.F. Wong, S. See, Sensor grid: integration of wireless sensor networks and the grid, in: Proceedings of the Annual IEEE Conference on Local Computer Networks, IEEE Computer Society, Los Alamitos, CA, USA, 2005, pp. 91–99. doi:10.1109/LCN.2005.123.

[7] M. Balazinska, A. Deshpande, M.J. Franklin, P.B. Gibbons, J. Gray, M. Hansen, M. Liebhold, S. Nath, A. Szalay, V. Tao, Data management in the worldwide sensor web, IEEE Pervasive Computing 6 (2007) 30–40. doi:10.1109/MPRV.2007.27.

[8] A. Kansal, S. Nath, J. Liu, F. Zhao, Senseweb: an infrastructure for shared sensing, IEEE Multimedia 14 (4) (2007) 8–13.

[9] A.K. Dey, Understanding and using context, Personal and Ubiquitous Computing 5 (1) (2001) 4–7. doi:10.1007/s007790170019.

[10] J. Ye, L. Coyle, S. Dobson, P. Nixon, Ontology-based models in pervasive computing systems, Knowledge Engineering Review 22 (4) (2007) 315–347. doi:10.1017/S0269888907001208.

[11] H. Chen, F. Perich, T. Finin, A. Joshi, Soupa: standard ontology for ubiquitous and pervasive applications, in: Proceedings of the International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004, pp. 258–267.

[12] S.W. Loke, Representing and reasoning with situations for context-aware pervasive computing: a logic programming perspective, Knowledge Engineering Review 19 (3) (2004) 213–233. doi:10.1017/S0269888905000263.

[13] G. Judd, P. Steenkiste, Providing contextual information to pervasive computing applications, in: PERCOM'03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, IEEE Computer Society, Washington, DC, USA, 2003, p. 133.

[14] C. Endres, A. Butz, A. MacWilliams, A survey of software infrastructures and frameworks for ubiquitous computing, Mobile Information Systems 1 (1) (2005) 41–80.

[15] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, D. Riboni, A survey of context modelling and reasoning techniques, Pervasive and Mobile Computing (2010). doi:10.1016/j.pmcj.2009.06.002. URL: http://dx.doi.org/10.1016/j.pmcj.2009.06.002.

[16] C. Anagnostopoulos, S. Hadjiefthymiades, Enhancing situation-aware systems through imprecise reasoning, IEEE Transactions on Mobile Computing 7 (10) (2008) 1153–1168. doi:10.1109/TMC.2008.34.

[17] J. Ye, L. Coyle, S.A. Dobson, P. Nixon, Representing and manipulating situation hierarchies using situation lattices, Revue d'Intelligence Artificielle 22 (5) (2008) 647–667.

[18] S.W. Loke, On representing situations for context-aware pervasive computing: six ways to tell if you are in a meeting, in: Pervasive Computing and Communications Workshops, IEEE International Conference on, IEEE Computer Society, Los Alamitos, CA, USA, 2006, pp. 35–39. doi:10.1109/PERCOMW.2006.102.

[19] M. Raento, A. Oulasvirta, R. Petit, H. Toivonen, Contextphone: a prototyping platform for context-aware mobile applications, IEEE Pervasive Computing 4 (2) (2005) 51–59. doi:10.1109/MPRV.2005.29.

[20] J. Kukkonen, E. Lagerspetz, P. Nurmi, M. Andersson, Betelgeuse: a platform for gathering and processing situational data, IEEE Pervasive Computing 8 (2) (2009) 49–56. doi:10.1109/MPRV.2009.23.

[21] C. Julien, G.-C. Roman, Egospaces: facilitating rapid development of context-aware mobile applications, IEEE Transactions on Software Engineering 32 (2006) 281–298. doi:10.1109/TSE.2006.47.

[22] M. Mamei, F. Zambonelli, Programming pervasive and mobile computing applications: the tota approach, ACM Transactions on Software Engineering and Methodology 18 (4) (2009) 1–56. doi:10.1145/1538942.1538945.

[23] L.M. Vaquero, L. Rodero-Merino, J. Caceres, M. Lindner, A break in the clouds: towards a cloud definition, SIGCOMM Computer Communication Review 39 (1) (2009) 50–55. doi:10.1145/1496091.1496100.

[24] D. Linthicum, Cloud Computing and SOA Convergence in Your Enterprise, Addison-Wesley, 2010.

[25] J. Varia, Best practices in architecting cloud applications in the AWS cloud, in: R. Buyya, J. Broberg, A. Goscinski (Eds.), Cloud Computing: Principles and Paradigms, John Wiley & Sons, Inc., 2011, pp. 457–490.

[26] C. Vecchiola, X. Chu, M. Mattess, R. Buyya, Aneka—integration of private and public clouds, in: R. Buyya, J. Broberg, A. Goscinski (Eds.), Cloud Computing: Principles and Paradigms, John Wiley & Sons, Inc., 2011, pp. 249–274.

[27] M.M. Hassan, B. Song, E.-N. Huh, A framework of sensor-cloud integration opportunities and challenges, in: ICUIMC'09: Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication, ACM, New York, NY, USA, 2009, pp. 618–626. doi:10.1145/1516241.1516350.

[28] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for vm-based cloudlets in mobile computing, IEEE Pervasive Computing 8 (4) (2009) 14–23. doi:10.1109/MPRV.2009.82.

[29] C.-K. Tham, R. Buyya, Sensor grid: integrating sensor networks and grid computing, CSI Communications 29 (1) (2005) 24–29.

[30] H.B. Lim, Y.M. Teo, P. Mukherjee, V.T. Lam, W.F. Wong, S. See, Sensor grid: integration of wireless sensor networks and the grid, in: Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary, LCN'05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 91–99. doi:10.1109/LCN.2005.123. URL: http://dx.doi.org/10.1109/LCN.2005.123.

[31] T. Kobialka, R. Buyya, C. Leckie, R. Kotagiri, A sensor web middleware with stateful services for heterogeneous sensor networks, in: 2007 3rd International Conference on Intelligent Sensors, Sensor Networks and Information, IEEE, 2007, pp. 491–496. doi:10.1109/ISSNIP.2007.4496892. URL: http://dx.doi.org/10.1109/ISSNIP.2007.4496892.

[32] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems 25 (2009) 599–616. doi:10.1016/j.future.2008.12.001. URL: http://dl.acm.org/citation.cfm?id=1528937.1529211.

[33] S. Pandey, W. Voorsluys, S. Niu, A. Khandoker, R. Buyya, An autonomic cloud environment for hosting ECG data analysis services, Future Generation Computer Systems 28 (1) (2012) 147–154. URL: http://www.sciencedirect.com/science/article/pii/S0167739X11000732.

[34] E. Miluzzo, N.D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S.B. Eisenman, X. Zheng, A.T. Campbell, Sensing meets mobile social networks:

the design, implementation and evaluation of the cenceme application, in: SenSys'08: Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems, ACM, New York, NY, USA, 2008, pp. 337–350. doi:10.1145/1460412.1460445.

[35] O. Riva, Contory: a middleware for the provisioning of context information on smart phones, in: Middleware'06: Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware, Springer-Verlag New York, Inc., New York, NY, USA, 2006, pp. 219–239.

[36] S.B. Eisenman, E. Miluzzo, N.D. Lane, R.A. Peterson, G.-S. Ahn, A.T. Campbell, The bikenet mobile sensing system for cyclist experience mapping, in: SenSys'07: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems, ACM, New York, NY, USA, 2007, pp. 87–101. doi:10.1145/1322263.1322273.

[37] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, N. Triandopoulos, Anonysense: privacy-aware people-centric sensing, in: MobiSys'08: Proceeding of the 6th International Conference on Mobile Systems, Applications, and Services, ACM, New York, NY, USA, 2008, pp. 211–224. doi:10.1145/1378600.1378624.

[38] S. Neely, M. Stabeler, P. Nixon, Sensormash: exploring system fidelity through sensor mashup, in: R. Mayrhofer, A. Quigley, J. Kay, G. Kortuem (Eds.), Adjunct Proceedings of the Sixth International Conference on Pervasive Computing, 2008, pp. 83–86.

[39] L. Costabello, O.R. Rocha, L.W. Goix, Sharing mobile user experiences with context-based mashups, in: Mobiquitous'08: Proceedings of the 5th Annual International Conference on Mobile and Ubiquitous Systems, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2008, pp. 1–4. doi:10.4108/ICST.MOBIQUITOUS2008.3978.

[40] A. Brodt, D. Nicklas, S. Sathish, B. Mitschang, Context-aware mashups for mobile devices, in: WISE'08: Proceedings of the 9th International Conference on Web Information Systems Engineering, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 280–291. doi:10.1007/978-3-540-85481-4_22.

[41] S.W. Loke, Towards declarative programming for sensor-based situation-aware applications: the logiccap approach, in: Proceedings of the International Conference on Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP, 2008, pp. 447–452.

[42] T. Abdelzaher, Y. Anokwa, P. Boda, J. Burke, D. Estrin, L. Guibas, A. Kansal, S. Madden, J. Reich, Mobiscopes for human spaces, Pervasive Computing, IEEE 6 (2) (2007) 20–29. doi:10.1109/MPRV.2007.38.

[43] Y. Ma, M. Ghanem, Y. Guo, M. Richards, Air pollution monitoring and mining based on sensor grid in London, Sensors: Special Issue on Urban Environmental Monitoring 8 (2008) 3601–3623. URL: http://pubs.doc.ic.ac.uk/Sensors/.

[44] E. Paulos, R. Honicky, B. Hooker, Citizen science: enabling participatory urbanism, in: M. Foth (Ed.), Handbook of Research on Urban Informatics: The Practice and Promise of the Real-Time City, IGI Global, 2008.

[45] O. Riva, C. Borcea, The urbanet revolution: sensor power to the people! IEEE Pervasive Computing 6 (2) (2007) 41–49. doi:10.1109/MPRV.2007.46.

[46] D. Cuff, M. Hansen, J. Kang, Urban sensing: out of the woods, Communications of the ACM 51 (3) (2008) 24–33. doi:10.1145/1325555.1325562.

[47] A.T. Campbell, S.B. Eisenman, N.D. Lane, E. Miluzzo, R.A. Peterson, People-centric urban sensing, in: WICON'06: Proceedings of the 2nd Annual International Workshop on Wireless Internet, ACM, New York, NY, USA, 2006, p. 18. doi:10.1145/1234161.1234179.

[48] M. Gerla, Vehicular urban sensing: efficiency and privacy, in: MSWiM'08: Proceedings of the 11th International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems, ACM, New York, NY, USA, 2008, doi:10.1145/1454503.1454504. 1–1.

[49] B. Hooker, W. Gaver, A. Steed, J. Bowers, The pollution e-sign, in: Workshop on Ubiquitous Sustainability, UbiComp, 2007.

[50] S. Kim, E. Paulos, inAir: measuring and visualizing indoor air quality, in: Ubicomp'09: Proceedings of the 11th International Conference on Ubiquitous Computing, ACM, New York, NY, USA, 2009, pp. 81–84. doi:10.1145/1620545.1620557.

[51] A.F. gen. Schieck, A. Penn, E. O'Neill, Mapping, sensing and visualising the digital co-presence in the public arena, in: In 9th International Conference on Design and Decision Support Systems in Architecture and Urban Planning, 2008, pp. 38–58.

[52] J. Perkio, V. Tuulos, M. Hermersdorf, H. Nyholm, J. Salminen, H. Tirri, Utilizing rich Bluetooth environments for identity prediction and exploring social networks as techniques for ubiquitous computing, in: WI'06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence, IEEE Computer Society, Washington, DC, USA, 2006, pp. 137–144. doi:10.1109/WI.2006.185.

[53] H. Lu, N.D. Lane, S.B. Eisenman, A.T. Campbell, Bubble-sensing: Binding sensing tasks to the physical world, Pervasive and Mobile Computing 6 (1) (2010) 58–71. URL: http://dl.acm.org/citation.cfm?id=1716217.

[54] S.W. Loke, Incremental awareness and compositionality: a design philosophy for context-aware pervasive systems, Pervasive and Mobile Computing 6 (2) (2010) 239–253. doi:10.1016/j.pmcj.2009.03.004. URL: http://dx.doi.org/10.1016/j.pmcj.2009.03.004.

[55] J. Rao, X. Su, A survey of automated web service composition methods, in: SWSWPC, 2004, pp. 43–54.

[56] Q.Z. Sheng, B. Benatallah, Z. Maamar, A.H.H. Ngu, Configurable composition and adaptive provisioning of web services, IEEE Transactions on Services Computing 2 (1) (2009) 34–49.

[57] X. Wang, D. Zhang, T. Gu, H. Pung, Ontology based context modeling and reasoning using OWL, in: PERCOMW'04: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, IEEE Computer Society, Washington, DC, USA, 2004, pp. 18–22.

[58] N. Baumgartner, W. Retschitzegger, A Survey of Upper Ontologies for Situation Awareness, in: Proceedings of Knowledge Sharing and Collaborative Engineering, ACTA Press, 2006.

[59] F. Ay, Context modeling and reasoning using ontologies. Available at: http://www.aywa.de/cmaruo/cmaruo.pdf.

[60] C.B. Anagnostopoulos, Y. Ntarladimas, S. Hadjiefthymiades, Situational computing: an innovative architecture with imprecise reasoning, Journal of Systems and Software 80 (12) (2007) 1993–2014.

[61] S. Yau, J. Liu, Hierarchical situation modeling and reasoning for pervasive computing, in: SEUS-WCCIA'06: Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance, SEUS-WCCIA'06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 5–10.

[62] P. Costa, G. Guizzardi, J. Almeida, L. Pires, M. van Sinderen, Situations in conceptual modeling of context, in: EDOCW'06: Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops, IEEE Computer Society, Washington, DC, USA, 2006, Available at: http://www.loa-cnr.it/Guizzardi/DockhornCosta-et-al-VORTE06_final.pdf.

[63] K. Henricksen, J. Indulska, Developing context-aware pervasive computing applications: models and approach, Journal of Pervasive and Mobile Computing 2 (1) (2006) 37–64.

[64] A. Ranganathan, R. Campbell, An infrastructure for context-awareness based on first-order logic, Personal and Ubiquitous Computing 7 (6) (2003) 353–364.

[65] A. Brogi, P. Mancarella, D. Pedreschi, F. Turini, Modular logic programming, ACM Transactions on Programming Languages and Systems 16 (4) (1994) 1361–1398. doi:10.1145/183432.183528.

[66] G. Plotkin, A structural approach to operational semantics, Journal of Logic and Algebraic Programming 60–61 (2004) 17–139. Available at: http://homepages.inf.ed.ac.uk/gdp/publications/sos_jlap.pdf.

[67] L. Sterling, E. Shapiro, The Art of Prolog, MIT Press, Cambridge, MA, 1986.

[68] M. Satyanarayanan, Pervasive computing: vision and challenges, IEEE Personal Communications 8 (4) (2001) 10–17 [see also IEEE Wireless Communications].

[69] T. Pfeifer, Redundant positioning architecture, in: Wireless Sensor Networks and Applications—Proceedings of the Dagstuhl Seminar 04122, Computer Communications 28 (13) (2005) 1575–1585. doi:10.1016/j.comcom.2004.12.042. URL: http://www.sciencedirect.com/science/article/B6TYP-4G4WYH1-1/2/acf2af0c25405340067ff1891de54a0f.

[70] T. Kindberg, T. Jones, Merolyn the phone: a study of Bluetooth naming practices, in: J. Krumm, G.D. Abowd, A. Seneviratne, T. Strang (Eds.), Ubicomp, in: LNCS, vol. 4717, Springer, 2007, pp. 318–335.

[71] T. Nguyen, S.W. Loke, T. Torabi, H. Lu, Placesense: a tool for sensing communities, in: 4th International Symposium on Wireless Pervasive Computing, 2009. ISWPC 2009, 2009, pp. 1–5. doi:10.1109/ISWPC.2009.4800601.

[72] P. Bellavista, A. Corradi, R. Montanari, C. Stefanelli, Dynamic binding in mobile applications: a middleware approach, IEEE Internet Computing 7 (2) (2003) 34–42. doi:10.1109/MIC.2003.1189187.

Seng W. Loke is Reader and Associate Professor at the Department of Computer Science and Computer Engineering in La Trobe University. He leads the Pervasive Computing Group at La Trobe, and has authored 'Context-Aware Pervasive Systems: Architectures for a New Breed of Applications' published by Auerbach (CRC Press), Dec 2006. He has (co-)authored more than 210 research publications, with numerous works on context-aware computing, and mobile and pervasive computing. His research has been published in journals such as IEEE Pervasive, Knowledge Engineering Review, Elsevier's Pervasive and Mobile Computing Journal, IEEE Transactions on SMC, MONET, Journal of Systems and Software, and Theory and Practice of Logic Programming.