Enabling Migration Decisions Through Policy Services in SOA Mobile Cloud Computing Systems

John M. Medellin*, Osama Barack, Qiao Zhang, and LiGuo Huang
Department of Computer Science and Engineering
Southern Methodist University, Dallas, Texas 75275-0122, USA
Email: $\{jmedellin, obarack, qiaoz, lghuang\}$ @smu.edu

***Abstract*—According to some estimates, mobile applications will drive over half of the volume on the internet by the second half of the 2010s. This powerful technology is constrained by energy sources (battery), mobile spectrums and other physical characteristics. Mobile Cloud Computing (MCC) complements these devices by bringing the vast resources available in Clouds to them. Applications can be modularized and shipped for processing to these centers of greater capability freeing up computation, data requirements and ultimately energy.**

Our approach in this study is to first use SOA/BPEL/services design principles to modularize the application. Once the application is modularized we propose an architecture for a service that drives the decision of what to migrate. The application is scaled and migrated with the AppleHandoff (iOS 8.1.3) facility to provide objective results to the theory. The service architecture designed for the policy service is based on SOA principles and includes Quality of Service (QoS) and Quality of Experience (QoE) drivers identified by the user.

Keywords—Mobile Cloud Computing, Business Process Execution Language, BPEL, Services Oriented Architecture, SOA Services, Cloud Process Migration

I. Introduction

Mobile and Cloud Computing continue their march towards becoming pervasive players in all aspects of life. Cisco estimates that by 2016 more than 60% of world IP traffic will be generated by mobile devices [1] and the number of mobile multimedia users will top 800 Million by 2015 according to other estimates [2]. The global mobile application market is estimated at $9.4 BUSD in 2014 [3] [4] and will grow at a compound annual growth rate of close to 38% per year to top out at almost $47 BUSD in 2019 [4].

Cai et al [5] declare that the essential cloud computing characteristics include on-demand service, broadband network access, resource pooling and measured service. Those characteristics can be delivered to devices such as laptops and desktops that have reliable network connections, access to electrical power and sufficient computing capacity. Correspondingly, Zhang et al [6] identify battery life, network connectivity and computational power as being the three largest challenges to mobile computing environments.

A first critical limitation to these environments is battery life. While computing capabilities for mobile devices have been significantly improving over time, battery life has not kept up and has only been improving at a rate of 5% per year, far below the improvements in computing power [4]. As computer chips gain in computational ability they also increase their voracity for energy, nothing is for free. In order to preserve resources and battery life, execution must be offloaded from the hand held device [1] in order to preserve as much energy as possible.

This paper proposes the usage of a centralized service on the mobile device containing user preferences, description of mobile and cloud plans and a history repository in order to make the decision to migrate (or not) and what to migrate to the Cloud.

II. Motivation

One of the great promises of the Cloud revolution is the ability to unleash the vast resources of this architecture to supplement the smaller capabilities of mobile devices. Processes can be migrated as entire applications, methods or threads [7] and tradeoffs have to be evaluated on what to migrate. Users of mobile devices are mostly concerned with receiving output as quickly and inexpensively as possible.

A. Current approaches to modularization

Several methods exist for modularizing mobile applications for migration to the cloud. They range from entire application migration to replication with state transfer and finally some propose to modularize based on Services Oriented Architecture/BPEL concepts.

Modularization at the application level has been widely used by most Cloud providers. In this pattern, the function call is issued to a remote URL with some parameters passed to it. The cloud executes the required functionality and returns a payload to the device. The UI logic of the application displays the output to the user. In this particular case, most of the application is resident in the cloud and very little decision making is left to the device itself (it either uses the logic on the cloud or it does not for most of the functionality requested).

A second pattern replicates the application on both the device and the cloud and annotates the code at certain points where, according to developer judgment, an exchange of state is feasible between the two environments (to continue on the Cloud, or resume on the device). The compiler exposes those particular partitions and exchange of data happens at those junctures. The exchange of data occurs as a function of a mobile enabled Cloud offering through the cloud vendor. This exchange does not have to happen on the entire state, some

significant optimizations have been proposed by Yang et al [8].

Our approach proposes the use of Services Oriented Architecture Business Process Execution Language (BPEL). BPEL is a formal semantic language that allows the analyst to express business process concepts in both sequential and parallel tasks. It assumes the invocation of Services by means of contracts requiring only what is needed by the service to execute and what it will return as output. This natural granularity (only the input needed or output return) forms the basis for a smaller set of data transfer between functions being executed. BPEL is based on business process decomposition. In this technique, macro business processes (e.g., invoicing) are decomposed into sub-processes until they can be encapsulated into standardized services (e.g., vendor master data maintenance). BPEL is a recommended tool for implementation of Services Oriented Architecture (SOA) design patterns and has become more prevalent since introduction of the 2.0 standard in 2007 [9].

B. The decision to migrate is complex

The decision to migrate an application or portion of it to the Cloud for execution is complex because some of the factors are not known. Known factors include the amount of battery remaining and the connectivity scheme (WiFi or mobile). Some of the more important unknown factors include:

1) Whether the entire application needs to be migrated or if certain portions can be migrated and what those portions are.
2) The amount of CPU cycles (and therefore battery) that is required to encapsulate all or portions of the application.
3) The mobile data/roaming plan economics of the user and the current level of consumption of base charges.
4) The encapsulation and round trip (to the Cloud and back) latency of the application under the current spectral scenario.
5) The amount of CPU cycles the application will consume in computation and data intensive tasks if left to execute on the device.

The objectives of this document are to investigate the implications on partitioning the application at the service contract level and to propose an architecture to enable decisions on what portions (if any) to migrate. We begin this discussion by presenting prior work applicable to modularizing and migrating of objects in MCC. The key areas reviewed are research on SOA and the services contract, MCCs heterogeneous environments, various methods for offloading while conserving battery, and tradeoff evaluations for offloading.

Next we build a hypothetical MCC business application that is created under a SOA/BPEL script and contains four services. The data and computation service contracts are analyzed versus the service operations themselves; scenarios for partitioning at the contract and other areas of the application are analyzed.

Finally, a services architecture is presented that accumulates historical, user preference and empirical data in order to facilitate the decision of what services to execute in the Cloud. The application and service architecture operates in a mix of handheld, laptop and cloud resources and is scaled in order to determine the feasibility of making service contract based partition decisions.

This paper leverages SOA/BPEL and service contracts to modularize applications and recommends a services architecture to assist in the migration of all or portions of that application. The two contributions that are made by this research are answers to the following questions:

- Where are the most efficient points to partition applications that have been designed using SOA/BPEL/services principles?
- What is a potential service architecture that could be used to accumulate data parameters relevant to making the migration decisions in the same application?

III. Related Work

Most of the research in this domain has been conducted in networks and devices. That research has been focused on uncovering significant issues and potential approaches to offload computation under the constraints of unstable network connectivity and energy usage.

A. Heterogeneity and its problems

The MCC domain is fertile with issues related to heterogeneity, very little has been created to enhance attributes such as portability or transferability in these applications. In very similar discussions Shiraz et al [7] and Sanaei et al [10] identify heterogeneity in MCC as a major stumbling block in achieving the vision of delivery of similar capability to the mobile device as can be delivered to the desktop from the Cloud.

Both Shiraz and Saneti discuss a variety of networks available to overcome the issues of heterogeneity. MAUI, CloneCloud and COMET are discussed as three tools that are able to implement computing migration in different ways. MAUI focuses on application migration, CloneCloud on method level migration and finally, COMETs focus in on thread-based migration. Examples are provided for migration under each discussion and are very much dependent on the Android hand held device and intel-based proprietary Clouds.

B. Offloading Tradeoff Analysis

A key decision to be made is if the application/method-/thread should be migrated to the Cloud. Yang et al [8] propose models for offloading based on acyclic graphs of processes under execution. This approach assumes that an application/method/tread either can or cannot be modularized to be migrated. If it cannot, then the migration decision is binary, it either is or is not migrated and certain formulas based on historic prediction are provided. If the application can be modularized, then the analysis focuses on which portions to be migrated based on the acyclic graph (sequential, hierarchic or mesh). The offloading algorithm measures the tradeoff between analysis and migrating versus executing the process on the device.

C. Modularizing and Efficient Transfer

Yang et al [8] provide a very insightful study on application migration using CloneCloud. In that paper, they identify modularizing approaches as static and dynamic. Static application modularizing requires the developer to annotate the application code into partitions that are able to be migrated as parts of a whole. The compiler uses these annotations to direct where the code should be executed (mobile or cloud) as the application is built. The approach is viable where all aspects of the mobile app can be predicted (e.g., how much data, where executed, device) which is typically not the case in MCC applications without a SOA architecture. In the alternative dynamic approach, the decision of where to locate the execution those partitions is made as the application is processing. They indicate that due to the nature of MCC (as described in other sections of this documents) and the lack of predictability in the environment, this is perhaps the best approach to take in modularizing. Shiraz et al [7] further indicate that the dynamic approach seems to be the prevailing approach to modularizing.

Yang et al also proceed to propose algorithms to reduce the transfer of stack frames and heap objects within CloneCloud and achieve a 97% reduction in transferred data. This is impressive and a very good example on how certain frameworks can be further optimized. We believe these types of algorithms and approaches will be implemented by the frameworks as they further mature into this space.

D. Corporate Mobile Applications

Connectivity and battery are key determinants to application success. Cox et al [11] and Privat and Warner [12] have developed several methods for industrializing Apple iPhone Apps which can further be deployed to modularize applications and distribute them for commercial purposes. Their frameworks provide a certain level of resiliency and application survivability that is necessary in commercial applications. They base their methodology on developer analysis and intuition on how to modularize and how to migrate. Most of this analysis is done on the static code base.

E. Cloud Vendor-specific MCC Tools

Most of the significant Cloud vendors also provide toolkits for development and optimization of MCC on their offering suites. We do not address those particular frameworks in this document due to potential Cloud vendor lock-in.

F. SOA, BPEL and services contracts

Erl [13] defines SOA as: Service-oriented architecture represents an archtiectural model.It accomplishes this by positioning services as the primary means through which solution logic is presented

Many organizations have also adopted a language called BPEL to enable orchestration of services in support of execution of business processes [9]. The objective of BPEL is to provide a structured language where business process operations are decomposed until they can be syntactically and semantically executed by services. The services are orchestrated by a workflow server and enable the execution of a particular process by sequencing them in a prescribed manner. These services can also be reused in other business processes provided their context requirement is the same (for example, the vendor data service can be used to verify a vendor address in order to mail a check or can also be used to verify the tax identity number in a tax filing process).

BPEL has its own structure and hierarchy for defining actions and agents that enable transactions in processes to be executed. This document uses the BPEL 2.0 for defining the process to be modeled by the MCC application [9].

As discussed above, a major component of the SOA architecture are services. Services have two fundamental components; a contract and a set of methods (logic to perform their tasks). The contract segment of the service is exposed to those needing to use it while the logic of the operations being performed is hidden from the requester. The vision for SOA is a reusable set of services that can assist in composing or extending existing functionality required by the enterprise [14].

IV. Experiment Setup

The objective of this experiment is to provide the feasibility for modularizing applications based on BPEL and services analysis.

A. Business Process Being Modeled

World currency exchange rates are set on a variety of parameters including the debt of the country, the trade balance with the other countrys economy, the relative stability and other macroeconomic factors. Sometimes the currencies are in parity (meaning that if one was to exchange US Dollars (USD) for Euros (EUR) at a given rate or instead bought first British Pounds (GBP) and then EUR with them the operation would yield the same monetary effects; with fees held constant). In practice this is not the case and the disparity between the currencies is a daily aspect of world markets and connerstones of trading desks, currency hedging services and other firms.

The scenario that is being modeled is one in which a company has to pay an invoice of a given amount in a currency that is not its own. The company is interested if it should go do a direct exchange (e.g. USD to EUR) or if there is economic advantage by triangulating (e.g. USD to GBP to EUR). The user keys in the amount, selects the target currency, the source currency and the triangulating currency. The mobile application outputs how much it would cost for a direct exchange vs a triangulated exchange as in Figure 1 below:

In the above example, it is more expensive to pay a direct exchange of US Dollars to Euros ($113.25USD * 0.883=100EUR) versus triangulating through the British Pound ($113.22USD * 0.6639 GBP *1.13304=100EUR). This scenario assumes no currency fees or other factors affecting the currency exchange (for example, this could be a multinational which has checking accounts in USD or GBP and needs to pay in EUR with its main currency being the USD).

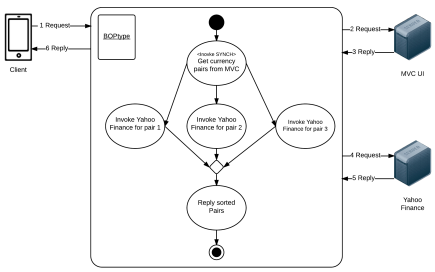

Fig. 1. Triangulated vs Direct Currency Analysis.

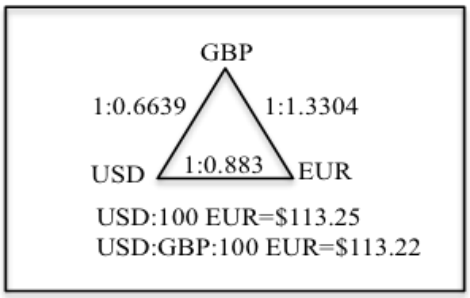

Fig. 2. UML Activity Diagram for Currency Analysis.

B. Application Modeling with BPEL, SOA and services

A best practice for modeling BPEL is through the UML Activity Diagram [15]. The following activity diagram would represent the BPEL process/method for the currency analysis described above and is an adaptation of the UML Activity Diagram:

An excerpt of the corresponding BPEL for that process is in Listing 1 as follows:

```
L#    BPEL Statements
--  ------------------------------------
01 <process>
02 <process name= CurrencyCalculation
   >Start identification of service<
03 <partnerLinks>
04       <partnerLink name= ratePair
05        partnerLinkType= rt:ratePair
06        myRole= rateService
07        partnerRole= rateServiceExchange
08 </partnerLinks>
   >End identification of service<
   >Start contract variables<
09 <variables>
10  <!-input currency source->
11       <variable name= currencySource
12        messageType= fromMessage />
13        <!-input currency target->
14       <variable name= currencyTarget
15        messageType= toMessage />
16        <!-input pair exchange rate->
17       <variable name= exchangeRate
18        messageType= rt:pairExchange />
19       <variable name= invoiceAmount
20        messageType= invAmt >
21 </variables>
   >End contract variables<

22 <sequence>
   >Start invocation of retrieval service<
23       <!-retrieve currency pairs->
24       <!-First module to encapsulate->
25       <receive partnerLink= ratePair
26       portType= rt:retrieveRate
27       operation= pairExchangeRate
28       variable= currencySource
29       variable= currencyTarget
31       variable= exchangeRate
32       createInstance= yes />
   >End invocation of retrieval service<
33 (. . . . )
34 </sequence>
35 (. . . . )
36 </process>
```

The above BPEL declares the process, identifies the service to be invoked, contract variables and invokes the service to retrieve rates. The additional BPEL statements for the other services and contracts (get variables, identify lowest value combination and display results) are omitted.

C. Cloud/Application Architecture

An application using the Apple SWIFT [16] coding environment was constructed to execute the triangulation analysis. The application uses a source currency, a target currency, and a target payable amount as inputs. The application triangulation currencies are driven from an array that is populated by the user. The Apple infrastructure allows the ability to handoff an activity from one AppleApp to another device operating the same AppleApp through their iCloud infrastructure [17]. Apple also provides the facility to notify the user if a pattern of CPU, Energy or Data Plan usage is detected so it can choose a hybrid mode of execution (with offloading). That algorithm can be created and implemented through the analysis of patterns in the App trace file with the Apple Instruments package. The following figures illustrate the application architecture of the system that was constructed (the iPad has an electrical connection removing the battery consumption constraint from that machine). They are presented as a static view (allocation) and sequence (dynamic) views. The allocation view illustrates the residence of key components on devices/services while the sequence view illustrates the key steps in execution of the application scope.

Four activities were created inside the application. Two of them were modeled to work with the AppleHandoff utility [18].

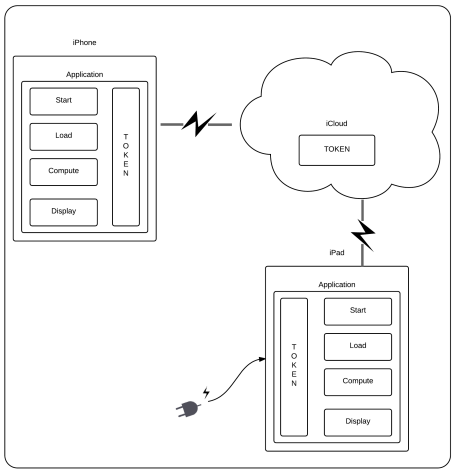

Fig. 3. AppleApp System Architecture: Allocation View.

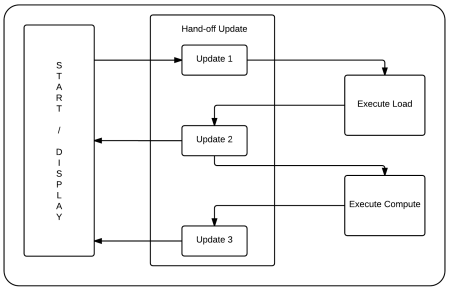

Fig. 4. AppleApp System Architecture: Dynamic View.

The activities modeled were (web queries and calculations can be handed off to a remote iPad or a Mac connected through iCloud):

- Parameter setup
- Web queries
- Calculations
- Return result

Figure 4 depicts the execution sequence and the update of the state token after each method is executed.

D. Technical Infrastructure

The application was developed with the following technical environment:

- Apple Macintosh working on OSX 10.10.2 Yosemite Operating System blue tooth 4.0
- Apple iPhone 6, operating on iOS 8.1.3 Mobile Operating System blue tooth 4.0
- Apple iPadMini 3, operating on iOS 8.1.3 Mobile Operating System blue tooth 4.0
- Apple XCODE 6.1.1 with SWIFT version LLVM compiler, Apple Instrument Package
- Synchronization via iCloud WiFi Ethernet

TABLE I
COMPUTATION OF LOAD BY COMPONENT OF APPLEAPP.

Item	Assumption	Type	Value
Input Currencies	1 array passed	Message	1
Contract 1	1 array passed	Message	1
Retrieve	603 query/parse	Mixed	603
Contract 2	1 array passed	Message	1
Compute 3	computations/point	Computation	603
Contract 3	1 array passed	Message	1
Output Results	1 array passed	Message	1

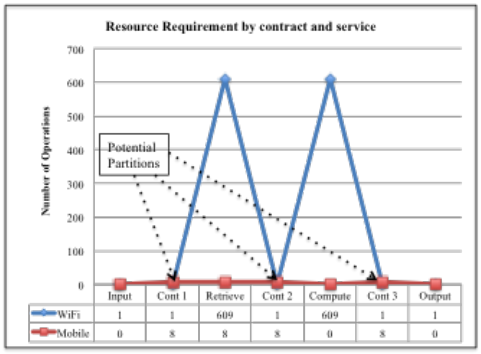

Fig. 5. Hypothetical distribution of load in AppleApp.

V. EXPERIMENT EXECUTION AND RESULTS

The application was scaled to 100 currencies to be triangulated in order to identify the differences in data requirements between the contracts and the services themselves.

A. Estimating volume of operations by each service contract and service

The service contracts remain constant during the execution of the applications. One array with input values for the currency pairs to be retrieved is passed to the retrieval service contract and one array with the currency pair exchange rate results is passed to the calculation service. The size of these arrays increases by three characters for the initial array and by 4 characters for the second array for every additional currency to be evaluated. This is a modest increase when compared of the incremental call to one web function, passing of parameters, receiving reply, parsing for values and storing the currency pair values in the retrieval service. It is also modest when compared to the triangulated calculation to compute the computation of load components is given in Table I.

A hypothetical resource graphic is included in Figure 5 to identify the relative size of computational/data requirements by each service and service contract. The graphic identifies the contracts as lowest points of transfer between the services. These points were used to partition the application.

B. Experiment output when triangulated to 100 currencies

The experiment was scaled to evaluate 100 currencies for triangulation. The application was able to be modularized and each module measured according to the attributes of energy, cpu and IO. The IO and computational characteristics of the

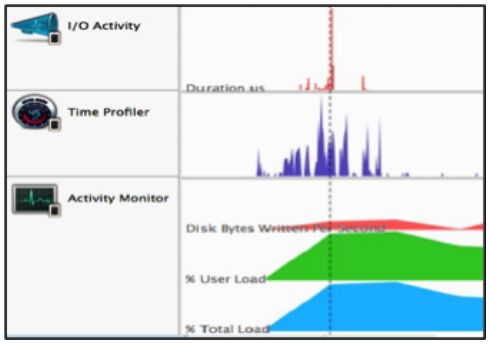

Fig. 6. Key Execution Characteristics.

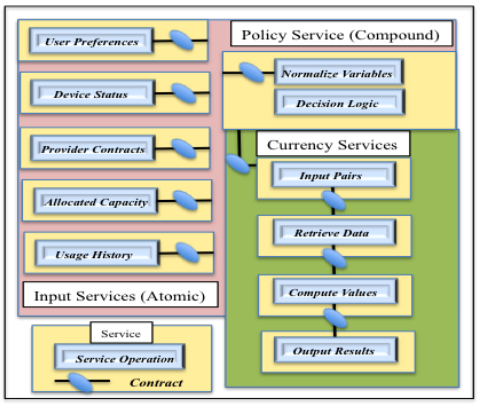

Fig. 7. Proposed Migration Service Architecture.

application volumes and metrics were captured through the use of the Apple Instruments and key results are shown in Figure 6.

The above instruments quantify impacts as follows:

1) IO Activity: captures information about I/O events, such as reads, writes, file open and closes.
2) Time Profiler: usage of CPU with time-based sampling of the application.
3) Activity Monitor: traces the overall system load along with the application running.

IO activity is significantly impacted when the queries to the web are executed to retrieve exchange rates (the retrieve service). Time profiler is again significantly impacted when the calculations are executed to derive the low cost alternative (the calculate service). The activity monitor reveals that most of the load in the application is being driven by the application and no extraneous factors. The dotted line is the flag where the initialization of application finished and it also identifies the peaks among all three of the measurement instruments (during the IO partition of the App).

VI. Architecture of Policy Services to assist in Migration Decision

As mentioned above, the migration decision is dependent on known and unknown factors. The proposed architecture gathers user, contract, current state, forecasted usage and historical metrics to drive a decision of what to migrate.

A. Proposed services architecture

A services architecture was created to facilitate the decision of what to migrate for execution to the Cloud. This architecture is illustrated in Figure 7 and consists of atomic services that assist in providing key decision variables to the centralized policy service. The policy service implements the migration decision and if it does allow for migration of all or parts, will allocate future capacity to execute those particular tasks. Figure 7 also includes the application designed and implemented above for illustration purposes.

B. Atomic services context

Table II illustrates the context of services that are proposed and the operations each will perform in support of the policy compound service. The user preferences and provider contracts services are populated from input by the user on the device. The device status is populated by querying the status (general) App on the mobile. Finally, the allocated capacity and usage history services are maintained by the services themselves from prior decisions and trace files of the apps on the device.

These variables are used in the decision making process of the compound migration service.

TABLE II
Services and Contexts.

Service	Context Provided
User Preferences	Tradeoffs and tolerances
Device Status	Usage, remaining resources
Provider Contracts	Mobile contract and Cloud contract
Allocated Capacity	Estimated usage for current tasks
Usage History	Prior executions and characteristics

C. Migration decision logic

The migration decision service logic operates on the following three equations:

Let:
EQ1: $\{\forall p \&\& \forall h : \exists w, \exists c \equiv u\}$
EQ2: $\{\forall p \&\& \forall h : \exists m, \exists c \equiv u\}$
EQ3: $\{\forall p : \nexists h\}$
Where:
p=partition, h=history,
w=wifi, u=user preferences
c=capacity, m=mobile spectrum

The decision to migrate the service is true in any of the above three equations. In EQ1 there exists a valid partition with prior migration history, there is also a WiFi spectrum and enough capacity (latency, battery/cycles) on the device to transfer to the Cloud for a round trip that is equal to the ranges defined in the user preferences. In EQ2 there is also a valid partition with prior migration history but there is a mobile spectrum with enough capacity (cost, latency, battery/cycles) on the device plans to transfer to the Cloud within tolerable

ranges established in the user preferences. Finally in EQ3, there is a partition but no history and the decision is to migrate in order to begin the history creation process (assumes at least one execution will be needed to create the equation variables). In all other cases, the partition is executed on the device.

VII. Discussion / Potential Risks

Perhaps the two most complex aspects of migration are what to migrate and if the migration should occur. The objective is always to minimize impact and maximize usage of low cost computing capacity. This document has offered a method for partitioning based on services contracts and a method for evaluation of the migration decision based on historical and environmental factors.

A. Potential risk: dynamic binding.

In most SOA environments, binding of services to processes is performed as the execution occurs. Commercial vendor offerings allow for modification of execution scripts on-the-fly while they are being executed. In our particular architecture, the services are bound and analyzed by the instruments and are not able to be modified without re-compilation. This potential limitation may inhibit flexibility similar to that offered by the traditional SOA Web Services which allows for changes up to the point of dynamic binding at execution.

B. Potential risk: broad applicability of BPEL.

The BPEL 2.0 standard has now been in existence almost 8 years. However, this particular language is far from universal in adoption within the business community. The language is complete for the most common processes but still might need additional maturity in some of the less frequently used. In addition, some methodologies challenge its usage and contend that such a rigorous process may constrain delivery of code and functionality on a timely basis (in Agile development for example).

Our objective in using BPEL was to provide an alternative to acyclic and semantic code analysis in support of migration to reduce load on the hand held device. Certainly other languages and methods exist to analyze required functionality and determine where the least amount of state data is required for transfer between the environments. The objective is to minimize that state transfer and potential latency while conserving energy on the device.

VIII. CONCLUSIONS AND FUTURE WORK

A. Conclusions

This document explored the usage of BPEL in partitioning applications for migration and executed an experiment to review if the service contract was a potentially efficient segment to do so. The data from the services in the experiment support that conclusion.

A second contribution was the definition of an architecture to store preferences, history and state information so that the decisions to migrate can be made under certain variables. That architecture is itself a service and is being modeled in apps on the environment described above. At the point of this writing, those applications are under development.

B. Future work

This second study in modularization continues to identify the service contract as a potentially optimum location for partitioning applications for migration to the Cloud. The migration decision however is not as clear-cut. The proposed service architecture for the composite service is in process of being implemented in the Apple SWIFT environment. Additional field work will be required to demonstrate to what extent the variables and architecture are viable for delivering efficient decisions to requesting applications when they are ready to decide what to migrate.

ACKNOWLEGEMENTS

This research was supported by the U.S. National Science Foundation (NSF CNS Award 1126747).

References

[1] F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, and B. Li, "Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications," *Wireless Communications, IEEE*, vol. 20, no. 3, pp. 14–22, 2013.

[2] M. Felemban, S. Basalamah, and A. Ghafoor, "A distributed cloud architecture for mobile multimedia services," *Network, IEEE*, vol. 27, no. 5, pp. 20–27, 2013.

[3] www.finance.yahoo.com/news/research-markets-global-mobile cloud.

[4] X. Ma, Y. Zhao, L. Zhang, H. Wang, and L. Peng, "When mobile terminals meet the cloud: computation offloading as the bridge," *Network, IEEE*, vol. 27, no. 5, pp. 28–33, 2013.

[5] Y. Cai, F. R. Yu, and S. Bu, "Cloud computing meets mobile wireless communications in next generation cellular networks," *Network, IEEE*, vol. 28, no. 6, pp. 54–59, 2014.

[6] W. Zhang, Y. Wen, J. Wu, and H. Li, "Toward a unified elastic computing platform for smartphones with cloud support," *Network, IEEE*, vol. 27, no. 5, pp. 34–40, 2013.

[7] M. Shiraz, A. Gani, R. H. Khokhar, and R. Buyya, "A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 3, pp. 1294–1313, 2013.

[8] S. Yang, D. Kwon, H. Yi, Y. Cho, Y. Kwon, and Y. Paek, "Techniques to minimize state transfer costs for dynamic execution offloading in mobile cloud computing," 2014.

[9] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0 OS.html.

[10] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: taxonomy and open challenges," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 1, pp. 369–392, 2014.

[11] J. Cox, N. Jones, and J. Szumski, *Professional IOS Network Programming: Connecting the Enterprise to the IPhone and IPad*. John Wiley & Sons, 2012.

[12] M. Privat and R. Warner, *Pro Core Data for IOS: Data Access and Persistence Engine for IPhone, IPad, and IPod Touch*. Apress, 2011.

[13] T. Erl, *SOA Design Patterns with Foreword by Grady Booch; The Prentice Hall Service-Oriented Computing Series from Thomas Erl*, 6th ed. SOA Systems, Inc., Prentice Hall Publisher, 2009.

[14] T. Erl, C. Gee, P. R. Chelliah, J. Kress, H. Normann, B. Maier, L. Shuster, B. Trops, T. Winterberg, C. Utschig *et al.*, *Next Generation SOA: A Concise Introduction to Service Technology & Service-Orientation*. Pearson Education, 2014.

[15] www.oracle.com/technetwork/articles/matjaz-bpel1 090575.html.

[16] B. G. Pitre, *Swift for Beginners: Develop and Design*. Pearson Education, 2014.

[17] A. Freeman, "Pro design patterns in swift," 2015.

[18] T. C. S. D. J. G. M. K. C. L. V. N. R. N. C. R. A. T. C. W. N. W. J. W. S. Azarpour, R. Rendon Cepeda, *iOS 8 by Tutorials; Learning the New iOS8 APIs wih SWIFT*. Razerware, LLC, raywenderlich.com, 2014.