CAwbWeb: Towards a Standardized Programming Framework to Enable a Context-Aware Web

Aaron Beach, Mike Gartrell, Richard Han, Shivakant Mishra

University of Colorado at Boulder

Contact: {aaron.beach, mike.gartrell}@colorado.edu

Department of Computer Science

University of Colorado at Boulder

Technical Report CU-CS-1063-10

March 2010

CAwbWeb: Towards a Standardized Programming Framework to Enable a Context-Aware Web

Aaron Beach, Mike Gartrell, Richard Han, Shivakant Mishra
University of Colorado at Boulder
{aaron.beach, mike.gartrell}@colorado.edu

ABSTRACT

This paper presents a vision of mobile-cloud computing systems which looks beyond simply moving data processing from mobile to cloud computing systems and considers a larger vision in which services within the cloud are aggregated and automatically integrated in real-time, based on context, to satisfy the intentions of mobile computing applications. We suggest that research and development of context-aware mobile-cloud systems adopt an appropriate "separation of concerns" in which application intentions, context, action, and actuation are isolated through a framework of software services and standards we call CAwbWeb (Context-Aware Web). These software services and standards allow for many different mobile-cloud computing problems to be abstracted and solved generally.

1. INTRODUCTION

This paper suggests a new framework to support the vision of context-aware mobile cloud computing in which mobile-aware services running on the cloud are automatically integrated with local services offered by the surrounding physical environment to achieve fully context-aware smart environments. The proposed Context-Aware Web (CAwbWeb) framework is motivated by a number of new trends, namely the rise of "app" mobile smartphones, the growth of mobile social networks, the increasing popularity of cloud computing, and the evolution of Web 2.0 Web services. We believe it is important and timely to develop a unifying framework that spans and harnesses the capabilities offered by each of these individual trends to collectively empower modern context-aware computing.

The rise of the "app phone": The past few years have seen explosive growth in mobile smartphones, and in particular "app phones" such as the iPhone, Droid, Nexus One, Palm Pre, and Windows Phone. These phones, mated to app stores, have simplified loading of applications and resulted in commonplace use of third-party applications. This trend toward "app phone" usage has accelerated the development of contextual mobile computing services. App phone users may browse near their location for friends (Latititude), events (Yelp), tweets (Twitter), or buzzes (Google Buzz). Also, apps allow people to connect virtual information with their surroundings through augmented reality apps such as virtual spray-painting (SimSpray) or a virtual wiki for the physical world (wikitude). Location-aware toolkits and Web services (ARToolKit, SimpleGEO) are emerging to support such applications. The mobile phone and particularly the "app phone" is becoming an important window into the world of Web services.

Mobile social networks: Mobile social networks combine location awareness from mobile phones with context awareness from social networks, e.g. friendship relationships and personal profile preferences, to offer novel new context-aware services, such as viewing my friends nearby, or showing services that I might be interested in that are nearby. Commercial mobile social networks such as Loopt, Brightkite, and Foursquare have taken off - and over 100 million of the 400+ million Facebook users now access Facebook through their mobile devices. Many online data sources such as Facebook now have Web APIs which open up their information to mobile Web applications. Research projects in mobile social networks have considered how mining social information about proximate individuals might enhance or drive their interaction [11, 16, 6]. Privacy concerns with presence sharing in mobile social networks have also been investigated [8].

Clouds on the horizon: Many of today's mobile computing applications are distributed, with a mobile component being supported by a remote service component often being hosted on computing clouds such as Amazon EC2, Microsoft Azure, Google AppEngine, RackSpace, etc. This model allows the services to scale with demand and allows designers to deploy ideas and services with little overhead cost. However, these mobile services are generally one-off stand-alone systems targeted at a particular phone application and do not generally interact with other services. While computing clouds and app phones support rapid deployment of mobile computing services, there is still no general system for integrating these services or for automatic discovery and on-the-fly use by phone applications. We believe

that it is important and timely to develop a unifying standardized framework that can harness the power of the cloud for mobile applications, so that mobile applications can automatically lookup and integrate these services to achieve their goals in real-time. Developers of mobile computing services should not have to worry about writing custom applications for each of their services. Instead, they should be able to simply describe them and leverage the power of Web services offered by the cloud to implement their intentions.

Context-aware systems and frameworks: The dream of ubiquitous computing is to interact with a context-aware smart space that is richly embedded with computational elements both for sensing and actuation. There has been a rich exploration of context-aware systems and frameworks [5, 14, 7, 15, 12, 17, 9]. Much of this work occurred prior to the advent of social networks, app phones, and cloud computing. Our work does not propose a new context-aware system - but rather a framework to connect context-aware services from the cloud to the mobile application in a general way. Our framework seeks to use standardized Web services technologies and protocols to enable uniform and interoperable interfaces between context-aware systems, so that context-aware systems can scale up and effectively use cloud-based services.

The CAwbWeb Vision: We seek to achieve comprehensive context-aware computing in local physical smart spaces by designing a unifying framework that integrates mobile app phones with cloud computing services, thus enabling automated development and deployment of context-aware applications and services. The overarching challenges that we've identified in our design involve how mobile applications describe, find, and integrate appropriate cloud services. We envision a usage model in which environments - people, places, and things - "deploy" themselves and their services to the Internet as cloud computing services, forming "clouds" of data and services. These clouds must then be organized so that they may be browsed by the user and his mobile phone. The user's phone will have a client or browser that knows how to search these clouds for appropriate data and services. Once found, each user's intentions are mapped to and activate an appropriate set of data and services in order to realize the user's intentions in the local context-aware smart space. Furthermore, this mobile-cloud interaction adheres to the appropriate personal and contextual rules governing the usage of the particular services and data (e.g., privacy, legal, temporal, or spatial limitations).

For example, our framework should support the following context-aware scenario. Suppose a customer, Mark, enters a video store. As Mark approaches the new releases aisle, a mobile-cloud service senses Mark's presence and plays a movie trailer most suited to Mark's preferences on the large-screen display that is closest and most visible to Mark. Mark finds this trailer interesting, so he grabs the DVD for this movie and proceeds to the checkout counter. As Mark is waiting in line, a fire occurs in another part of the store. The fire suppression system is also context-aware, and understands that Mark has photosensitive epilepsy, and therefore activates the fire alarm and sprinklers, but does not activate the strobe lights. Mark is instructed on how to exit the building and the appropriate city services are immediately notified. In this example, the framework should support announcement of Mark's presence, discovery of available services in this context, and automatic interconnection and use of the appropriate Web services to execute the intended application, e.g. either context-aware video presentation or context-aware emergency services. We will revisit this story later to detail how our framework supports such a scenario.

2. OVERVIEW OF THE CAWBWEB FRAMEWORK

The design of new types of mobile-cloud context-aware systems requires a new and appropriate separation of concerns. Dijkstra pointed out, apropos "intelligent thought", that an appropriate separation of concerns is the only (or primary) technique for effectively ordering ones thoughts about a complex problem [10]. In order to address the difficulties of developing mobile-cloud context-aware applications, we seek to provide a framework that adheres to a "separation of concerns" in order to simplify the task of programming for the developer, while also enabling the evolution of sophisticated context-aware mobile-cloud systems. Our proposed programming framework identifies a set of four major problem spaces (or "concerns") that can be addressed independently. These four concerns are:

- **"Intention"** Applications should specify what they intend to do.
- **"Context"** Applications should describe in what context their intentions should be executed.
- **"Action"** Identifying actions within a context that appropriately satisfy an intention.
- **"Actuation"** Synthesizing these actions into a composition of mobile-cloud services or actuators to execute an intention.

This approach achieves a clean separation of concerns, wherein we identify components that have clearly decoupled duties and interfaces, enabling each component to be developed independently. The only responsibilities of the mobile application would be to clearly and simply describe its context (location, time, temperature, etc.) and specify what it is it wants to do in

that context (the intention). Describing context could be implemented as a general service for each device that periodically reports sensor and location data to a context management service. This presents an optimal situation where the developer need only worry about appropriately expressing the intention of the application. The application developer no longer has to program for each Web service or mobile-cloud system, nor discover these services, nor manually integrate them. The problem of composing appropriate Web-services from those running in the cloud is abstracted and can be solved using many different systems. The task of composing Web services can further separated into two phases: service discovery or lookup, and integration of those services.

The essential components of our system revealed by this separation of concerns, and how they fit together, are shown in Figure 1. A context-aware mobile application developer first specifies his Intention using a high level language described later, i.e. what task he wants to accomplish, such as playing a video. Using another language, the developer specifies in what Context to execute the Intention, e.g. what location or time. The application then passes both the Intention and the Context to a generic Contextual Lookup Service (CLS). Web services in the cloud are registered with the CLS, which can find and return the set of possible Web services to satisfy the Intention in this Context, e.g. video monitors in this location. We term this set of possible Web services supported by the specified context as Actions, or more generally a Possible Action Set (PAS), since they could result in actions like playing a video. Next, the application will send the Intention combined with the Actions to a *compiler* which integrates actions from the different Web services along with their dependencies. This Context-Aware Intention Compiler (CAIC) generates a program or script capable of executing the Intention in that Context, which is then run on an Interpreter to effectuate the context-aware result.

By separating intention, context, action, and actuation, we provide a general-purpose framework for developing mobile-cloud applications that run reusable context-aware services. The CLS allows a mobile application to submit an intention and a description of the context for that intention to a Web service without specific knowledge of how that intention will be implemented by the system. This is a key innovation that frees the application developer to develop context-aware mobile-cloud applications without having to worry about the low-level mechanics. The CAIC serves the critical role of isolating the mobile client from these low-level details by incorporating sufficient intelligence to generate an actuation program that contains all of the details about how to assemble and compose Web services to accomplish the intentions of the application.

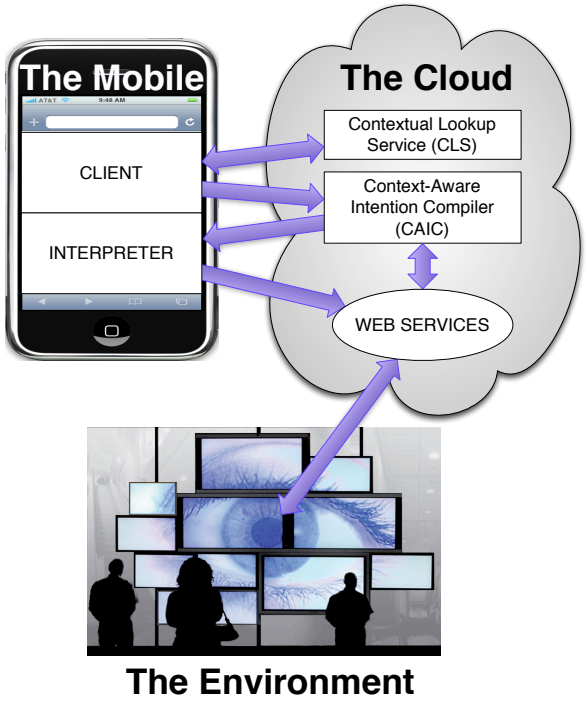

Figure 1: Overview of CAwbWeb Framework.

The entire framework can be summarized as including six major abstractions or software components as depicted in figure 1, they are: Mobile Application, Mobile Client, Contextual Lookup Service (CLS), Context-Aware Intention Compiler (CAIC), Actuation Program Interpreter, and the supporting Mobile-Cloud Web Services. The framework also includes five classes of document or language standards which provide powerful abstraction layers between the components. The five standards are discussed in section 4 and include: Context Description Language (CDL), Intention Specification Language (ISL), Action Specification Language (ASL), Actuation Instruction Language (AIL), Web Services Description Language (WSDL).

3. SOFTWARE COMPONENTS

This section describes the proposed software components to support the CAwbWeb framework. This section also discusses how each standard and language described in section 4 is used by the components, further motivating the particular abstractions that we have chosen. Furthermore, the discussion of each component explicitly states the related research challenges.

3.1 Mobile Client

This paper proposes the design of a mobile client that would provide client support for applications designed on the CAwbWeb framework. The client would provide an API for use by mobile-cloud applications. At mini-

mum, an API would need to support a request method allowing the application to submit context descriptions and intention objects. However, a client could also support a higher level interface, such as automatic context inference or a textual/graphical intention interface, allowing the user to express their intentions naturally and freely.

Once the client has an Intention Object and Context Description it must know of a Contextual Lookup Service (CLS) to which it can submit a context request. The addresses of contextual lookup services must be stored by the client in much the same way that the IP address of Domain Name Servers are stored by hosts on the network. The appropriate CLS to use will depend on the specific client and what type of contexts the application(s) supports. For instance a client might support a method which allows the application to specify a CLS or, in much the same way as an Internet user may choose one search engine over another, a client's interface could allow the application user to explicitly specify the URI of the lookup service.

Many interesting research challenges are associated with the development of useful mobile clients within the CAwbWeb framework. The long-term vision is that through a general purpose interface (such as a Web browser), contextual Web-pages could be dynamically generated and allow the user to both request interfaces appropriate to different contexts and interact with their environment through a context-aware mobile "browser".

3.2 Contextual Lookup Service

Once the client has produced a context description and specified intention it may pass these along to a lookup service. The Contextual Lookup Service (CLS) maps Web services in the cloud to appropriate contexts. This enables the CLS to determine whether or not a context exists within which appropriate services can satisfy a specified intention. If such a context is found then its appropriate Possible Action Set is returned to the client. The Possible Action Set (PAS) specifies those actions supported by Web services in the cloud appropriate to the context. The PAS also specifies how to access the appropriate Context-Aware Intention Compiler(s) (CAICs) that support the PAS.

The interface of the CLS is defined as accepting an Intention Object (specified in the Intention Specification Language from section 4.2) and a Context Description (specified in a Context Description Language discussed in section 4.1). The lookup service must return a status (OK or NOT FOUND) along with a Possible Action Set (PAS) if status is OK. However, while the interface of the CLS is well defined, its implementation is flexible. For instance, a CLS may simply map geographic locations to nearby contexts. A more complicated CLS may behave like a search or recommendation engine, using the Context Description to find one or more optimal services to satisfy the intention.

A particular CLS may also define its own PAS registration interface. Once a PAS has been specified for a particular mobile-cloud system, the PAS must be registered with a CLS along with a description of its appropriate context. For instance a CLS could allow PASs to be registered based on their geographical location. Alternatively, a CLS could mine the Web Service Description Language (WSDL) [3] specifications of many mobile-cloud services and automatically generate appropriate PAS specifications to support a mobile-cloud services search engine.

The proposed CLS presents multiple research challenges. There has been much prior research on specifying or describing context [18] such as Context OWL [1], however a language to specify possible actions within these contexts along with the rules that govern them must be formalized. This language of possible actions must be related to how Intentions are specified, such that a PAS could be evaluated for satisfiability relative to an intention. Therefore, it is suggested that the Action Specification Language (ASL) and Intention Specification Language (ISL), specified in sections 4.3 and 4.2 respectively, be developed in conjunction with one another. Finally, the implementation of contextual search or recommendation engines can draw upon the wealth of research in context-aware recommendation [13].

3.3 Context-Aware Intention Compiler

Given a set of possible actions within a context and the rules that govern these actions, the Context-Aware Intention Compiler (CAIC) would support just-in-time compilation of intentions specified in the Intention Specification Language (ISL). The CAIC produces "Actuation Programs", which are XML files specifying "Actuation Instructions" and their data dependencies. The instructions themselves map to the URIs of mobile-cloud Web service methods. These Web-services are specified using WSDL. The compiler uses these WSDL specifications to build a resource-method graph mapping the data dependencies of the specified intentions to sets of actuation instructions, which in turn map to a set of Web-service resource methods. The set of instructions supported by a particular compiler and implemented by context-aware Web services is called an Actuation Instruction Set (AIS). Actuation Programs are executed by a Actuation Program Interpreter. In order to interpret an Actuation Program, the Interpreter must support the AIS used by the program. The interpreter is an integral part of the mobile client discussed in section 3.1. Starting with a set of mobile-cloud Web services, the remainder of this section will describe how the compiler processes Web service descriptions, integrates intentions together with contextual actions and

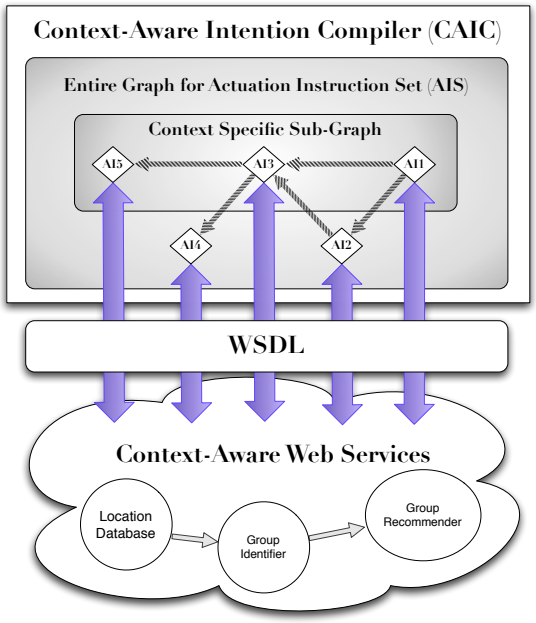

Figure 2: The dependency graph of actuation instructions used by the Context-Aware Intention Compiler and the mapping from instructions to mobile-cloud Web services.

rules, and produces actuation programs that can be interpreted by the client.

A mobile-cloud system should describe its function using a WSDL specification (as described in section 4.5). The CAIC can then use the WSDL document to create or modify its resource-method graph to reflect the system functionality and data dependencies (as depicted in Figure 2). Sub-graphs within the resource-method graph would then correlate to actions in a certain environment or application. Using a Possible Action Set (PAS), which specifies the possible actions and rules within a specific context, the graph could be modified to represent the possible actions and rules in the particular context as shown in Figure 2. This resource-method subgraph represents a set of actions that achieve the desired intention that is appropriate to the context.

The following explains the steps involved in the basic CAIC compilation process, from the mobile client's request to returning the Actuation Program to the client. Given a compiler (CAIC) that supports a certain instruction set (AIS) and associated mobile-cloud Web services, **(Step 1)** the mobile client may submit an Intention object and governing Possible Action Set (PAS) to the compiler. **(Step 2)** The compiler then uses the PAS to modify the full WSDL-based resource-method graph to represent the context-appropriate actions and rules. **(Step 3)** The intention object is then mapped to a set of possible actions in the contextual sub-graph that satisfy the desired intention. **(Step 4)** This data dependent and contextually appropriate resource-method sub-graph is then represented in XML as an Actuation Program and **(Step 5)** returned to the client to be processed by its Program Interpreter.

The proposed CAIC presents a number of interesting research challenges. An internal model (the instruction graph) will need to be implemented to represent data dependencies between mobile-cloud resources and their methods. Also, algorithms will have to be developed that modify the graph appropriate to the Possible Action Sets, enforcing all necessary contextual requirements such as privacy and security. Finally, a language must be formalized to efficiently represent the Actuation Program and its inherent data dependencies. While existing build languages like Apache Ant are able to express tasks and their dependencies, this Actuation Program language will have to be optimized for efficient distributed interpretation, which may include security or trust requirements.

3.4 Actuation Program Interpreter

Once an Actuation Program has been created by the CAIC, it must be executed. This is to be done by an Actuation Program Interpreter. The interpreter must know how to interpret the Actuation Instruction Language. In particular, an interpreter must support the Actuation Instruction Language used by the actuation program.

The most basic interpreter would probably be a simple program that accepts a set of tasks and their URI methods as inputs, and then calls all URI methods in the appropriate order, passing the data from task to task as specified by the program. This type of interpreter may exist solely on the client for convenience as shown in Figure 1. However, more mature Actuation Instruction Languages (AILs) may support execution of separate tasks in such a way as to support trusted (distributed) computing spaces protected from one another. More mature AILs may also allow specification of security requirements for data transfer between tasks, such as encryption between private data sources and anonymization services.

The design of an Actuation Instruction Language (AIL) and Interpreters to support Actuation Instruction Sets specified in the AIL presents an interesting language research challenge. However, maturing the AIL and specifying rich Actuation Instruction Sets supporting dynamically generated Actuation Programs poses a range of research challenges that span many fields; for example:

- **Distributed Systems** A scalable protocol for distributed execution of actuation programs.
- **Security** Secure/Trusted Computing spaces using

```
<intention name="playRecommendedMovieTrailerForGroup">
    <input>
        <group near="display-1223"/>
    </input>

    <output>
        <action name="playRecommendedMovieTrailer">
            <videoPlayback type="recommendedMovieTrailer"/>
        </action>
    </output>
</intention>
```

Figure 3: ISL example

Onion-like encryption and execution of tasks by trusted and un-trusted Web services.

- **Privacy** Specifying privacy requirements such as encryption between private data sources and trusted anonymization services.

- **Adaptive Execution Optimization** Supporting parallel tasks, reuse, caching, and conditional branch prediction to optimize execution of Actuation Programs.

4. LANGUAGE STANDARDS

This section describes the languages used in CAwbWeb. The development of each of these languages presents important challenges in the CAwbWeb framework.

4.1 Context Description Language

The Context Description Language (CDL) describes the types of context supported by our framework. Some examples of contexts described by CDL include location coordinates (latitude and longitude), the name of a location (The Village Mall), time and date, social connections (friends of John), environmental conditions (the current temperature at a specified location), and historical data (the location trace history for John). As described in [5], there are a number of ways to model context, including key-value models, markup scheme models, graphical models, object oriented models, logic-based models, and ontology based models. [5] indicates that ontologies are the most expressive models and fulfill most of the requirements for context modeling. Therefore, we choose to represent CDL as an ontology using the Web Ontology Language (OWL) [2]. The Context OWL ontology [1] could be used as a basis for defining our CDL standard.

The major research challenge regarding CDL is generalizing and extending prior work, such as Context OWL, to model the wide range of contexts supported by our framework. We also need to investigate metrics and standards for measuring how well contexts are defined with regard to a particular Contextual Lookup Service (CLS). The CLS will use these metrics when searching for a context that can best satisfy a specified intention.

4.2 Intention Specification Language

The Intention Specification Language (ISL) describes actions that may be performed on objects. Actions that may be performed on objects are specified in terms of inputs and outputs. For example, suppose that we intend to play a recommended movie trailer for a group of individuals jointly viewing a large-screen display. The inputs for this intention are the group of users that are near the display. The output for this intention is playing the recommended movie trailer. In this example, the inputs consist of objects and the output consists of an action and an object. Figure 3 shows a portion of the ISL document used to express the intention described in this example.

The major ISL-related research challenge is defining a rich ontology for expressing the wide range of possible intentions in context-aware systems. The development of ISL is closely linked to the development of the Action Specification Language (ASL), since a Possible Action Set (PAS) expressed in ASL will be used to determine if an intention can be satisfied for a particular PAS.

4.3 Action Specification Language

The Action Specification Language (ASL) describes the possible actions that a system may perform. Each action is specified in terms of the instructions and data used to perform the action. Actions may be associated with rules. Rules may define the requirements for inputs to actions, or may be used to describe the requirements for interactions between actions. For example, consider the process of retrieving the anonymized social network profile information for a user. The first step in this process is to perform the action of anonymizing the user's social network information. This action uses the "anonymize" instruction, and the data item for this instruction is the user's social network ID. The next step in this process is to perform the action of retrieving the user's social network information, which is dependent on the first anonymization action. This dependency is specified as a rule. Figure 4 shows a portion of the ASL document used to express the sequence of actions described in this example. In our framework, each ASL document is associated with a specified context and intention.

Regarding ASL, the major research challenge is to define a high-level language for expressing the possible actions that may be performed in context-aware systems. ASL must be expressive enough to fully represent the actions and rules that may govern the interaction of context-aware Web services.

4.4 Actuation Instruction Language

The Actuation Instruction Language (AIL) describes the instructions to execute to implement an intention. Given a set of possible actions (also called a Possible

```
<?xml version="1.0" encoding="UTF-8"?>

<action name="anonymizeSocialNetworkData">
    <anonymize type="socialNetworkProfile"
        src="${socialNetworkID}"/>
</action>

<action name="getSocialNetworkProfile"
    depends="anonymizeSocialNetworkData">
    <get type="socialNetworkProfile"
        src="${anonymizedSocialNetworkID}"/>
</action>
```

Figure 4: ASL example

```
<task name="getGroupMembers">
    <instruction name="getUsersNearby" method="GET"
        targetURI="${getNearbyUsersServiceURI}"/>
    <instruction name="getSocialInfoForGroup" method="GET"
        targetURI="${getSocialInfoServiceURI}"/>
</task>

<task name="getRecommendedMovieTrailer"
    depends="getGroupMembers">
    <instruction name="getRecommendedMovie"
       method="GET" targetURI="${getRecommendedMovieServiceURI}"/>
    <instruction name="getTrailerForMovie"
       method="GET" targetURI="${getTrailerForMovieServiceURI}"/>
</task>

<task name="playRecommendedMovieTrailer"
    depends="getRecommendedMovieTrailer">
    <instruction name="playRecommendedMovieTrailer"
       method="PUT" targetURI="${videoPlaybackServiceURI}"/>
</task>
```

Figure 5: AIL example

Action Set, or PAS) and an intention document, the Context-Aware Intention Compiler (CAIC) generates an AIL program for implementing the intention. Consider the example intention described in subsection 4.2. The AIL generated by the CAIC for this intention is shown in Figure 5. The AIL program is composed of a series of tasks, with each task containing one or more atomic instructions. Each task may optionally specify its dependencies as a list of other tasks. If dependencies are specified, then the tasks in this dependency list are executed before running the task that specifies those dependencies.

The primary AIL research challenge involves developing a language that completely describes how to use distributed actuation program interpreters to execute a compiled intention. Since an AIL program may be processed and passed through a series of distributed interpreters, only those sections of the AIL program that pertain to a specific interpreter should be visible to that interpreter. To securely isolate each interpreter into its own trusted computing space, we will need to encrypt each section of the AIL program with the public key of the interpreter for that section. The XML Encryption standard [4] can be used to perform this encryption of the AIL program.

4.5 Web Services Description Language

The Web Services Description Language (WSDL) is an XML-based language for describing web services [3]. WSDL is a well established standard; WSDL version 2.0 is a W3C recommendation. As described in subsection 3.3, the CAIC uses WSDL documents for each Web service supported by our framework to compile intention documents into AIL programs.

5. A CAWBWEB EXAMPLE

Recall the story in section 1, in which Mark visited the video rental store, was presented with trailers chosen specifically for him, and was then saved from a fire by an alarm system which took his particular health needs into account. This example was chosen because it demonstrates how disparate context-aware services residing on the cloud can be driven by a mobile application without knowledge of the particular services and their interfaces. We will now discuss how such a situation would be implemented using the CAwbWeb framework.

Mark enters the video rental store. As Mark approaches the new releases aisle, the mobile client application on his Nokia N97 smartphone expresses the intention to update its location. This intention, and a context description including GPS location, is sent to the Contextual Lookup Service (CLS). The CLS identifies a compiler service (CAIC) that supports the context-aware services at this rental store. The CLS returns the compiler's PAS to the client, which in turn sends a program request including location context to the compiler. The CAIC compiles the request into an actuation program which specifies the location update URI of the video store's "SocialFlix" service. SocialFlix is a service which plays trailers throughout the store recommended for the people in the store at that time.

Once aware of Mark's presence, SocialFlix becomes a client in our framework. Since SocialFlix performs trailer requests regularly it has cached the information for the appropriate CAIC, which it originally received from the CLS. SocialFlix submits a request to the CAIC to play a trailer recommended for Mark. The CAIC returns an actuation program to SocialFlix specifying the appropriate "play trailer" URI for the screen nearest Mark, along with the video URI of that trailer. Mark finds this trailer interesting, so he grabs the DVD for this movie and proceeds to the checkout counter.

As Mark is waiting in line, an electrical short spontaneously occurs at the checkout counter, initiating a fire. The smoke detector senses the presence of smoke and notifies the fire suppression system by forming and submitting"suppress fire" and "fire alert" intentions to

the appropriate CAIC. The CAIC composes an actuation program which suppresses the fire and appropriately alerts those in the store to leave. In this case, Mark's presence in the store changes the possible action set (PAS) so that strobe alert lights are not used unless they can be strobed slower than 5 hz, due to Mark's photosensitive epilepsy. The actuation program is run, activating the fire alarm and sprinklers. Mark leaves the building and the fire is extinguished by the sprinklers.

6. CONCLUSIONS

This paper has presented a vision of mobile-cloud computing in which context-aware services are organized and integrated by a Context-Aware Intention Compiler (CAIC) which turns well defined intentions into "actuation programs." Run-time creation of these programs allows contextual information from a mobile phone and the environment to be integrated in real-time. Furthermore, the mobile device can look up context-aware services using a Contextual Lookup Service, which maps context and intention to the appropriate Context-Aware Intention Compiler. Use of the CAwbWeb framework allows mobile-cloud challenges to be divided into four major concerns: specifying intention, describing context, identifying appropriate actions, and efficient actuation of those actions. We believe the adoption of the CAwbWeb framework will allow future mobile-cloud research to focus on particular problems whose solutions can be quickly integrated and further developed, accelerating research in the area of context-aware mobile-cloud systems.

7. REFERENCES

[1] Context owl. `http://on.cs.unibas.ch/owl/1.0/Context.owl`.

[2] Owl web ontology language reference. `http://www.w3.org/TR/owl-ref/`.

[3] Web services description language (wsdl) version 2.0 part 1: Core language. `http://www.w3.org/TR/wsdl20/`.

[4] Xml encryption syntax and processing. `http://www.w3.org/TR/xmlenc-core/`.

[5] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[6] A. Beach, M. Gartrell, X. Xing, R. Han, Q. Lv, S. Mishra, and K. Seada. Fusing mobile, sensor, and social data to fully enable context-aware computing. In *The Eleventh Workshop on Mobile Computing, Systems, and Applications (ACM HOTMOBILE) 2010*, 2010.

[7] G. Biegel and V. Cahill. A framework for developing mobile, context-aware applications. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, page 361, 2004.

[8] L. P. Cox, A. Dalton, and V. Marupadi. Smokescreen: flexible privacy controls for presence-sharing. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 233–245, New York, NY, USA, 2007. ACM.

[9] A. Dey, G. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166, 2001.

[10] E. W. Dijkstra. On the role of scientific thought. In *Selected writings on Computing: A Personal Perspective*, pages 60–66. Springer-Verlag New York, Inc., 1982.

[11] N. Eagle and A. Pentland. Social serendipity: Mobilizing social software. *IEEE Pervasive Computing*, 4(2), April-June 2005.

[12] W. K. Edwards and R. Grinter. At home with ubiquitous computing: Seven challenges. In *Proceedings of the 3rd International Conference on Ubiquitous Computing (Ubicomp 2001)*, pages 256–272, May 2001.

[13] C. M. Gartrell. Socialaware: Context-aware multimedia presentation via mobile social networks. Master's thesis, University of Colorado at Boulder, December 2008.

[14] T. Gu, H. Pung, and D. Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1–18, 2005.

[15] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *Proc. of the 6th ACM Conf. on Embedded Network Sensor Systems (SenSys 2008)*. ACM, Nov. 2008.

[16] A. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot. MobiClique: middleware for mobile social networking. In *Proceedings of the 2nd ACM workshop on Online social networks*, pages 49–54. ACM, 2009.

[17] B. Schilit, N. Adams, R. Gold, M. Tso, and R. Want. The PARCTAB mobile computing system. In *Proceedings Fourth Workshop on Workstation Operating systems (IEEE WWOS-IV)*, page 2. Citeseer, 1993.

[18] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *First International Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp*, September 2004.