ECC : Edge Cloud Composites

Ketan Bhardwaj, Sreenidhy Sreepathy, Ada Gavrilovska, Karsten Schwan
College of Computing
Georgia Institute of Technology
Atlanta, GA
{*ketanbj, sreenidhy*}*@gatech.edu*, {*ada,schwan*}*@cc.gatech.edu*

***Abstract*—With an ever increasing number of networked devices used in mobile settings, or residing in homes, offices, and elsewhere, there is a plethora of potential computational infrastructure available for providing end users with new functionality and improved experiences for their interactions with the cyberphysical world. The goal of our research is to permit end user applications to take advantage of dynamically available, local and remote computational infrastructure, without requiring applications to be explicitly rewritten and/or reconfigured for each scenario and with minimal end user intervention.**

***Edge Cloud Composites* (ECC) make possible the dynamic creation of virtual computational platforms that (i) can be composed from specific capabilities – competences – of participating devices, (ii) are guided by end user-centric abstractions capturing current user context and user intent, and (iii) use dynamic methods for device discovery and ECC maintenance. In contrast to datacenter clouds, ECC participants can include both virtualized and non-virtualized devices, and in addition, services running remotely, made possible by ECC's CIC abstractions, where C(ompetence) captures the functional capabilities of accessible devices and/or remote services, (I)ntent articulates end user desires, and (C)ontext describing the current operating environment.**

Concrete examples prototyped in this work include Android applications for distributed video playback, collaborative UI, and a distributed augmented reality application. For all such applications, an ECC composed from available devices, and guided by ECC's CIC notions, obtains up to 86% performance improvements and reductions in energy consumption of up to 37% compared to running on a single device. A resultant advantage in using ECCs to run applications is the ability to avoid the unpredictable latency variations seen in device-remote cloud interactions.

***Keywords*-Device cloud; Device services; Virtual platforms; Context-aware services; Platform virtualization; Competence; Intent;**

I. Introduction

To obtain long battery lifetimes, today's mobile devices, designed for low energy consumption via energy-efficient processors, limited memory capacities, and on-device active power management, intentionally sacrifice the potential levels of computational ability and functionality attainable within their current form factors. Cloud-based services run by companies like Google, Apple, Amazon, etc., offer one means for applications to escape the limits imposed by individual devices, yet at the same time, mobile end users are surrounded by numerous other networked devices, whether at home, in the office or marketplace, or in the car. Surprisingly, most of those devices do not inter-operate seamlessly, with each other and/or with the mobile user's portable device, thereby compromising theoretically achievable computational efficiency and more importantly, end user's potential experiences [1].

Current ways to use nearby devices continue to require manual orchestration [2][3], typically demanding end users to take explicit steps to setup devices for inter-operation, and in addition, such setups may differ across the tasks undertaken, e.g., file sharing, multimedia hubs, etc. As a result, the eco-system in which these devices operate remains fragmented. Beyond well-known hardware-software incompatibilities and lack of standardized protocols [4], reasons for such fragmentation include limits in (i) knowledge about the context in which a device is operating, i.e., which nearby devices are currently accessible and usable, (ii) information about nearby device capabilities, (iii) instructions about the task at hand, and (iv) data about the steps needed to combine and jointly use the capabilities of multiple devices.

Our research explores the utility and efficient use of 'nearby devices' for enhancing the capabilities of end users' mobile devices and experiences. We borrow from data center systems the notions of resource consolidation, elasticity, and dynamic orchestration of many machines' computational capabilities, with minimum manual intervention (e.g., as in 'cloud computing'). The implicit knowledge about tasks to be carried out, resource accessibility and capabilities plays a vital role in could computing to achieve benefits of these notions. On that basis, our solution approach creates a consolidated virtual pool of device capabilities – Competences – composed into a virtual platform able to efficiently carry out some currently intented task – Intent – enabled and constrained by the context in which the end user currently operates – Context. The resulting E(dge)C(loud)C(omposities) are comprised of sets of cooperating devices guided by CIC: Competences, Intent, and Context.

ECC technology builds on our own earlier work on 'device clouds' [5] as well as on previous efforts like [1][2][3], which showcased the need and benefits of platform composition, enumerated and compared technologies that can aid creation of non-fragmented device ecosystems [6].

Also leveraging middleware for ubiquitous/pervasive applications [7][8][9], etc., ECC contributes the novel CIC abstractions and associated runtime mechanisms that make it easier for end user applications to interact with and take advantage of nearby devices and the remote cloud, all without requiring applications to be rewritten and/or reconfigured (for each scenario) and with minimal end user intervention. This work contributes the following:

- The proposed CIC concepts are shown to be sufficient and necessary for consolidating the device ecosystem.
- The concepts' implementation gives rise to mechanisms in which network enabled devices can be composed into efficient virtual computing platforms, termed Edge Cloud Composites (ECCs), providing a model for application developers to realize purposeful device-device and/or device-cloud interactions.
- CIC guides the creation of ECCs that are able to carry out representative, useful end user tasks, shown superior in performance and the consequent levels of end user experiences offered to end users, compared to the capabilities of a single device and the use of remote cloud resources. The ECCs created are dynamic, reactive to altered device competences, and to changes in intent or context.

In remainder of this paper, we discuss the need and benefits of using ECCs with real world usecases, in the home, office, and in-vehicle environments in section II. Section III introduces the CIC concepts. Sections IV and V describe the design and implementation of ECC respectively. Experimental results, shown in section VI, demonstrate the importance of the CIC concepts and the overheads with which changes in CIC induce consequent changes in ECC used to carry out some currently running user tasks. Related work and concluding remarks appear at the end.

II. Motivating Use Cases

We first discuss the need for ECC by presenting several motivating use cases and the resulting benefits. Illustrated in Figure 1 (i) is a distributed video playback application in the context of an end-user's home environment, consisting of a tablet, PC and/or laptop, large screen TV, and a home theater audio system. In this scenario, to play a video from the handheld as a file source, ECC chooses the laptop for its video decoder competence, and the large TV for its display. User intent requires a linear, i.e., synchronous, video playout pipeline extending across participating devices.

The second example, illustrated in Figure 1 (ii), shows a collaborative UI in an office context, where multiple devices like laptops, phones, and projector are present with participants in a meeting. In order to allow participants to use their own devices to manipulate UI elements on the projected screen, ECC seamlessly instantiates multiple instances of the same competence, i.e., touch pads, to create a collaborative user input system. User intent requires instantiation of a same capability of many users converging to one device i.e., many to one processing pipeline spanning participating devices.

The third example, illustrated in Figure 1 (iii), shows an in-vehicle distributed augmented reality application, where information about the vehicle's current surroundings is projected on the car's windscreen. To do so, ECC may utilize the car's IVI (assumed to include a front facing camera) and the user's smartphone to seamlessly collaborate. Since the camera cannot wait for the remote cloud to send back its response or annotations to be displayed on the windshield of the car, this connection has to be asynchronous in nature. User intent requires a branching, i.e., asynchronous processing pipeline spanning participating devices.

The scenarios described above illustrate the need of CIC notions and explain how they give rise to the construction of appropriate edge cloud composites (ECCs). More generally, they demonstrate ECCs guided by CIC to be capable of achieving following benefits.

Seamless use of accessible devices. The current state of art in multi-device protocols include uPnP Alliance [10] Samsung AllShare [11], Microsoft's SmartGlass [12], and Apple AirPlay [13] – facilitates content sharing among devices, but does not yet make it possible to automatically create a pool of resources available for shared use. ECC makes possible the automated assembly of computational capabilities, to carry out user-intended tasks. Because the devices being composed can vary widely in terms of their distinct features and in addition, operate in dynamic surroundings, ECC goes beyond datacenter cloud technology to explicitly formulate and use Competence, Intent, and Context, as a way to expose and use the diverse features of edge devices.

We motivate the use of these three notions as follows. First, competence captures the device' capability to carry out certain task. Next, intent is needed because assembling devices purely on the basis of performance-centric metrics is insufficient, as evident from the simple example above in which the user wishes to control video playback via her (relatively slow) handheld device vs. the (faster) home PC. The example demonstrates that it is important for the user to be able to explicitly express the intent to view the video on her handheld, independently of the availability of other resources. Finally, contextual information like an end-user's presence in a particular room, available network connectivity, etc., may warrant changes in the resultant assembly. For example, choosing the PC situated in a study room and the TV in the living room achieves best balance between performance and power, but does not make sense for an end user who is working in the kitchen.

End-user device consolidation. While virtualization has enabled hardware consolidation in large data centers, end consumers continue to have to explicitly use multiple, disparate devices to run desired functions or utilize certain

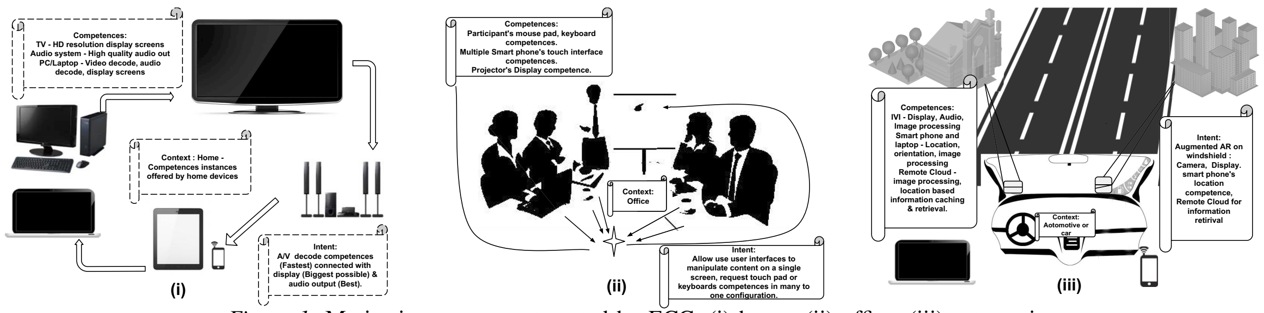
Competences:
TV - HD resolution display screens
Audio system - High quality audio out
PC/Laptop - Video decode, audio decode, display screens
Context : Home - Competences instances offered by home devices
Intent:
A/V decode competences (Fastest) connected with display (Biggest possible) & audio output (Best).
(i)
Competences:
Participant's mouse pad, keyboard competences.
Multiple Smart phone's touch interface competences.
Projector's Display competence.
Context: Office
Intent:
Allow use user interfaces to manipulate content on a single screen, request touch pad or keyboards competences in many to one configuration.
(ii)
Competences:
IVI - Display, Audio, Image processing
Smart phone and laptop - Location, orientation, image processing
Remote Cloud - image processing, location based information caching & retrieval.
Context: Aotomotive or car
Intent:
Augmented AR on windshield : Camera, Display. smart phone's location competence, Remote Cloud for information retirival
(iii)

Figure 1: Motivating use cases targeted by ECC: (i) home, (ii) office, (iii) automotive.

device features. ECC proposes to dynamically assemble capabilities or disparate devices' functionalities based on their relative abilities – competences – and to do so in ways that permit devices to be shared across multiple users and applications. ECC can do so without further device-level support, or it can leverage lower level functionality helping with device integration and use. One such functionality, based on extensions of modern hypervisor technologies, is described in [5], and it is this functionality on which the current implementation of ECC relies. However, ECC can also be layered on alternative 'cloud' realizations, like the 'cloudlet' technology using entire virtual machines to run desired end user functions [14]. We note this fact as an important element of the ECC design, because it demonstrates its unique attribute of being able to work with (i) 'raw' devices – devices that can serve only certain and specific purposes in user environments, like cameras or displays, (ii) programmable devices – those that can run diverse, pre-programmed functions, and (iii) fully virtualized resources, including home PCs running hypervisors and remote datacenter machines running cloud infrastructures. ECC supports these choices because it does not want to rely solely on remote cloud resources to augment end users' mobile devices, due to unpredictable end-to-end network latencies to remote cloud resources, intermittent connectivity, and the potential costs related to data usage on handhelds.

Leveraging locality in tasks. Typically, the tasks accomplished by devices fall into two categories i.e., (i) the tasks requiring aggregation of information from different sources, efficiently delivered using central entities like cloud based services, e.g., crowd sourcing applications, social network applications, etc., vs. (ii) the tasks that can potentially leverage locality, e.g., distributed multimedia playback, collborative UI for brainstorming in a office or classroom, etc. For the latter, employing remote cloud based services can actually deteriorate overall efficiency and experience, due to unpredictable communication latencies as well as dynamic changes in cloud accessibility. ECC provides to such applications 'fallback' options that leverage nearby in place of and/or in addition to remote cloud resources.

The manner in which ECC achieves these goals is discussed next.

III. ECC Concepts

Intuitively, ECC articulates (i) what a user wants to do, (ii) which resources are available, and (iii) what those resources' abilities are. Concretely, in creating ECC, these notions are used to form an assembly ready to be mapped to some set of physical devices (and their resources) present in an environment which includes multiple devices and/or cloud based services. Concretely, they take the form of CIC concepts in ECC described in this section.

A. Device Competences

ECC operates at a level of abstraction in which device descriptions go beyond their innate compute, memory, communication capacities to instead, characterize devices by the functionalities they are able to provide. ECC's notion of *competence* has two parts – static and dynamic. The static part comprises of an abstract functional description or *functionality* (e.g., display) with associated qualitative parameters (e.g., frames per second, resolution) and, if applicable, certain physical characteristics (e.g., screen size) or *characteristics*. The dynamic part describes the means to utilize the device competence, i.e., its *accessibility*, and its current state in terms of resources, or *availability*. Stated more generally, the competence is described as a structure with tuples listing the physical and functional resources – static and dynamic. In keeping up with other cloud abstractions (e.g., EC2's 'small' vs. 'large' virtual machines), ECC does not describe underlying devices at the level of detail of their hardware characteristics. It is up to either lower levels of the system software or implementation of specific instances of competence on a device. The examples of such low level support include hypervisor-level 'device cloud' functionality as described in [5], to provide the network linked physical devices (and/or remotely run virtual machines) to which ECC can map its assemblies. The approach is consistent with how virtual machine ensembles in data centers are formed – ECC's level of operation – vs. how they are deployed – via cloud infrastructures and hypervisors. Details about specific competence instances are provided in implementation section.

Summarizing,

Competence *is defined as a tuple representing a device's exposed functionality, characteristics, availability and accessibility.*

Specifying competence is a semantic problem which is inherently hard for machines or devices. This is a way more general problem and in ECC, we handle it partially. Many approaches are proposed by researchers to address the problem of representing semantics, e.g., various model based approaches [15], XML, JSON, RDF [16] based structures utilized for semantic web, etc. The novelty of this work lies in simple and effective formulations to address the issues stated above by decomposing the information into separate and manageable chunks of information suitable for implementation on different type of devices. One may argue that it is not feasible to specify the semantic information and/or the application requirements or structure in advance. But, this is a well known trade off between application flexibility and scalability in deployment of distributed applications in the data center world. Also, we argue below that a bottom up hierarchical representation of competence in multiple levels of abstractions provides a reasonable way to handle the complexity of representing functionalities in devices. At the lowest level, competence represents a piece of software running on a hardware platform of a device which has some compute capability with some specific I/O which do not have any dependency, termed as *raw competence*. Moving a level up, this computation and I/O operation result in some user observable events or the other way round that some user initiated event cause a specific piece of software to run or I/O operation to be performed, termed as *feature competence*. Moving further up, temporally combination these events result in realization of a user desired task, termed *usecase competence*. ECC uses competence to define 'context' and 'intent' which are described next.

B. User Intent

ECC *intent* specifies (i) the set of tasks, and (ii) their interactions, required to attain some higher level user goal, with (i) stated as required competences and (ii) as guidelines. By stating tasks in terms of required competences, ECC leaves room for widely varying implementations of these tasks i.e., specific instances running on certain devices, or as instances running in remote virtual machines, with differing degrees of CPU and memory usage, etc., but all competent to provide the stated functionality at sufficient levels of quality. Concerning task connectivities, ECC does not simply list how one task connects to another but rather, states *intent guidelines* about task interactions, such as whether interactions are synchronous vs. asynchronous, for example. Concrete examples of intents with guidelines are shown in Figure 1 (i) for synchronous and Figure 1 (iii), for asynchronous interactions between competences, and depicting many-to-one interactions in Figure 1 (ii). More generally, guidelines specify (i) *topology* – the connection configuration between competence instances (e.g., linear, many to one, etc.), (ii) *traversal* – referring to how events travel among competences (e.g., synchronous, asynchronous, listening, etc.), and (iii) *tie* – placing constraints on certain competences o be instantiated on a particular device. Collectively, we refer to these as the 'three Ts'. Concisely, ECC defines intent as follows:

Intent *is defined as an ordered sequence of events on 'partially specified' competences linked by guidelines.*

where guidelines place certain constraints on competences, including on their communications. ECC uses the concept of context to handle the dynamimism in cyberphysical environment which is described next.

C. Context

Devices operate in dynamic environments, so that information about the device functionalities available at a given point is crucial for applications crossing device boundaries. The notion of context abstracts this information for ECC participants. Context captures current device availability and states, including their status due to multi-tenant operation, mobility constraints, current location, etc. Since ECC can operate across distributed devices – device clouds – context is a distributed entity maintained at runtime, where a device's competence to carry out some intended functionality is affected by factors that include its current availability, accessibility, etc. To reason about choosing the best competence to realize an intent, in a given context, ECC distinguishes (i) *offered competences*, which reside on the same device, and are accessible directly and offered for provisioning, from (ii) *local competences* that exist on a nearby device currently accessible over some local area network, from (iii) *remote competences* that exist on a device or a remote cloud accessible via the Internet, from *cached competences*, which were recently provisioned in previously requested intent. Stated precisely, a ECC context is defined as follows:

Context *is defined as the set of currently accessible competences.*

Context as a list of competences and not a list of devices provides ECC the ability to represent a situational as well as operational context because of the way competences are defined. Using competences in a context, ECC can successfully derive the operational and situational information. The ECC context is sufficient to handle its goals (or any other system) which are to provide dynamic selection of the most appropriate device where a functionality should be instantiate.

IV. Design

A. Software Architecture

ECC provides an application-conscious layer of abstraction on top of lower level functionality that permits nearby (and remote cloud) devices to cooperatively carry out the tasks desired by end users. For any such set of tasks, the

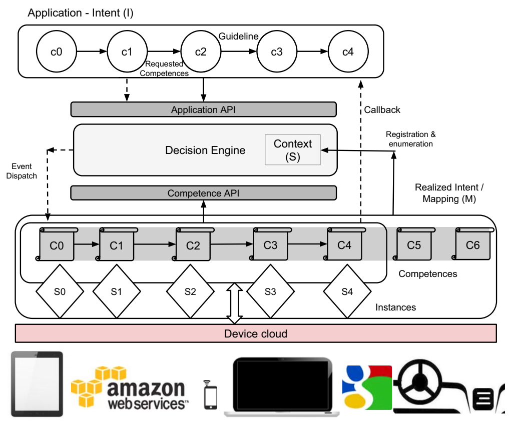

Figure 2: ECC Software Architecture

role of ECC is to acquire and use the resources required to carry them out. The role of the underlying device cloud infrastructure (if available), then, is to provide the actual resources or devices to which ECC tasks can be mapped. The outcome is the two-tier infrastructure shown in Figure 2, depicting sample devices and the low-level device cloud software supporting their joint use, with ECC utilizing these resources on behalf of specific end users.

The ECC accepts 'intent' descriptions. These are translated to a set of devices offering appropriate competences. Their description, along with additional context information, are used by a decision engine to determine the mapping to underlying competences implemented as node provided by the device cloud layer or implemented as standalone competences in a non-virtualized device. ECC places constraints on competence mapping, but leaves it up to the underlying device cloud to map its competence to specific underlying physical devices (note the analogy to VCPU to PCPU mappings in hypervisors, where it is up the a virtual machine to create the VCPUs to be run, but the hypervisor is the one that determines their mapping to an adequate PCPUs [17]). ECC picks federation of competence which may be present on the same device or multiple devices in a given context. The decision for choosing the device is transparent to ECCs decision engine.

The architecture's description makes clear the dual application-facing and cloud-facing roles of ECC, driven by intent for the former and utilizing competences and context for the latter. In this architecture, the abstractions of competences, intent, context (CIC) are what drive the decision engine that constitutes the logic or control tier that handles the registration of competences (discovery and enumeration), performs mappings of requested competences – composes ECC – as per the intent's guidelines, and establishes and maintains context.

Shown in Figure 3 is an example highlighting the necessity of ECC's dynamic mapping methods. For a video playback usecase, we show two different contexts, marked by blue dashed lines: A – with an accessible remote cloud as well as some local devices such as a TV and laptop, and B – comprised of only locally accessible devices. For different user intents – to view the video on a large screen vs. on a handheld device – ECC mappings will differ, as well. Mapping differences also arise from changes in competences – depending on the network connections between devices or their current loads – and from changes in context – where the user is currently located. The figure illustrates this by showing multiple possible competence assemblies marked by red dashed line (Figure 3).

B. Operation

Next, we outline the operation and interaction of the ECC elements shown in Figure 2.

1) ECC Application: ECCs application structure is inspired from widely deployed SOA distributed applications which are deployed on loosely coupled devices as opposed to tightly coupled servers in the data center world. The devices are said to be loosely coupled because of the dynamic nature of context which may change at any point in execution of applications. An ECC application starts by calling the initialization API, which in turn instantiates the context and initializes the decision engine. It then requests an intent using the dispatch API which triggers the decision engine to search for the competences required to realize the requested intent from the ones available in the current context, also considering the constraints (accessibility, network latency, etc.) and opportunities (availability of better competence). When specific competences are determined, an interaction with the underlying device cloud or ECC stack running on a device acquires the resources required by these instances of competences, much like regular cloud applications would launch VMs of varying capabilities into a datacenter cloud. Note, however, the importance of intent and context information for this mapping, as jointly, these may impose constraints on deployment, i.e., where specific functionality may be run (e.g., the only device suitable for running some functionality, say, a large-screen display in a specific location). Once successfully deployed, ECC then carries out an initialization phase in which it calls the initialization routine for selected instances of competence, creates and connects their communication channels as per the intent's guidelines, and finally, hands off control to the application for its execution. After this happens, the application dispatches a sequence of events on realized intent to carry out the desired task. During deinitialization phase, the context, decision engine and the provisioned competences are deinitialized. To specify an intent application developers need to create a list of descriptors which guide ECC's search for appropriate functionality. ECC doesn't impose any restriction on the extent of specification. Dependencies are specified in advance by developer in intent guidelines. Also needed are three Ts as guidelines to create the appropriate

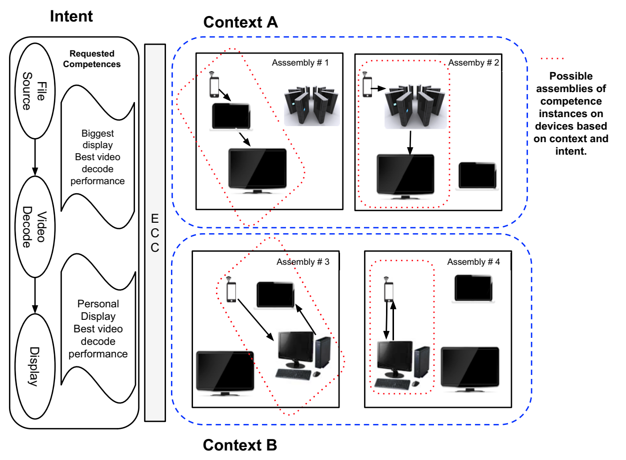
Intent
Context A
Requested Competences
File Source
Video Decode
Display
Biggest display Best video decode performance
Personal Display Best video decode performance
E C C
Asssembly # 1
Assemby # 2
Assembly # 3
Assembly # 4
Possible assemblies of competence instances on devices based on context and intent.
Context B

Figure 3: ECC operation and creation of different assemblies with different context and intent descriptions for distributed video playback.

overlay. The protocol for data transfer implemented in ECC leverages EVPath's transport (briefly described in Section V) to actually transfer the data in form of attached payload with each event but the interpretation of data content is implementation specific.

2) ECC Competence Instance: The competence instance is implementation of some functionality offered by a device. During ECC initialization, the instance's registration routine is triggered by the decision engine. In this phase, each instance generates its competence's static part and uses ECC's registration API to register itself. When an instance is in the initialization phase, it generates two addresses: (i) a process address and (ii) a dispatch address. The competence's static part and these addresses are used by the decision engine to realize the requested intent.

3) ECC Decision Engine: The decision engine handles the creation and/or synchronization of contexts, advertises offered competences to devices, and dispatches intents, the latter via intelligent and automatic composition of competences and/or by reconfiguring a realized intent in response to failure or context changes. The decision engine employs the following mechanisms to achieve these goals.

Discovery mechanisms handle the naming and enumeration of the competences that are available in the user's current context, using a datagram-based protocol for device discovery implemented by the EVPath event-based middleware [18] which are then used by the decision engine to advertise said competences to other devices.

Compose mechanisms search and select appropriate competence instances from the current context when dispatched. ECC's decision making process can be formalized mathematically as follows: Let A $\leftrightarrow$ B mean that there exists a network connection between devices A and B, i.e., A can access B's competences and vice-versa. If there are N number of devices offering competences at a given point of time, represented by competences (C_i), we present the CIC formulation as shown below.

Competences$(\mathbf{C_i})$ = Competences offered by device D_i

Let S_i represent the competence instances available in the current context, and defined as follows

$S_i = \cup_{j=0}^{j=n} C_j \exists\ D_i \leftrightarrow D_j$

We can now define context as a superset of all S_i and intent as a function of competences and guidelines as shown below:

Context (S) = $\cup_{i=0}^{i=n} S_i$

Intent (I) = $f(R, G)\ \forall\ R \subseteq C$

Where, R is set of requested competences, G are the guidelines specified in user intent and C is the overall competence of a system (C) = $\cup\ C_i$.

With these formulations, the goal of the decision engine is to find the best set of competence to instance mapping (**M**) for a given intent (**I**) in the current context (**S**). As discussed earlier also in section III, competence in (**R**) may be partially specified in varying details e.g. in distributed video playback usecase, R may include display competence which specifies display competence with 46 inch screen (constraint on physical characteristic) or 60 fps H.264 video decoder (performance constraint) etc. Instance selection is accomplished by reasoning on competence description, physical characteristics, quality parameters and instantaneous network latency in the same order and is also constrained by intent guidelines.

Goal : find M $\forall\ m_i \in$ **S**

Dispatch mechanisms facilitate direct event traversal among competence instances during operation, using EvPath [18] for event transport. Each event includes the event description and the associated payload, to facilitate seamless interactions between instances in a realized intent.

Reconfiguration and Failure mechanisms. In case of context change, or if any instance is not meeting the quality constraints as specified in intent, reconfiguration mechanisms are triggered. If any instance detects that it is unable to dispatch events to the connected instance, failure mechanisms are triggered. In both cases, the decision engine checks for another competence which is suitable replacement of competence instance. If found, it plugs in the new instance, and restarts the intent, again without involving the application else intent is destroyed and the application that requested the intent is notified using a callback.

V. Implementation Details

In this section, we briefly introduce EVPath, describe ECC implementation, Android-specific details and provide information about the competences implemented for motivating use cases.

A. EVpath

EVpath is designed to be an event transport middleware layer. Specifically, it is designed to allow for the easy implementation of overlay networks, with active data processing, routing and management at all points in the overlay. EVpath specifically does not encompass global overlay creation, management or destruction functions. Rather, it focusses on providing efficient environment for routine transport, while providing interfaces necessary for external management layers. The package is built around the concept of ”stones” (like stepping stones) which are linked to build end-to-end ”paths”. While the ”path” is not an explicitly supported construct in EVpath, the goal is to provide all the local, stone-level, support necessary to accomplish their setup, monitoring, management, modification and tear-down. Stones in EVpath are lightweight entities that roughly correspond to processing points in dataflow diagrams. For more details on evpath, you can refer [18].

B. *ECC Implementation*

ECC is implemented in low level C language to ensure its portability and ease of integration with device cloud software such as Cloudlets[14] and Stratus[5]. The implementation uses EVPath to abstract adressing, network access and realize the mechanisms described above. During initialization, the decision engine enumerates competences which are stored as shared libraries which are dynamically instantiated at runtime creating a fresh context. When a competence is instantiated, it results in creation of two evpath stones for processing and dispatching events. Two separate threads are created for listening for a context update and for handling competence events. The self-explanatory ECC API is shown below:

```
ECC_init();             // initialize
ECC_get_competence(); // explicit context sync
ECC_dispatch_intent();// dispatch intent
ECC_dispatch_events();// dispatch events
ECC_deinit();           // deinitialize
```

C. *Android-specific Details*

In order to expose the ECC functionality to Android’s JAVA applications, JNI based interface is implemented for ECC. Also, to allow native code to call JAVA code of applications and/or services in Android, message passing mechanism is required to avoid referencing grabage collected objects currently in use. Currently, only native instances of competences are supported in Android ECC. Android’s features to kill an unresponsive application which does time consuming activity in UI thread is circumvented by rooting the device and adding an Android system service exposing ECC.

D. *Required Competences for Usecases*

Available open source software packages are used to implement prototype competences. Specifically, we use the OpenCV libraries to implement competencies such as video decode, display, image segmentation, image descriptor extraction, and image descriptor matching. For collaborative UI, we use X11-tst and Android’s native input API. Another competence implements retrieving an image from Google StreetView for some given coordinates to show that any arbitrary functionality can be represented as a competence. All competencies are pre-deployed on the devices in the experimental testbed. We next present details of our experimental setup along with performance results and associated overheads.

VI. Evaluation

A. *Experimental Setup*

Experimental testbed comprised of a laptop with an Intel Core i7-2670QM CPU (2.2 GHz), a remote cloud participant realized by an Amazon EC2 high-CPU Medium instance (running Ubuntu 12.04) and Nexus 7 Android tablet with Jelly Bean cynogenmMod Android. Cloud access is via a commercially available 20Mbps Internet connection over WiFi.

B. *Roles of* ***Competences*** **and** ***Context***

1) Application performance improvement through context-sensitive ECC deployment: We contrast the performances of two requested intents (i.e., distributed video playback and in vehicle AR usecases) in three different contexts, including scenarios when (i) competences are offered by only a single device (i.e., the tablet), (ii) competences are offered by multiple devices (i.e., tablet and nearby laptop), and (iii) competences are offered one nearby and one remote device (i.e., the tablet and an Amazon EC2 High CPU Medium instance). We employ end to end processing time for each frame time as a metric to contrast performance and to highlight ECC deployment benefits. The evaluation of third usecase i.e. collaborative UI show similar numbers are omitted in favour of space in this paper. Figures 4 and 5 show execution performance using tablet only, with a nearby laptop, and with Amazon EC2 instance. The results of the first experiment are shown in Figures 4 (i) and 5 (i). Note that with different contexts, the mapping of competences changes, i.e., instead of choosing a competence implemented on the same device, ECC’s decision engine chooses a better competence (nearby laptop or the Amazon EC2 instance) to realize the given intent. The advantages seen from using nearby devices (to enhance user experience compared to using a single mobile device) vs. using the cloud are consistently apparent, when running these experiment over a two week period and at different times of day. Performance advantages over using only a single device hold for both classes of applications, i.e., the linear synchronous (video playback) and the branching asynchronous (AR), for average improvements of 70.42% and 86.07%, respectively, when using nearby devices, and of 58.35% and 69.66%, respectively, when using the remote cloud. Effect of unreliable internet latencies can be easily seen (in both Figure 4 and Figure 5) that can render remote clouds unusable. ECC can shield end users from such lack of reliability by using nearby devices, specifically, when seeing internet access to be unreliable, ECC simply switches

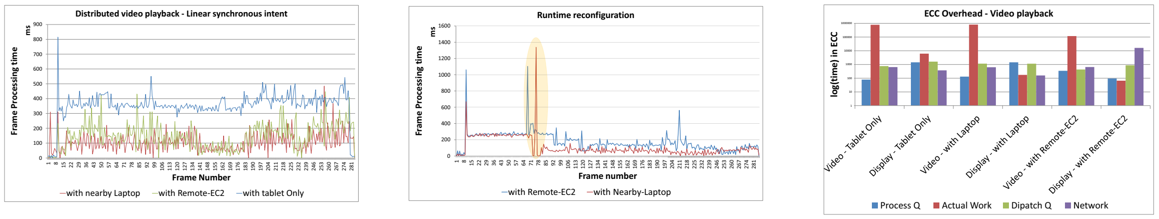

Figure 4: (i) Performance benefits in the distributed video playback (linear synchronous intent guideline) use case. (ii) ECC reconfiguration due to change in context. (iii) Total execution time - ECC's queues and actual computation.

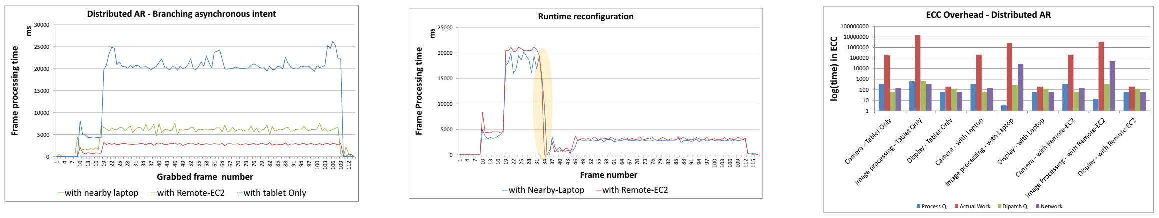

Figure 5: (i) Performance benefits in the distributed AR (branching asynchronous intent guideline) use case. (ii) ECC reconfiguration due to change in context. (iii) Total execution time - ECC's queues and actual computation.

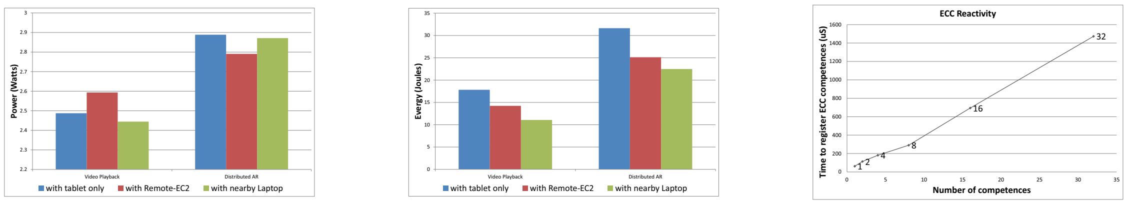

Figure 6: (i) Dynamic power utilization (ii) Energy consumption and (iii) Variation of context update delay with number of competences.

competences between remote clouds and nearby devices for the given user intent. ECC's functional formulation could be further strengthened by use of formal methods like those first described in [19], which we leave for future work.

2) Reduction in overall energy consumption: For the experiments described above, we also measure the power consumed by the tablet, using a Watts up power monitor. For both usecases, Figure 6(i) shows average dynamic power utilization (i.e., total power utilization - idle power utilization) and Figure 6(ii) shows energy consumption (power x execution time) during execution of the ECC usecases. There is no substantial difference in power utilization, i.e., differences are in within 0.1 watts, but ECC reduces total energy consumption for both classes of application, i.e., linear synchronous (video playback) and branching asynchronous (in-vehicle AR) by 37.91% and 28.92%, respectively, when using nearby devices, and by 28.92% and 20.58%, respectively, when using the remote cloud. This is due to reduced application execution time. The results indicate that using nearby devices can also serve to save the energy consumed by battery operated end user devices.

*C. Role of **'Intent'***

Use of competences and context can help gain performance improvements and energy savings, but additional information is needed to compose the set of devices needed to run some user-desired task. ECC's formulation of *"intent"* enables its decision engine to choose the 'best' possible mapping from competences to concrete instances in a current context. We highlight the importance of intent by showcasing its role below.

Performance based decisions - not sufficient. The previous experiments use a performance metric to illustrate the importance of context, but such performance based selection may not be 'best' for end users. For example, for distributed video playback (see Figure 3), consider a user desiring to display on her phone screen, but also available in the same room is a large TV display. Based on context and performance criteria only, the TV will be selected as the best competence. Intent guidelines, then, can be used to *tie* display competence to the instance available on the handheld, hence resulting in a different competence assembly. In other words and as also shown in Figure 3, different intents can result in entirely different mappings in the same context, thereby also creating entirely different user experiences. In another example, ECC leverages low latency in-home networking to always select the nearby laptop, but to offload functionality from that machine, the user can state an intent so as to force ECC to select the Amazon EC2 instance. Figures 4 (i), (ii) and Figures 5 (i), (ii) clearly show differences in performance for such alternative configurations (i.e., the nearby laptop is preferred), hence demonstrating the ability of intent specifications to control the deployment.

D. Reactivity

To demonstrate ECC's ability to deal with runtime changes in context, in a second experiment, only one device is available which results in mapping of intent to that device (i.e., the tablet with no other devices accessible). During execution, a second device is turned on (i.e., a nearby laptop or the Amazon EC2 instance), triggering a change in context by making its competences accessible. In response, ECC automatically re-configures the realized mapping to use the better competence instances now accessible, as highlighted by the ellipses in Figures 4(ii) and 5(ii). This happens seamlessly without any action required from the application. The change can also be seen to improve performance, i.e., a reduction in the end-to-end time per frame. A temporary spike, i.e., degradation, observed in Figure 4(ii) corresponds to the reconfiguration period, during which previously instantiated competences need to stop processing due to synchronous nature of processing pipeline. The spike doesn't show up in Figure 5(ii) because of asynchronous nature of pipeline utilized in the distributed AR usecase. Through additional experimentation, we verified that time taken to create and/or update a context scale linearly with the number of reachable devices and number of competences offered by devices, as shown in Figure 6(iii).

E. ECC Overhead

To analyze ECC overheads, we measure the time spent by each trigger event in competence instance queues, the time spent in communication, and the time spent in the actual implementation of competence. As clearly seen in Figures 4 (iii) and 5 (iii), which show the average times spent in the ECC code vs. that in the actual competence implementation, ECC overhead is very small. When operating only on tablet, the overhead is only 0.1% of total execution time. It increases to 3% when the competence instances are located on different devices, with a nearby laptop and with the Amazon EC2 instance.

F. Limitations

ECC's current implementation serves to illustrate the CIC notions advanced in our work, but remains limited in several ways. First, ECC applications must be programmed once as sets of competences, much like the SOA-based applications routinely run in datacenter systems. Secondly, we do not automatically capture end user intent, nor do we offer formal specification of intent or competences that can be used to prove correctness properties or performance bounds. Finally, it lacks learning capabilities in decision engine to improve its process based on user behavior.

VII. Related work

Several research efforts explore possibilities to mitigate fragmentation in the device ecosystems [1][2][3][4][7][8][9]. By introducing the CIC concepts, ECC goes a step further by providing mechanisms to express and reason about the diversity in device capabilities and supported functionality, in the end-users' tasks' requirements and their operating contexts, and, thereby, to best leverage the aggregate resources available across nearby devices and remote clouds.

Cuckoo [20], MAUI [21] and Clone cloud [22], try to leverage the availability of remote clouds to offload computationally demanding parts of applications from end-user devices. Partitions of applications to offload are automatically generated based on static analysis or runtime estimation of resource requirements. ECC's concepts extend this by also including nearby devices as possible offloading targets, and provide mechanisms to deal with the increased dynamism and diversity in the resulting eco-system.

The MUSIC middleware [23] and Interplay [24] take a component based software engineering approach to achieve the self-adaptation of applications on 'single' devices. The approaches provide semantics for implementing components and a middleware for composition of components into applications. Industry standards such as OpenMax, OpenCL, etc., also reflect this approach. ECC leverages these ideas to include 'multiple' devices in such compositions, leveraging underlying support for device integration via cloud infrastructures. ECC takes a step beyond the goal of works like Groupware [25], i.e., to go from the seamless interaction of heterogeneous devices to seamlessly collaborating devices.

Creation of cloud-like infrastructure from co-located devices has also been explored in recent projects like Pocket Cloudlets [14] and Stratus [5], which propose seamless integration of computational resources of devices to enhance the performance of unmodified applications running on participating devices. ECC employs such infrastructure for mapping its competence assemblies to physical nodes. The vision for multi-device systems has been proposed and prototyped in projects like Pervasive Collaboration [2], device ensembles [6] and Dynamic Composable Computing [3]. ECC presents a concrete realization of this vision and several accompanying use cases.

VIII. Conclusions and Future Work

This paper presents the ECC for multi-device environments. ECC offers C(ompetence)I(ntent)C(ontext) as first class concepts, in order to make it possible for end users to enhance the capabilities of their own devices with additional nearby devices and remote cloud resources. ECC-composed applications are shown to provide better performances and can also hide from users the unpredictable internet latencies inherent in remote cloud access.

However, it will be important to automate the capture and specification of user intent, possibly by translation from higher level user statements (e.g., speech). Also needed are more formal specifications of competences, along with the use of formal methods when creating competence assemblies, to prove correctness and/or other desired assembly properties (e.g., to address privacy issues), all of which will give rise to alternative or more powerful decision engines. More specific, shorter term goals of our work include (i) improved methods for device discovery, and (ii) experimentation with interfaces between ECC and underlying device cloud infrastructures.

References

[1] M. Satyanarayanan, M. A. Kozuch, C. J. Helfrich, and D. R. O'Hallaron, "Towards seamless mobility on pervasive hardware," *Pervasive Mob. Comput.*, vol. 1, no. 2, pp. 157–189, Jul. 2005.

[2] T. Pering, R. Want, B. Rosario, S. Sud, and K. Lyons, "Enabling pervasive collaboration with platform composition," in *Proceedings of the 7th International Conference on Pervasive Computing*, ser. Pervasive '09. Springer-Verlag New York, 2009, pp. 184–201.

[3] R. Want, T. Pering, S. Sud, and B. Rosario, "Dynamic composable computing," in *Proceedings of the 9th workshop on Mobile computing systems and applications*, ser. HotMobile '08. ACM, 2008, pp. 17–21.

[4] G. Schiele, M. Handte, and C. Becker, "Pervasive computing middleware," in *Handbook of Ambient Intelligence and Smart Environments*. Springer, 2010, pp. 201–227.

[5] M. Jang and K. Schwan, "Stratus: Assembling virtual platforms from device clouds," in *IEEE CLOUD*, 2011, pp. 476–483.

[6] B. N. Schilit and U. Sengupta, "Device ensembles," *Computer*, vol. 37, no. 12, pp. 56–64, Dec. 2004.

[7] S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta, "Reconfigurable context-sensitive middleware for pervasive computing," *Pervasive Computing, IEEE*, vol. 1, no. 3, pp. 33–40, 2002.

[8] V. Raychoudhury, J. Cao, M. Kumar, and D. Zhang, "Middleware for pervasive computing: A survey," *Pervasive and Mobile Computing*, vol. 9, no. 2, pp. 177 – 200, 2013.

[9] A. Ranganathan and R. H. Campbell, "A middleware for context-aware agents in ubiquitous computing environments," in *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*. Springer-Verlag, 2003, pp. 143–161.

[10] "Universal plug and play alliance (upnp)." [Online]. Available: http://www.upnp.org/

[11] "Samsung allshare." [Online]. Available: http://www.samsung.com/us/2012-allshare-play/

[12] "Microsoft xbox's smart glass." [Online]. Available: http://www.xbox.com/en-US/smartglass

[13] "Apple airplay." [Online]. Available: http://www.apple.com/airplay/

[14] E. Koukoumidis, D. Lymberopoulos, K. Strauss, J. Liu, and D. Burger, "Pocket cloudlets," *ACM SIGPLAN Notices*, vol. 47, no. 4, p. 171, Jun. 2012.

[15] C. Henson, K. Thirunarayan, and A. Sheth, "An efficient bit vector approach to semantics-based machine perception in resource-constrained devices," in *Proceedings of the 11th international conference on The Semantic Web - Volume Part I*, ser. ISWC'12. Springer-Verlag, 2012, pp. 149–164.

[16] "Resource description framework." [Online]. Available: http://www.w3.org/RDF/

[17] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, ser. SOSP '03. ACM, 2003, pp. 164–177.

[18] G. Eisenhauer, M. Wolf, H. Abbasi, and K. Schwan, "Event-based systems: opportunities and challenges at exascale," in *DEBS '09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*. ACM, 2009, pp. 1–10.

[19] X. Liu, C. Kreitz, R. van Renesse, J. Hickey, M. Hayden, K. Birman, and R. Constable, "Building reliable, high-performance communication systems from components," in *Proceedings of the seventeenth ACM symposium on Operating systems principles*, ser. SOSP '99. ACM, 1999, pp. 80–92.

[20] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones," in *Mobile Computing, Applications, and Services*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, M. Gris and G. Yang, Eds. Springer Berlin Heidelberg, 2012, vol. 76, pp. 59–79.

[21] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Ch, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *In In Proceedings of ACM MobiSys*, 2010.

[22] B. Chun, S. Ihm, and P. Maniatis, "Clonecloud: Elastic execution between mobile device and cloud," *Proceedings of the sixth conference on Computer systems*, 2011.

[23] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, and U. Scholz, "Software engineering for self-adaptive systems," B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Springer-Verlag, 2009, ch. MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments, pp. 164–182.

[24] A. Messer, A. Kunjithapatham, M. Sheshagiri, H. Song, P. Kumar, P. Nguyen, and K. H. Yi, "Interplay: a middleware for seamless device integration and task orchestration in a networked home," in *Pervasive Computing and Communications, 2006. PerCom 2006. Fourth Annual IEEE International Conference on*, march 2006, pp. 10 pp. –307.

[25] A. Markarian, J. Favela, M. Tentori, and L. Castro, "Seamless interaction among heterogeneous devices in support for colocated collaboration," in *Groupware: Design, Implementation, and Use*, ser. Lecture Notes in Computer Science, Y. Dimitriadis, I. Zigurs, and E. Gmez-Snchez, Eds. Springer Berlin / Heidelberg, 2006, pp. 389–404.