A Decision Model for Cyber-Foraging Systems

Grace A. Lewis

Carnegie Mellon University

127 PUBLICATIONS **1,243** CITATIONS

SEE PROFILE

Patricia Lago

VU University Amsterdam

202 PUBLICATIONS **2,042** CITATIONS

SEE PROFILE

Paris Avgeriou

University of Groningen

214 PUBLICATIONS **2,217** CITATIONS

SEE PROFILE

A Decision Model for Cyber-Foraging Systems

Grace A. Lewis*‡, Patricia Lago*, Paris Avgeriou†
*VU University Amsterdam, the Netherlands
E-mail: {g.a.lewis, p.lago}@vu.nl
†University of Groningen, the Netherlands
E-mail: paris@cs.rug.nl
‡Carnegie Mellon Software Engineering Institute, USA

Abstract—**Cyber-foraging is a technique to enable mobile devices to extend their computing power and storage by offloading computation or data to more powerful servers located in the cloud or in single-hop proximity. While there is a large amount of research in this area, the reality is that there are not many deployed, operational cyber-foraging systems. As these systems become more prevalent due to their proven benefits, combined with the emergence of micro data centers and edge clouds, a need will arise for guidance on their architecture and development. To provide this guidance, we present a decision model based on a mapping of functional and non-functional requirements for cyber-foraging systems to a set of architectural tactics. The decision model was validated by developers to obtain an expert opinion on its correctness and usefulness for guiding the architecture, design, and evolution of cyber-foraging systems that meet their intended functional and non-functional requirements, while understanding the effects of decisions.**

I. Introduction

Cyber-foraging is a mechanism that leverages cloud servers, or local servers called surrogates, to augment the computation and storage capabilities of resource-limited mobile devices while extending their battery life [1]. There are two main forms of cyber-foraging. One is computation offload, which is the offload of expensive computation in order to extend battery life and increase computational power. The second is data staging to improve data transfers between mobile devices and the cloud by temporarily staging data in transit on intermediate, proximate nodes. While cyber-foraging can take place between mobile devices and cloud resources, our focus is on systems that use intermediate, proximate surrogates.

There is a large amount of research in cyber-foraging, but the reality is that there are not many deployed, operational cyber-foraging systems. In previous work we conducted a systematic literature review (SLR) on architectures for cyber-foraging systems [2] [3]. One of the main findings was the lack of focus on system-level concerns that are necessary when moving from experimental prototypes to operational systems. Given the promising results of cyber-foraging in terms of energy efficiency, reduced latency, and increased availability, combined with the emergence of cloudlets, micro data centers, and edge clouds [4], the need for cyber-foraging systems will arise from industry and government, along with a need for guidance for system architects and developers. In order to start providing this guidance, the common design decisions present in the cyber-foraging systems identified in the SLR were codified into functional and non-functional architectural tactics [3] [5]. In addition, we developed a characterization of the usage domains and contexts that benefit from surrogate-based cyber-foraging, defined in terms of functional and non-functional requirements [6].

This paper builds on our previous work and presents a decision model based on the mapping of functional and non-functional requirements for cyber-foraging systems to the set of architectural tactics, and the dependencies between tactics. The goal of the decision model is to provide guidance for the architecture and evolution of cyber-foraging systems that meet their intended functional and non-functional requirements, while understanding the effects of decisions. Section II presents the mapping of the problem space to the solution space. Section III shows how to use the model. Section IV presents the details of the decision model. Section V describes the validation of the decision model. Section VI presents related work. Finally, Section VII concludes the paper.

II. Mapping the Problem Space to the Solution Space

The creation of a decision model involves mapping elements of the problem space to elements of the solution space. In software architecture and design, the problem space is commonly represented as a set of requirements and the solution space as a set of design elements [7]. For the development of a decision model for cyber-foraging systems, we represent the problem space as a set of functional and non-functional requirements, and the solution space as a set of architectural tactics, as shown in Figure 1. An single-headed arrow between a requirement and a tactic signifies that the tactic can be used to satisfy the requirement, as shown by the *satisfies* relationship in the figure. All architectural decisions have benefits and tradeoffs [8]. The benefits of using a tactic are represented with a plus sign (+) followed by the promoted system quality. The tradeoffs of using a tactic are represented with a minus sign (-) followed by the system quality that is negatively affected. A plus/minus sign (+/-) indicates the potential for both a positive and negative effect.

To represent that a tactic could be used in combination with another tactic to address tradeoffs, we use a line with a double-headed arrow between tactics, as shown by the *complements* relationship in Figure 1. It is qualified in the same way as the *satisfies* relationship. If the use of a complementary tactic improves a system quality beyond the original tactic, or affects the system quality negatively beyond the original tactic, this is represented by a double plus sign (++) or a double negative sign (- -), respectively. If there are conditions that have to be true for the tactic to effectively satisfy the requirement, or for a tactic to complement another tactic, these are represented as constraints connected to the *satisfies* or *complements* relationship with a dashed line. When a tactic complements another tactic it means that the initial tactic is required. Therefore, the qualities of using the initial tactic also apply to the combination of the tactics. If a system quality is associated to both the initial tactic and the complementary tactic but with a different qualification, the qualification of the complementary tactic overrides the qualification of the initial tactic. To represent that there are several tactics that could complement a tactic, or could satisfy a requirement, and lead to the same result, we use the label *[alternatives]* to qualify the *complements* or *satisfies* relationship.

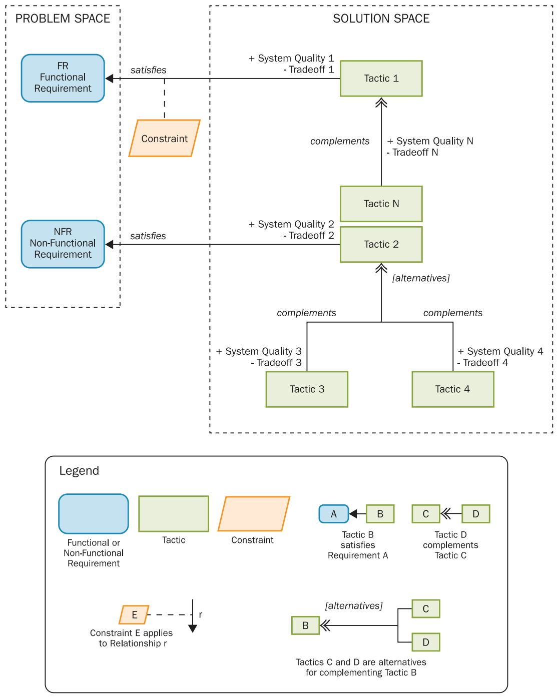

Fig. 1. Decision Model Notation

III. How to Use the Decision Models

Cyber-foraging systems have at a minimum the following combination of functional requirements [3], which map to the first four tactic selection processes in Figure 2.

- A need for computation offload, data staging, or both
- A need to provision a surrogate with the offloaded computation or data staging capabilities
- A need for the mobile device to locate a surrogate at runtime

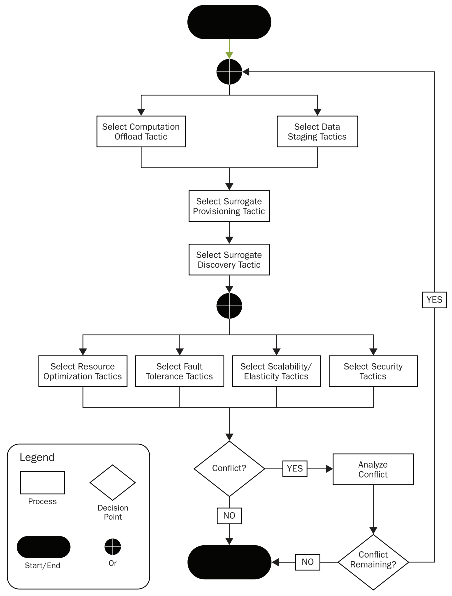
Select Computation Offload Tactic
Select Data Staging Tactics
Select Surrogate Provisioning Tactic
Select Surrogate Discovery Tactic
Select Resource Optimization Tactics
Select Fault Tolerance Tactics
Select Scalability/ Elasticity Tactics
Select Security Tactics
Conflict?
YES
Analyze Conflict
NO
Conflict Remaining?
NO
YES
Legend
Process
Decision Point
Start/End
Or

Fig. 2. How to Use the Decision Models

Then, based on additional functional and non-functional requirements, such as fault tolerance, resource optimization, scalability/elasticity, and security, complementary tactics are selected. As tactics in the decision models are combined, it is possible that system qualities are affected positively by one tactic and negatively by another. This is represented by the *Conflict?* decision point in Figure 2. This conflict needs to be analyzed to determine if the positive effects offset the negative effects, or if there is indeed a conflict. For example, if the negative effect on availability of using one tactic is because a surrogate may become disconnected (unreachable), and the positive effect of another tactic on availability is that it provides mechanisms for continuing operation when a surrogate is not available, then they offset each other (i.e., one tactic addresses the specific shortcoming of the other). However, if the positive effect of the second tactic on availability is that it enables surrogates to recover from failure, then the tactic is not addressing the specific shortcoming of the first tactic. In this case there is a conflict. Although out of scope fo this paper, architecture evaluation techniques such as ATAM [8] could be used to further understand the resulting effect of the combination of tactics. If there is a conflict, the architect should look for additional tactics, or components outside of the tactics, to address the shortcomings. The following section contains the decision models for tactic selection.

IV. Decision Models for Cyber-Foraging Systems

A. *Computation Offload*

The **Computation Offload** tactic enables mobile devices to offload expensive computation to surrogates. The decision model in Figure 3 starts from a functional requirement that a mobile system has a computing requirement where the cost to execute the computation locally on the mobile device is greater than the cost to send and execute the computation on a surrogate. The result is *increased computing power* and *increased energy efficiency*. However, this tactic is based on an always-offload strategy, meaning that computation is always offloaded to a surrogate and never executed locally. This can lead to *reduced availability* because the computation will only execute if a surrogate is available. It can also lead to *reduced resource efficiency* because even though executing the expensive computation on a surrogate leads to energy efficiency, changing network conditions might cause greater resource consumption [9].

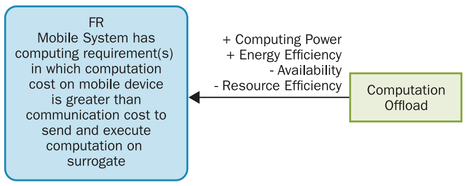
FR
Mobile System has computing requirement(s) in which computation cost on mobile device is greater than communication cost to send and execute computation on surrogate
+ Computing Power
+ Energy Efficiency
- Availability
- Resource Efficiency
Computation Offload

Fig. 3. Decision Model for Computation Offload

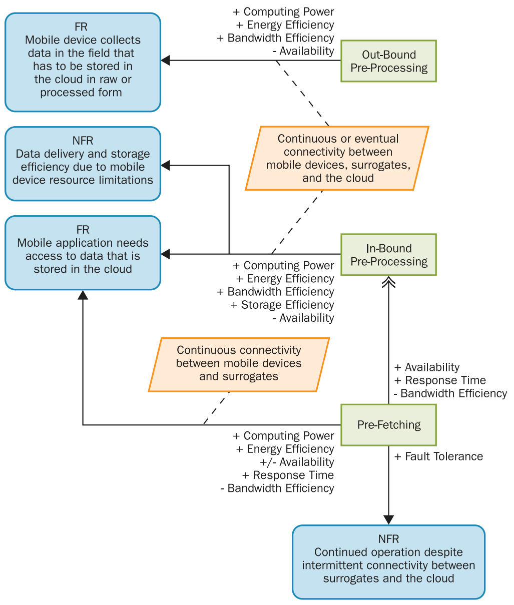
FR
Mobile device collects data in the field that has to be stored in the cloud in raw or processed form
+ Computing Power
+ Energy Efficiency
+ Bandwidth Efficiency
- Availability
Out-Bound Pre-Processing
NFR
Data delivery and storage efficiency due to mobile device resource limitations
Continuous or eventual connectivity between mobile devices, surrogates, and the cloud
FR
Mobile application needs access to data that is stored in the cloud
In-Bound Pre-Processing
+ Computing Power
+ Energy Efficiency
+ Bandwidth Efficiency
+ Storage Efficiency
- Availability
Continuous connectivity between mobile devices and surrogates
+ Availability
+ Response Time
- Bandwidth Efficiency
Pre-Fetching
+ Computing Power
+ Energy Efficiency
+/- Availability
+ Response Time
- Bandwidth Efficiency
+ Fault Tolerance
NFR
Continued operation despite intermittent connectivity between surrogates and the cloud

Fig. 4. Decision Model for Data Staging

B. Data Staging

Figure 4 presents a decision model for selecting data staging tactics. The **Out-Bound Pre-Processing** tactic enables mobile devices to collect data in the field, which is then stored on surrogates that can pre-process the data before sending it to the cloud. This tactic *increases computing power* in the form of greater storage and data processing capabilities on the surrogate. It also provides *increased energy efficiency* because of the energy savings from using WiFi or short-range radio to connect to the surrogate instead of broadband wireless to connect to the cloud [10]. Finally, it provides *increased bandwidth efficiency* because the surrogate can clean, filter, or summarize data before sending it to the cloud. However, *availability is compromised* when systems require continued or eventual connectivity between mobile devices and surrogates, and between surrogates and the cloud, to function properly.

The **In-Bound Pre-Processing** tactic enables a mobile device to access data that is stored in the cloud via an intermediate surrogate. The data received from the surrogate is pre-processed such that it is ready to be consumed, or filtered such that it is only data of interest or relevance. It provides *increased bandwidth and storage efficiency* because it enables the surrogate to control the amount of data received by the mobile device. It *increases computing power and energy efficiency* as in the Out-Bound Pre-Processing tactic, and in addition by (1) avoiding direct communication to the cloud for every data operation, and (2) processing data for visualization on mobile devices on the surrogate instead of the mobile device. However, similar to the Out-Bound Pre-Processing tactic, *availability is compromised* when system nodes require continued or eventual connectivity.

The **Pre-Fetching** tactic can complement the In-Bound Pre-Processing tactic, but can also be used on its own to enable a mobile device to access data that is stored in the cloud via an intermediate surrogate, while providing elements to deal with intermittent connectivity between surrogates and the cloud and therefore *improving availability*. The surrogate, according to a defined pre-fetch algorithm, retrieves data from the cloud and stores it locally so that it is available to the mobile device when it needs it. This tactic *increases computing power and energy efficiency* as in the In-Bound Pre-Processing tactic. It also *improves response time* because it anticipates data needs in order to minimize communication to the cloud and reduce latency. However, it can *affect bandwidth efficiency negatively* between surrogates and the cloud if the pre-fetching algorithm retrieves more data than is necessary (also if it retrieves less data than necessary and therefore has to continuously retrieve additional data from the cloud). In addition, because the tactic requires continuous connectivity between mobile devices and the cloud, it also has a *negative effect on availability*.

C. Surrogate Provisioning

Figure 5 presents a decision model for selecting a surrogate provisioning tactic. In the **Pre-Provisioned Surrogate** tactic, offloaded computation and/or data processing operations are already installed on the surrogate at deployment time. It is therefore a good match for when there is a small, known set of computations or data processing operations that can be preloaded on the surrogate. It is also a good match for usage contexts in which multiple surrogates offer the same capabilities because it *simplifies the deployment process*. Pre-provisioned surrogates have the advantage of *shorter provisioning times* because the capabilities already reside on the surrogate. In addition, they provide *shorter response times* to requests from mobile devices, especially if capabilities are already started on the surrogate. However, pre-provisioned surrogates offer *very little flexibility* in terms of capabilities because they are limited by what is installed on them. *Maintainability is also reduced* because changes to capabilities have to be propagated to all surrogates.

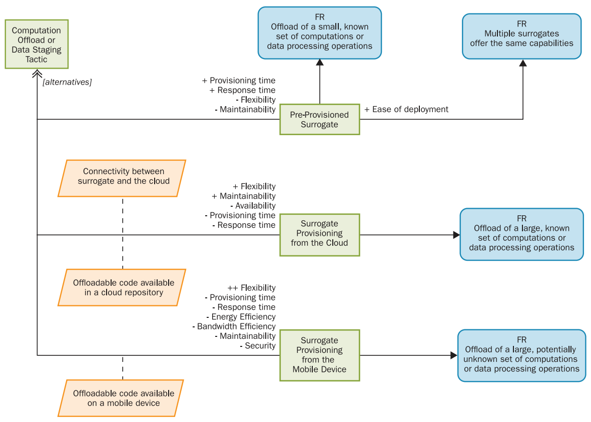
Computation Offload or Data Staging Tactic
[alternatives]
FR Offload of a small, known set of computations or data processing operations
FR Multiple surrogates offer the same capabilities
+ Provisioning time
+ Response time
- Flexibility
- Maintainability
Pre-Provisioned Surrogate
+ Ease of deployment
Connectivity between surrogate and the cloud
+ Flexibility
+ Maintainability
- Availability
- Provisioning time
- Response time
Surrogate Provisioning from the Cloud
FR Offload of a large, known set of computations or data processing operations
Offloadable code available in a cloud repository
++ Flexibility
- Provisioning time
- Response time
- Energy Efficiency
- Bandwidth Efficiency
- Maintainability
- Security
Surrogate Provisioning from the Mobile Device
FR Offload of a large, potentially unknown set of computations or data processing operations
Offloadable code available on a mobile device

Fig. 5. Decision Model for Surrogate Provisioning

In the **Surrogate Provisioning from the Cloud** tactic what is sent from the mobile device to the surrogate is the location of the offloaded computation or data processing operations in the form of a URL for the surrogate to download and install. It offers *greater flexibility* than the Pre-Provisioned Surrogate tactic because capabilities are not limited by what is already installed, making it a good match for when there is a large, known set of capabilities that can execute on a surrogate. However, these capabilities need to exist in a repository in the cloud, and connectivity between the surrogate and the repository is required for capabilities to be downloaded, therefore *affecting availability negatively*. It *improves maintainability* with respect to the Pre-Provisioned Surrogate tactic because changes to capabilities only need to be propagated to the cloud repository. However, *provisioning time is increased* with respect to the Pre-Provisioned Surrogate tactic because capabilities have to be downloaded from the repository and then installed and started. This also *increases response time*, at least for the first time the capability is executed.

In the **Surrogate Provisioning from the Mobile Device** tactic, the mobile device sends the offloaded computation or data processing operations to the surrogate at runtime. The surrogate installs and starts the capability on behalf of the mobile device. It offers the *greatest flexibility* because of the potential for executing any offloadable capability that resides on the mobile device, which makes it a good match for scenarios in which there is a large, potentially unknown set of capabilities that could execute on a surrogate. However, *provisioning time is increased* because the capability has to be transferred from the mobile device to the surrogate and then installed and started. *Energy efficiency* is decreased on the mobile device because of the battery power required on the mobile device to send the offloadable capability. This also *increases response time*, at least for the first time the offloaded capability is executed. In addition, depending on the size of the capability that is transferred, *bandwidth efficiency could be negatively affected*. *Maintainability is also reduced* because because changes to offloadable capabilities have to be propagated to all mobile devices. Finally, *security is negatively affected* because surrogates could be compromised by malicious code uploaded from mobile devices.

D. Surrogate Discovery

Figure 6 presents a decision model for selecting a surrogate discovery tactic. In the **Local Surrogate Directory** tactic the mobile device maintains a list of surrogates, with their network addresses or URLs in addition to any information that can help the mobile device to select the optimal surrogate in case more than one is available. This tactic has the *lowest complexity*. However, because the list is stored locally on the mobile device, if surrogate metadata changes or new surrogates are made available, a mobile device will not have an automated way of updating the surrogate directory, therefore having a *negative effect on maintainability*. It also *reduces flexibility* because the mobile device is limited to the surrogates on its list. The low complexity, along with the potential maintainability challenges, make this tactic a better fit for scenarios in which there is a relatively static, small number of surrogates. It *increases security* because a local list will likely include only surrogates that are trusted by the mobile device. If surrogates have information that can be used for identification, such as a QR code or a screen with configuration information, they could be added by the mobile device user to the list of surrogates. This option requires initial proximity between mobile devices and surrogates to scan or enter surrogate information, but would *improve maintainability and flexibility* because surrogates can be added or updated by the user. It can also *increase adaptability* to varying operational conditions if surrogate metadata is updated with offload execution data, such as response time and network conditions, and used by surrogate selection algorithms. It also has the potential to *improve response/execution time* if the surrogate selection algorithm uses the updated metadata. However, because the surrogate selection algorithm runs on the mobile device, it can *decrease energy efficiency* depending on the complexity of the algorithm and the number of monitored variables.

In the **Cloud Surrogate Directory** tactic the mobile device contacts a cloud server that maintains a list of potential surrogates. The cloud server selects the optimal surrogate from the directory, based on data such as mobile device characteristics, type of offload request, surrogate availability, or surrogate load, and sends its address back to the mobile device. Having the surrogate directory in the cloud has the advantage of a centralized location for surrogate registration, which *increases maintainability*. It also *increases flexibility* because all the

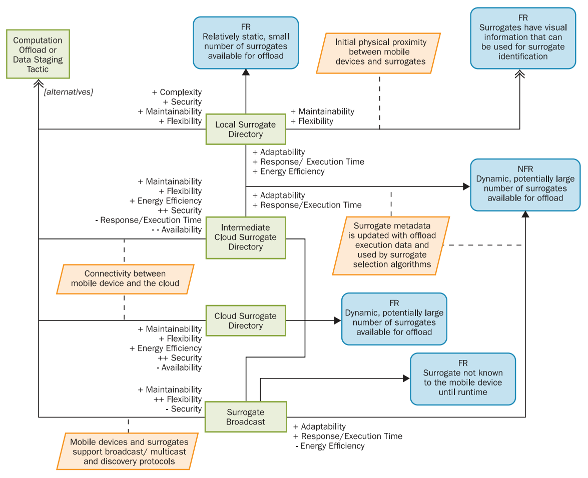
Computation Offload or Data Staging Tactic
[alternatives]
FR Relatively static, small number of surrogates available for offload
Initial physical proximity between mobile devices and surrogates
FR Surrogates have visual information that can be used for surrogate identification
+ Complexity
+ Security
+ Maintainability
+ Flexibility
Local Surrogate Directory
+ Maintainability
+ Flexibility
+ Adaptability
+ Response/ Execution Time
+ Energy Efficiency
NFR Dynamic, potentially large number of surrogates available for offload
+ Maintainability
+ Flexibility
+ Energy Efficiency
++ Security
- Response/Execution Time
- - Availability
+ Adaptability
+ Response/Execution Time
Intermediate Cloud Surrogate Directory
Surrogate metadata is updated with offload execution data and used by surrogate selection algorithms
Connectivity between mobile device and the cloud
Cloud Surrogate Directory
FR Dynamic, potentially large number of surrogates available for offload
+ Maintainability
+ Flexibility
+ Energy Efficiency
++ Security
- Availability
FR Surrogate not known to the mobile device until runtime
+ Maintainability
++ Flexibility
- Security
Surrogate Broadcast
+ Adaptability
+ Response/Execution Time
- Energy Efficiency
Mobile devices and surrogates support broadcast/ multicast and discovery protocols

Fig. 6. Decision Model for Surrogate Discovery

mobile device needs to know is the address of the cloud server, which maintains the list of all potential surrogates. Because the selection algorithm runs in the cloud it also *increases energy efficiency* on the mobile device. The centralization aspect enables this tactic to handle a dynamic, potentially large number of surrogates. *Security is highly increased* because the mobile device only needs to trust the cloud surrogate directory server and can pre-exchange credentials for authorization (Section IV-H). The surrogate directory server can also exchange credentials with its surrogates as part of the registration process, which means that the directory would only contains trusted surrogates. However, *response/execution time can be increased* because of the the additional directory query time. In addition, *availability is negatively affected* because the mobile device requires consistent connectivity to the cloud at least in the discovery phase, which means that the cloud server becomes a single-point-of-failure.

In the **Intermediary Cloud Surrogate Directory** tactic, a variation of the Cloud Surrogate Directory Tactic, the surrogate directory server does not return the selected surrogate address to the mobile device, but rather forwards the offload request to a selected surrogate, and then returns the results to the mobile device. Similar to the Cloud Surrogate Directory tactic, it *increases maintainability, flexibility, energy efficiency, and security*. An additional advantage is that because the directory server is involved in the communication with the surrogates, it can *increase adaptability* to varying operational conditions if surrogate metadata is updated with offload execution data. It also has the *potential to improve response/execution time* if the selection algorithm uses the updated metadata. However, *response/execution time can increase* because all mobile devices communicate with surrogates through the surrogate directory server and not directly, creating a potential bottleneck in the system. In addition, *availability is greatly decreased* because the mobile device requires consistent connectivity to the cloud in both the discovery and the offload phases, which means that the cloud server becomes a single-point-of-failure.

In the **Surrogate Broadcast** tactic, surrogates advertise their presence to mobile devices. This avoids having to keep surrogate directories up to date, which *improves maintainability*. It creates a much more dynamic environment in which mobile devices can discover nearby surrogates without needing to know their addresses in advance, or retrieving the addresses from a cloud server that could potentially not be available when needed, therefore providing a *high level of flexibility and availability*. The broadcast aspect enables this tactic to handle a dynamic, potentially large number of surrogates. *Providing security is challenging* because the surrogate may not be known to the mobile device until runtime and therefore no security credentials have been exchanged to generate trust between them (Section IV-H). Similar to the Local Surrogate Directory tactic, it *increases adaptability* to varying operational conditions and *improves response/execution time*, *potentially reducing energy efficiency* depending on the complexity of the surrogate selection process.

E. Resource Optimization

Figure 7 presents a decision model for resource optimization. These tactics are typically used to complement the Computation Offload tactic, but could complement Data Staging tactics if the surrogates provide computation-intensive data processing operations. The **Runtime Partitioning** tactic enables mobile systems to offload computation only if remote execution is better than local execution according to a defined optimization function. The complexity of this optimization function can range from a simple check to detect if a surrogate is available to a per-offload calculation based on code, device and network models. It *increases availability* because computation can execute locally if offload conditions are not optimal. It also *increases resource and energy efficiency* because offload decisions are made at runtime based on the operational environment. However, the execution of a very complex partitioning algorithm could *also lead to reduced energy efficiency*. In addition, the offloaded code has to exist on both the mobile device and the surrogate which can lead to *decreased maintainability* and also *reduces legacy leverage* because the code would have to be ported to also run on the mobile device. This tactic requires the development and profiling of the models and input data that are used in the optimization function, which can lead to *increased development time*. In addition, it is often difficult to create accurate models of device, network and code characteristics, which can therefore lead to *increased response time* and *reduced system performance* if the offload decision is not optimal [9].

The **Runtime Profiling** tactic enables mobile devices to gather data about current conditions to update the profiling data and models used in the calculation of the optimization function. It *increases energy efficiency* and *further increases resource efficiency* because current conditions are considered in the offload decision. Because the data used by the optimization function is updated either periodically or after every offload operation, any errors in the initial models and data are

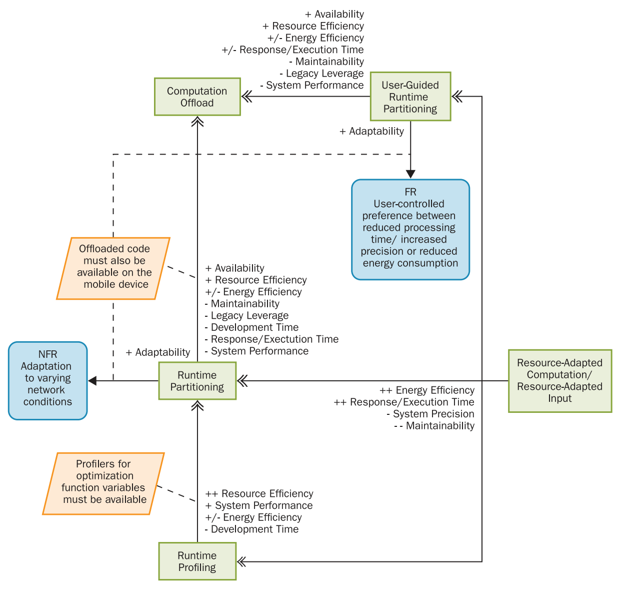

Fig. 7. Decision Model for Resource Optimization

adjusted over time, therefore *increasing system performance*. A constraint is that profilers have to be built to gather data necessary for the calculation of the optimization function, which can *increase development time*. However, the execution of the profilers could also lead to *reduced energy efficiency* if complexity and sampling frequency are high.

When computation offload systems are used for mission-critical or time-sensitive tasks, users may determine that, for example, reduced processing time or increased precision are preferred over reduced energy consumption. The **User-Guided Runtime Partitioning** tactic, a variation of the Runtime Partitioning tactic, enables users to select the goal of the optimization function therefore *increasing adaptability*. Similar to Runtime Partitioning, this tactic *increases availability, resource efficiency, and energy efficiency*, at the expense of *decreased maintainability, legacy leverage, system performance, and potentially energy efficiency and response/execution time*. However, this tactic can provide *better response and execution time* in mission-critical moments.

The **Resource-Adapted Computation** tactic enables systems to use different versions of offloadable code that match the resource characteristics of mobile devices and surrogates (i.e., computation that runs on the surrogate is more computation-intensive, and presumably more precise, than the equivalent computation that runs on the mobile device). The **Resource-Adapted Input** tactic enables systems to have identical versions of offloadable code but to operate on different input (e.g., lower or higher image resolution as input to an image processing algorithm may lead to different energy consumption). The difference with the previous tactics is that *both energy efficiency and response/execution are improved* because computation is matched to the node that is processing it. However, there is *decreased system precision* because the assumption is that the computation that runs on the mobile device is not as precise as what runs on the surrogate. In addition, *maintainability is decreased* because two versions of equivalent code have to be maintained.

F. Fault Tolerance

Figure 8 presents a decision model to select fault tolerance tactics to complement Computation Offload tactics. The **Local Fallback** tactic enables mobile devices to use a local copy of the offloadable computation in case the connectivity to the surrogate is lost, which *provides fault tolerance and increases availability*. It requires offloadable code to exist on the mobile device and the surrogate, and therefore *maintainability and legacy leverage are decreased* because there are multiple versions of the same code. Also, because execution restarts on the mobile device after disconnection is detected, *energy efficiency is decreased* because the computation executes locally. In addition, *response/execution time increases*, especially if disconnection is detected close to completion of execution on the surrogate.

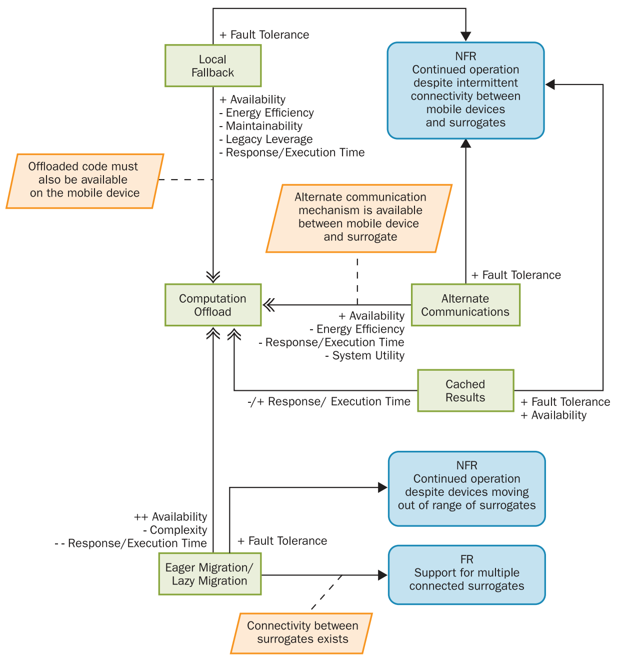
+ Fault Tolerance
Local Fallback
NFR Continued operation despite intermittent connectivity between mobile devices and surrogates
+ Availability
- Energy Efficiency
- Maintainability
- Legacy Leverage
- Response/Execution Time
Offloaded code must also be available on the mobile device
Alternate communication mechanism is available between mobile device and surrogate
+ Fault Tolerance
Computation Offload
Alternate Communications
+ Availability
- Energy Efficiency
- Response/Execution Time
- System Utility
Cached Results
-/+ Response/ Execution Time
+ Fault Tolerance
+ Availability
NFR Continued operation despite devices moving out of range of surrogates
++ Availability
- Complexity
- - Response/Execution Time
+ Fault Tolerance
Eager Migration/ Lazy Migration
FR Support for multiple connected surrogates
Connectivity between surrogates exists

Fig. 8. Decision Model for Fault Tolerance for Computation Offload

The **Alternate Communications** tactic enables a system to switch to an alternate, potentially less energy-efficient communications mechanism to continue serving the mobile user in spite of disconnection, to *provide fault tolerance, and increase availability*. While this tactic does not require offloadable code to be available on both the mobile device and the surrogate, it does require an alternate communications mechanism to exist between the mobile device and the surrogate (e.g., SMS). Because this alternate communication mechanism could be less optimal in terms of energy consumption, response time, and message size, therefore *energy efficiency, response/execution time, and system utility could be affected negatively*.

The **Eager Migration** tactic enables a surrogate to migrate offloaded computation to a connected surrogate when it detects that it might not be able to continue serving a mobile device. The **Lazy Migration** tactic, a variation of the Eager Migration tactic, does not migrate the computation, but rather continues execution of the offloaded computation on the same surrogate and routes the responses to the mobile device via a connected surrogate that is in its range. *Availability is greatly increased* because these tactics take a more proactive approach to detecting disconnection. However, *complexity increases* due to (1) support for multiple connected surrogates, (2) a mechanism to detect potential disconnection from a mobile device, and (3) a mechanism to determine the connected surrogate that will continue serving the mobile device. In Eager Migration, *response/execution time increases* based on the size of the computation that has to be migrated between surrogates, although this is a one-time cost. In Lazy Migration *response/execution time increases* due to the rerouting that takes place with every offload operation.

The **Cached Results** tactic enables surrogates to cache results of an offloaded operation if a mobile device becomes disconnected. The results are then delivered to, or retrieved by, the mobile device upon reconnection. *Fault tolerance and availability are increased* because even though the results are not immediately delivered, the system continues operating. For this same reason, *response time increases greatly* for the initial offload operation. However, once a mobile device is able to reconnect, because the offload operation has already been processed, the results are already available.

Figure 9 presents a decision model to select fault tolerance tactics to complement Data Staging tactics. The **Client-Side Data Caching** tactic, a variation of the Cached Results tactic, caches collected data on the mobile device if there is no connectivity to the surrogate, and eventually sends it to the surrogate when a connection is available. This tactic *increases fault tolerance and availability* due to continued operation despite loss of connectivity between the mobile device and the surrogate. However, because data is stored on the mobile device until a surrogate is available it can lead to *reduced storage efficiency*, even if data is deleted on the mobile device after it has been successfully uploaded to the surrogate.

The **Opportunistic Mobile-Surrogate Data Synchronization** tactic keeps data synchronized between mobile devices and surrogates during periods of connection, such that the system can continue operating in periods of disconnection. The use of this tactic *increases fault tolerance and availability*. Because data is stored on the mobile device, *response time improves* for data requests. However, limited storage and battery on mobile devices can lead to *storage inefficiency* if the size of the data set to synchronize is large, and to *reduced energy efficiency* if the complexity of the algorithms used to keep data synchronized is high. In addition, if data sets are synchronized that are never used, or synchronization policies

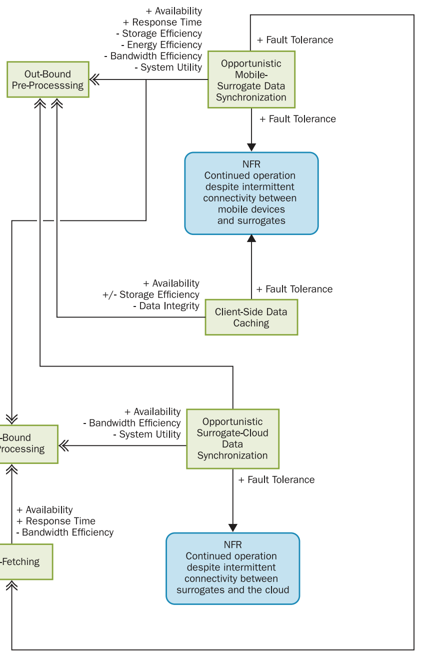

Fig. 9. Decision Model for Fault Tolerance for Data Staging

do not match the operational environment, it can lead to *bandwidth inefficiency*. Finally, *system utility may be reduced* if data on the mobile device becomes stale if not synchronized over time.

The **Opportunistic Surrogate-Cloud Data Synchronization** tactic, a variation of the Opportunistic Mobile-Surrogate Data Synchronization tactic, enables a system to continue operating in the event of disconnection between the surrogate and the cloud, and to synchronize data when reconnection occurs. The use of this tactic *increases fault tolerance and availability*. However, similar to the Opportunistic Mobile-Surrogate Data Synchronization tactic, there can be a *negative effect on system utility* if data becomes stale and also on *bandwidth inefficiency* if synchronized data is never used.

G. Scalability and Elasticity

Figure 10 presents a decision model for selecting scalability and elasticity tactics. These tactics are typically used to complement the Computation Offload tactic, but could also complement Data Staging tactics. The **Just-in-Time Containers** tactic creates a container and/or an instance of the offloaded code upon receipt of an offload request and then destroys the instance of the offloaded code when it completes, therefore *increasing scalability and elasticity*, which leads to *increased resource efficiency* on the surrogate. However, because the instance is created at runtime, there is a *response/execution time penalty* to create the instance before computation can execute.

The **Right-Sized Containers** tactic creates execution containers that are of the appropriate size for the offloaded com-

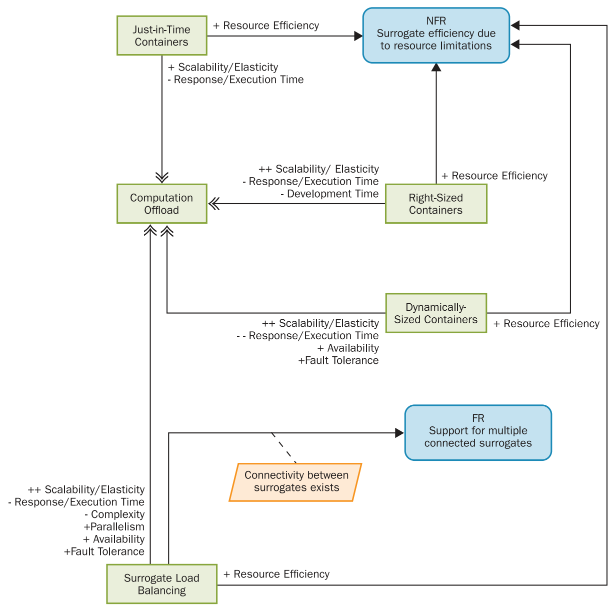
Just-in-Time Containers
+ Resource Efficiency
NFR Surrogate efficiency due to resource limitations
+ Scalability/Elasticity
- Response/Execution Time
++ Scalability/ Elasticity
- Response/Execution Time
- Development Time
+ Resource Efficiency
Computation Offload
Right-Sized Containers
Dynamically-Sized Containers
++ Scalability/Elasticity
- - Response/Execution Time
+ Availability
+Fault Tolerance
+ Resource Efficiency
FR Support for multiple connected surrogates
Connectivity between surrogates exists
++ Scalability/Elasticity
- Response/Execution Time
- Complexity
+Parallelism
+ Availability
+Fault Tolerance
Surrogate Load Balancing
+ Resource Efficiency

Fig. 10. Decision Model for Scalability and Elasticity

putation in order to optimize resource usage on the surrogate. Similar to the Just-in-Time Containers tactic, *scalability and elasticity, and resource efficiency are increased* because execution containers are created at runtime, but also contributes to *increased response/execution time*. There is *even greater scalability and elasticity* because there is a better match of code requirements to execution containers. However, there is potential for *increased development time* because offloadable code has to be profiled to determine the optimal size of its execution container.

The **Dynamically-Sized Containers** tactic, a variation of the Right-Sized Containers tactic, starts offloaded computation in a container of a predefined default size, but if an error occurs at runtime that indicates it does not have the necessary computing power, a larger container is created and the offload request is moved to the new container. As in the Right-Sized Containers tactic, *scalability, elasticity, and resource efficiency are increased, at the expense of increased response/execution time* because containers are created at runtime. However, this tactic creates the *potential for even greater response/execution time* due to the creation of the new container and migration of the computation to the new container when necessary. In exchange, the tactic *increases availability and fault tolerance* because of the continued operation of the offloaded task despite initially insufficient resources.

The **Surrogate Load Balancing** tactic enables surrogates to send offloaded computation to other less-loaded, connected surrogates in order to provide a better user experience to all mobile devices. *Scalability and elasticity are greatly enhanced, as well as resource efficiency* because offload requests are balanced across multiple connected surrogates. However, this tactic *increases response/execution time* for offload requests that are migrated during execution. It also *increases complexity* as it requires at least a load balancer and a system monitor to detect when thresholds have been reached and loads have to to migrated. In exchange, the tactic *increases fault tolerance and availability* because offload requests are migrated before the system is overloaded and stops responding.

H. Security

The decision model in Figure 11 shows that the **Trusted Surrogate** tactic addresses two non-functional requirements related to security: (1) mobile devices should only send requests to trusted surrogates, and (2) surrogates should only accept requests from trusted mobile devices. Because there are very few cyber-foraging systems that address security [3], the decision model informally presents different options for implementing the tactic instead of complementary tactics.

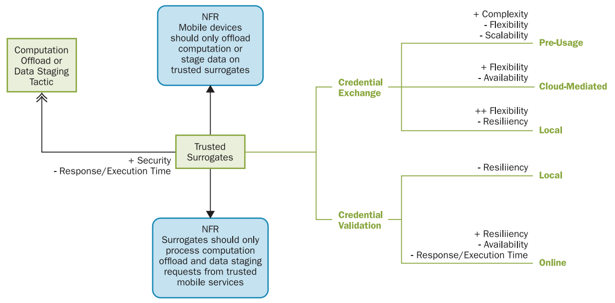
Computation Offload or Data Staging Tactic
NFR Mobile devices should only offload computation or stage data on trusted surrogates
+ Security
- Response/Execution Time
Trusted Surrogates
NFR Surrogates should only process computation offload and data staging requests from trusted mobile services
Credential Exchange
Credential Validation
+ Complexity
- Flexibility
- Scalability
Pre-Usage
+ Flexibility
- Availability
Cloud-Mediated
++ Flexibility
- Resiliiency
Local
- Resiliiency
Local
+ Resiliiency
- Availability
- Response/Execution Time
Online

Fig. 11. Decision Model for Security

1) Credential Exchange: Security credentials (e.g., username/password, keys, certificates, biometrics) have to be exchanged to create a trusted relationship between mobile devices and surrogates.

1) Pre-Usage Credential Exchange: Credentials are exchanged prior to usage such that there is a pre-existing security relationship between a mobile device and a surrogate. The advantage is *reduced complexity*. However, there is a *negative effect on flexibility* because mobile devices can only interact with surrogates with pre-existing security relationships. It also *may not scale* for systems where there are many-to-many relationships between mobile devices and surrogates.
2) Cloud-Mediated Credential Exchange: Credential exchange takes place as part of the offload process the first time that a mobile device uses a surrogate. The mobile device contacts a cloud-based system that is trusted by the surrogate to register its credentials. The cloud-based system validates the mobile device credentials and, if valid, sends the mobile device the surrogate credentials, and also sends the surrogate the mobile device credentials. The advantage is that it provides *greater flexibility* because it is not limited to surrogates with a pre-existing security relationship. However, *availability*

is decreased if the mobile device and the surrogate are not connected to the intermediary cloud-based system to exchange credentials.

3) Local Credential Exchange: Credential exchange happens directly between a mobile device and a surrogate as part of the offload process. This method offers the *greatest flexibility* because it enables a mobile device to use any surrogate. However, it has a *negative effect on resiliency* because of the security risks of not having a third party to validate credentials. It would have to rely on out-of-band channels for securely pairing mobile devices and surrogates such as physical proximity, context, sensors, visual channels and physical interactions [11].

2) Credential Validation: Once credentials have been exchanged, either in advance or during the offload process, they have to be validated at runtime.

1) Local Validation: Credentials are validated on each node according to defined security policies. The disadvantage is the *negative effect on resiliency* because it does not protect against revoked credentials.
2) Online Validation: Credentials are sent to an online trusted authority for validation. The advantage is that protects against revoked credentials because these are centrally validated. However, there is a *negative effect on availability* because the offload cannot happen unless there is connectivity to the trusted third party. In addition, *response/execution time increases* because of the additional time needed for online validation.

V. Validation

The architectural tactics for cyber-foraging presented in [3] were validated via three case studies of research protoype systems:

1) Tactical Cloudlets: Computation offload system for use in tactical environments [12]. Architecture reconstruction of the system was performed to identify implemented architectural tactics. Developers verified the identified tactics and answered questions regarding their validity and utility.
2) GigaSight: Data staging system for scalable crowd-sourcing of video from mobile devices [13]. Case study and developer involvement was the same as in the Tactical Cloudlets system.
3) AgroTempus: System targeted at agricultural knowledge exchange in resource-challenged regions [14]. This was a new development. The developer was observed and interviewed throughout the process to understand how the architectural tactics were used and how they influenced the development process.

After the execution of the case studies, the same developers were presented with the decision model and asked to answer the following questions to obtain their expert opinion on the correctness and usefulness of the tactics. The developers of the Tactical Cloudlets and GigaSight systems are experienced (>5 years of architecture and development experience) while the developer of the AgroTempus system is junior (<1 year).

1. Are the decision models correct? The developers of the three systems agreed that the decision models were correct, clear, coherent, and covered the main questions that would arise when doing a tradeoff analysis. The only comment was on the effect on response/execution time of the Cached Results tactics. After some discussion we separated the effect as negative for the initial offload operation and positive when the mobile device reconnects, as is now shown in Figure 8.

2. Would the decision models have been useful to support decision making in the development of your cyber-foraging system? Why? The developers of the three systems found the decision models useful. They found it very simple to use the "+ and -" notation to navigate through the models and visually understand the effect of using the different tactics. One of the experienced developers stated that it is valuable to have such an explicit guide when thinking about the architecture of a system. Even though experienced architects and developers already think in terms of design decisions and system qualities, the model provides a tool to reflect and reason, and helps them to be more thorough in their design considerations. The junior developer said that it would have greatly helped to have the decision model when he started the architecture of the system because he would have made better informed decisions, especially because of the explicit indication of benefits and tradeoffs. For inexperienced architects the model is even more helpful because it not only gives them a tool to think (like experienced architects) but it also codifies past reusable knowledge that they lack.

3. Would the decision models help with architecture evaluation, and particularly identification of tradeoffs and risks? Why? The developers agreed that the decision models would help with architecture evaluation because of the explicit qualification of benefits and tradeoffs. However, they all agreed that it would be necessary to take a closer look at scenarios in which one tactic has a positive effect and another has a negative effect, as described in Section III. The decision models help to identify the tradeoffs, but not to quantify the concrete effect because it varies depending on requirements and operational conditions. A future version of the decision model and tactics should include tools and methods to profile the parameters that influence the positive or negative effect on a system quality when using a particular tactic.

4. What else could the decision models be useful for? Ideas from the developers included:

- Template for creating similar models for other types of systems
- Instrument for communicating the different options and complexities of developing cyber-foraging systems
- Input for building developer tools that automatically generate components based on the tactics
- Aid for understanding system qualities that were likely influential when reverse engineering a system

VI. Related Work

Decision models have been used extensively in Software Engineering to model the problem and solution space, and eventually map the former to the latter to guide decision making. A well-known approach for creating decision models is Questions-Options-Criteria (QOC) [15], where questions represent problems, options map to candidate solutions, and criteria are used to determine how well the options fare with respect to the questions at hand. Another popular approach from the field of Software Measurement is Goal Questions Metric (GQM) [16]. The difference with QOC is that GQM does not look for candidate solutions; it only states the problem (in terms of the goal), then asks a number of questions to refine the goal and finally measures the object of study (e.g. product or process) according to the metrics. In essence, GQM allows to model the problem according to the goals and questions, but it also provides the metrics to be used for assessing an object and subsequently make decisions to improve it.

This paper proposes a decision model for the domain of cyber-foraging systems that links tactics to functional and non-functional requirements. Similarly, Gross and Yu [17] propose an approach to support designers in selecting design patterns based on their impact on non-functional requirements (represented as goals and related in graphs). Also, Zdun [18] proposes decision models made of software patterns, by formalizing pattern relationships and further annotating them with effects on quality goals. This allows architects or designers to parse through the design space by navigating from pattern to pattern until a combination of selected patterns optimally meets the quality goals. Finally Harrison and Avgeriou [19] propose an approach to model and annotate how tactics implementing specific quality attributes can fit within architecture patterns. They focus on helping architects choose between tactics depending on their compatibility with the overall architecture. Our focus is also tactic selection but targeted at understanding their effect on system qualities.

VII. Conclusions

The paper presented a decision model for cyber-foraging systems that maps functional and non-functional requirements to architectural tactics for cyber-foraging. Each mapping is qualified with the benefits and tradeoffs of using each tactic to help architects of cyber-foraging systems to understand the effect of their decisions. The decision model was validated by the developers of three cyber-foraging systems who agreed on the correctness and usefulness of the model for architecture and design of these types of systems. We believe this is a valuable instrument for moving cyber-foraging systems out of the labs and into real operational settings.

Acknowledgments

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. [Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution (DM-0003200). Special thanks to Reuel Brion, Sebastián Echeverría, and Pieter Simoens for their invaluable participation in the validation of the decision model.

References

[1] M. Satyanarayanan, "Pervasive computing: vision and challenges," *Personal Communications, IEEE*, vol. 8, no. 4, pp. 10–17, Aug 2001.

[2] G. Lewis, P. Lago, and G. Procaccianti, "Architecture strategies for cyber-foraging: Preliminary results from a systematic literature review," in *Proceedings of the 8th European Conference on Software Architecture (ECSA 2014)*, ser. Lecture Notes in Computer Science, vol. 8627, 2014, pp. 154–169.

[3] G. Lewis and P. Lago, "Architectural tactics for cyber-foraging: Results of a systematic literature review," *Journal of Systems and Software*, vol. 107, pp. 158–186, 2015.

[4] M. Satyanarayanan, R. Schuster, M. Ebling, G. Fettweis, H. Flinck, K. Joshi, and K. Sabnani, "An open ecosystem for mobile-cloud convergence," *Communications Magazine, IEEE*, vol. 53, no. 3, pp. 63–70, 2015.

[5] G. A. Lewis and P. Lago, "A catalog of architectural tactics for cyber-foraging," in *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA'15)*. ACM, 2015, pp. 53–62.

[6] ——, "Characterization of cyber-foraging usage contexts," in *Proceedings of the 9th European Conference on Software Architecture (ECSA 2015)*, ser. Lecture Notes in Computer Science, vol. 9278, 2015, pp. 195–211.

[7] K. Berg, J. Bishop, and D. Muthig, "Tracing software product line variability: From problem to solution space," in *Proceedings of the 2005 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*, ser. SAICSIT '05, 2005, pp. 182–191.

[8] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*, 3rd ed. Addison-Wesley, 2012.

[9] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 335–348.

[10] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 280–293.

[11] M. Farb, Y.-H. Lin, T. H.-J. Kim, J. M. McCune, and A. Perrig, "Safeslinger: Easy-to-use and secure public-key exchange," in *Proceedings of ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2013.

[12] S. Echeverria, G. A. Lewis, J. Root, and B. Bradshaw, "Cyber-foraging for improving survivability of mobile systems," in *Military Communications Conference, MILCOM 2015*, Oct 2015, pp. 1421–1426.

[13] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, and M. Satyanarayanan, "Scalable crowd-sourcing of video from mobile devices," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '13. New York, NY, USA: ACM, 2013, pp. 139–152.

[14] R. Brion, "AgroTempus," 2015. [Online]. Available: http://reuelbrion.github.io/AgroTempus/

[15] A. MacLean, R. M. Young, V. M. Bellotti, and T. P. Moran, "Questions, options, and criteria: Elements of design space analysis," *Human–computer interaction*, vol. 6, no. 3-4, pp. 201–250, 1991.

[16] V. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," *Encyclopedia of software engineering*, pp. 528–532, 1994.

[17] D. Gross and E. Yu, "From non-functional requirements to design through patterns," *Requirements Engineering*, vol. 6, no. 1, pp. 18–36, 2001.

[18] U. Zdun, "Systematic pattern selection using pattern language grammars and design space analysis," *Software: Practice and Experience*, vol. 37, no. 9, pp. 983–1016, 2007.

[19] N. B. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? a model and annotation," *Journal of Systems and Software*, vol. 83, no. 10, pp. 1735–1758, 2010.