Software Architecture Strategies for Cyber-Foraging Systems

Grace A. Lewis

2016

SIKS Dissertation Series No. 2016-22
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Software Engineering Institute
Carnegie Mellon University

This research was made possible, in part, by the kind cooperation of the Software Engineering Institute, a federally funded research and development center sponsored by the Department of Defense and operated by Carnegie Mellon University.

ISBN 978-94-6295-483-0

Cover design by Klaus Bellon
Published by Uitgeverij BOXpress || proefschriftmaken.nl
Typeset in LaTeX by the author

VRIJE UNIVERSITEIT

Software Architecture Strategies for Cyber-Foraging Systems

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. V. Subramaniam,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Exacte Wetenschappen
op dinsdag 7 juni 2016 om 13.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Grace Alexandra Lewis

geboren te Elizabeth, New Jersey, Verenigde Staten

promotor: prof.dr. P. Lago

Contents

Acknowledgements

There are so many people that I would like to thank for accompanying me on this journey.

I have to start by thanking my loving family. Without their continued support and understanding this journey would have been impossible. To my husband Mike for being mom and dad when I couldn't be there. I love you now and always. To Alex and Andrea for being the light of my life and the engine that keeps me going. I hope that one day you see this journey as an example that it is never too late to follow your dreams. I love you so much!

To Patricia Lago, my promotor and my guide. I could not have asked for a better advisor. I learned so much from you, not only in terms of research, but as someone who wants to make the world a better place. I am proud to call you my friend. Thank you!

To my Mom for always telling me that I could do anything that I wanted. To my sister Ingrid for being the best sister and friend that anyone could ask for. Gracias mis gordas. Las adoro!

To my bosses Ed and Mark for their invaluable support in making things easy so I could go on this journey. Thank you.

To the AMS team and friends for putting up with me these past three years: Aubrey, Ben, Bill, Dan, Gene, James, Jeff, Joe, Keegan, Kevin, Linda, Luis, Marc, Sebastián, and Soumya. Thanks. You are awesome!

To my friends Tina and Ipek. Having such sweet and caring friends make life a lot easier. Now we can relax and celebrate.

To my Amsterdam family for providing a home away from home: Damian, Eltjo, Ermeson, Fahimeh, Giuseppe, Han, Maryam, Mojca, Nelly, and Reuel. Thanks guys!

To Satya and the Elijah group at Carnegie Mellon University for inspiring my thesis topic and providing feedback and ideas.

To my thesis committee for their encouragement, insightful comments, and feedback.

I am sure there are many more people that I need to thank because journeys never happen alone. All I have left to say is: Thanks for joining me on my journey!

1 Introduction

1.1 Motivation

Smartphones, tablets, and now phablets, have become for many the preferred way of interacting with the Internet, social media and the enterprise:

- Mobile devices are increasingly becoming the first go-to device for communications and content consumption [24][25].
- The number of mobile devices will surpass desktops for the first time in 2015 [11].
- The time people spend using their smartphone is now exceeding the time spent looking at TV screens [45][87].
- It is not uncommon for there to be multiple mobile devices per user and household [46].

In addition, mobile traffic will keep increasing due to several factors:

- Wearable technology: Wearable technology is showing a consistent increase in popularity [85]. According to Cisco Systems, the wearables market will grow five-fold in the next five years from 109 million devices in 2014 to 578 million devices by 2019 [23]. It is expected that this growth will result in an 18-fold increase in mobile data traffic.
- Demanding content types: According to mobiForge, by the end of 2015, 4G LTE data use will rise by 59% and mobile video will account for 60% of data traffic [86].

- Internet of Things (IoT): According to Gartner, 4.9 billion connected things will be in use in 2015, up 30 percent from 2014, and will reach 25 billion by 2020 [44]. Cisco Systems forecasts that 99 percent of physical objects will eventually become part of a network [23].

- Smartphone penetration: Even though worldwide the number of feature phones is still greater than the number of smartphones, the rate of smartphone subscription penetration has been growing steadily over the past five years [66]. This rate is predicted to increase very quickly given that it is expected that by 2020, 75% of smartphone buyers will pay less than $100US dollars for a device [45].

Because of these mobile device trends, organizations are pushing out more and more content and functionality to mobile users. Therefore, it is not unreasonable for users to expect the performance and capabilities of mobile devices to be equal to laptops and desktops. However:

- mobile devices will always lag behind their PC counterparts due to size and battery limitations;

- limited battery life remains a problem especially for computation- and communication-intensive applications;

- large and variable end-to-end latency between mobile device and cloud, and the possibility of disruptions, have a negative effect on user experience; and

- it will only get worse with the amount of network traffic generated by IoT and growing market share of low-cost smartphones.

Cyber-foraging is a promising technology for providing increased computing power and network efficiency to mobile devices, while conserving battery life.

1.2 Mobile Cloud Computing and Cyber-Foraging

Mobile Cloud Computing (MCC) refers to the combination of mobile devices and cloud computing in which cloud resources perform computation-intensive tasks and store massive amounts of data [122]. Cyber-foraging is an area of work within MCC that leverages external resources (i.e., cloud servers or local servers called surrogates) to augment the computation and storage capabilities of resource-limited mobile devices while extending their battery life [107].

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 capability. The second is data staging to improve data transfers between mobile devices and the cloud by temporarily staging data in transit on surrogates.

1.3 Software Architecture and Cyber-Foraging

The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both [13]. Software architectures are created because a system's qualities, expressed as functional and non-functional requirements, can be analyzed and predicted by studying its architecture.

One of the main challenges of building cyber-foraging systems is the dynamic nature of the environments that they operate in. For example, the connection to an external resource may not be available when needed or may become unavailable during a computation offload or data staging operation. As another example, multiple external resources may be available for a cyber-foraging system but not all have the required capabilities. Adding capabilities to deal with the dynamicity of the environment has to be balanced against resource consumption on the mobile device so as to not defeat the benefits of cyber-foraging. Being able to reason about the behavior of a cyber-foraging system in light of this uncertainty is key to meeting all its desired qualities, which is why software architectures are especially important for cyber-foraging systems.

Given the potential complexity of cyber-foraging systems, it would be of great value for software architects to have a set of reusable software architectures and design decisions that can guide the development of these types of systems, the rationale behind these decisions, and the external context/environment in which they were made; this is called *architectural knowledge* [68][73]. One way to capture architectural knowledge is in the form of *software architecture strategies*.

We define a *software architecture strategy* as the set of architectural design decisions that are made in a particular external context/environment to achieve particular system qualities. Software architecture strategies are codified as architectural tactics that can be reused in the development of software systems. We define *architectural tactics* as design decisions that influence the achievement of a system quality (i.e., quality attribute) [13].

Software architecture strategies for cyber-foraging systems are therefore

the set of architectural design decisions, codified as reusable tactics, that can be used in the development of cyber-foraging systems to achieve particular system qualities such as resource optimization, fault tolerance, scalability and security, while conserving resources on the mobile device.

1.4 Research Questions

In an effort to define software architecture strategies for cyber-foraging systems, we formulate our main research question (RQ) as follows:

RQ: What software architecture strategies can be used to build cyber-foraging systems?

To define software architecture strategies for building cyber-foraging systems, we first need to understand the architecture and design decisions that have been made by others in the development of industrial and academic cyber-foraging systems. Our first research sub-question is thus:

RQ1: What software architecture design decisions for cyber-foraging systems can be identified in the literature?

Once architectural design decisions are identified, we need to (1) select those design decisions that are common across multiple cyber-foraging systems to achieve functional and non-functional requirements, and (2) codify them as reusable architectural tactics for cyber-foraging. This leads us to our second research sub-question:

RQ2: What architectural tactics can be derived from the identified architectural design decisions?

Because architectural design decisions are always made in a particular external context or environment, we need to understand the functional and non-functional requirements that drive the development of cyber-foraging systems in each of these usage contexts. Therefore, our third research sub-question is:

RQ3: What are the usage domains and contexts (defined in terms of functional and non-functional requirements) that benefit from cyber-foraging?

Finally, it is important for a software architect to know (1) what tactics can be used to satisfy different functional and non-functional requirements, and (2) the effect that combinations of tactics have on functional and non-functional requirements. Our fourth and final sub-question is thus:

RQ4: How to support architectural design decision making in cyber-foraging systems?

1.5 Thesis at a Glance

Our research context is presented in Figure 1.1, showing how the RQs are related to the goals of this thesis. This research started by conducting a systematic literature review (SLR) of architectural design decisions in cyber-foraging systems proposed in the literature, expressed as decisions related to where to offload, when to offload, and what to offload (RQ1). The results of the SLR showed common architectural design decisions that led to the identification of functional and non-functional architectural tactics designed to satisfy particular functional and non-functional requirements. We developed case studies for three different cyber-foraging systems to validate the application of the tactics to promote particular functional and non-functional requirements (RQ2). We then did a literature study to characterize the usage domains and contexts for the cyber-foraging systems in the primary studies identified in the SLR, in order to understand the functional and non-functional requirements that are relevant to these contexts (RQ3). Finally, we created a mapping between the problem space (functional and non-functional requirements) and the solution space (functional and non-functional architectural tactics), and identified dependencies between elements of the problem and solution space, as well as dependencies between tactics. The result of this mapping was a decision model to help software architects in the development of cyber-foraging systems that meet their intended functional and non-functional requirements while understanding the effects of their decisions (RQ4).

1.6 Research Methods

In this thesis we used a number of qualitative research methods that are common in software engineering research.

- Systematic literature review: This research method is an evidence-based approach to thoroughly search studies relevant to a set of pre-defined research questions and critically select, assess, and synthesize findings to answer the research questions [31][63]. We used this method to identify architectural design decisions in cyber-foraging systems present in the literature (Chapter 2).

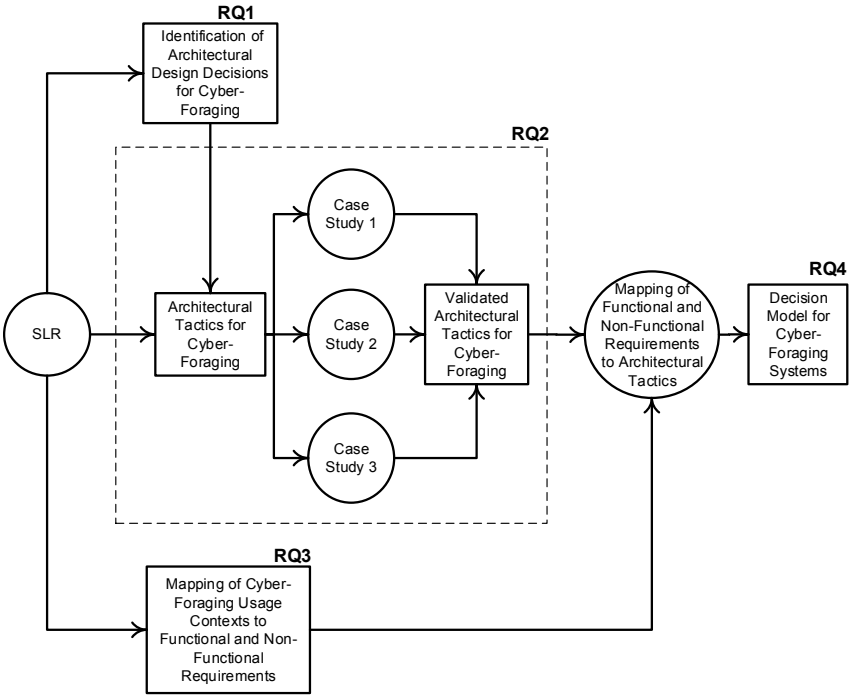

Figure 1.1: Research Context

- Literature study: Also called literature review, a literature study is different from a systematic literature review in that it is less formal and structured, but provides more freedom in selecting relevant studies and analyzing their content. Although the results might not be as complete and valid as those of a systematic literature review, thanks to its effectiveness and efficiency, this research method is often used to gain specific knowledge or understand a topic. We used this method to identify usage contexts and domains for cyber-foraging systems and map them to relevant functional and non-functional requirements in each context (Chapter 7).

- Case study: A case study is an empirical method aimed at investigating contemporary phenomena in their context [106]. They are descriptive and detailed, with a narrow focus, combining subjective and objective

data. We conducted three case studies to investigate the use of architectural tactics for cyber-foraging in real systems. In Chapter 4 we used a case study to investigate an existing cyber-foraging system for computation offload, and in Chapter 5 to investigate an existing cyber-foraging system for data staging. In addition, in Chapter 6 we used a case study to investigate the use of architectural tactics for the development of a new cyber-foraging system for both computation offload and data staging.

1.7 Outline of Thesis and Publications

The research presented in this thesis has either been published previously or is currently under submission. The following chapters are based on the following publications.

- Chapter 2: This chapter addresses research question RQ1 and presents the results of an SLR to investigate architectural design decisions in cyber-foraging systems. Parts of this chapter were previously published as:

  Grace A. Lewis, Patricia Lago, and Giuseppe Procaccianti. Architecture Strategies for Cyber-Foraging: Preliminary Results from a Systematic Literature Review. In proceedings of the 8th European Conference on Software Architecture, volume 8627 of Lecture Notes in Computer Science, pages 154-169. Springer International Publishing, 2014.

- Chapter 3: This chapter addresses research question RQ2 and presents the set of architectural tactics that were derived from the SLR. Parts of this chapter were previously published as:

  Grace A. Lewis and Patricia Lago. Architectural Tactics for Cyber-Foraging: Results of a Systematic Literature Review. Journal of Systems and Software, 107:158-186, 2015.

  Grace A. Lewis and Patricia Lago. A Catalog of Architectural Tactics for Cyber-Foraging. In Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures, pages 53-62. ACM, 2015.

- Chapter 4: This chapter addresses research question RQ2 and is the first of three case studies to validate the architectural tactics presented

in Chapter 3. Parts of this chapter were previously published as:

Grace A. Lewis, Sebastián Echeverría, Soumya Simanta, Ben Bradshaw, and James Root. Cloudlet-Based Cyber-Foraging for Mobile Systems in Resource-Constrained Edge Environments. In Companion Proceedings of the 36th International Conference on Software Engineering, pages 412-415. ACM, 2014.

Grace A. Lewis, Sebastián Echeverría, Soumya Simanta, Ben Bradshaw, and James Root. Tactical Cloudlets: Moving Cloud Computing to the Edge. In Military Communications Conference (MILCOM), 2014 IEEE, pages 1440-1446, Oct 2014.

Grace A. Lewis, Sebastián Echeverría, Soumya Simanta, James Root, and Ben Bradshaw. Cloudlet-Based Cyber-Foraging in Resource-Limited Environments. Emerging Research in Cloud Distributed Computing Systems, pages 92-121, 2015.

Parts of this chapter were submitted as:

Grace A. Lewis, Patricia Lago, Reuel Brion, Sebastián Echeverría, and Pieter Simoens. A Tale of Three Systems: Case Studies on Application of Architectural Tactics for Cyber-Foraging Systems. Journal of Software and Systems.

- Chapter 5: This chapter addresses research question RQ2 and is the second of three case studies to validate the architectural tactics presented in Chapter 3. Parts of this chapter were submitted as:

  Grace A. Lewis, Patricia Lago, Reuel Brion, Sebastián Echeverría, and Pieter Simoens. A Tale of Three Systems: Case Studies on Application of Architectural Tactics for Cyber-Foraging Systems. Journal of Software and Systems.

- Chapter 6: This chapter addresses research question RQ2 and is the third of three case studies to validate the architectural tactics presented in Chapter 3. Parts of this chapter were submitted as:

Grace A. Lewis, Patricia Lago, Reuel Brion, Sebastián Echeverría, and Pieter Simoens. A Tale of Three Systems: Case Studies on Application of Architectural Tactics for Cyber-Foraging Systems. Journal of Software and Systems.

- Chapter 7: This chapter addresses research question RQ3 and presents a mapping of usage contexts and domains for cyber-foraging systems to functional and non-functional requirements. Parts of this chapter have been published as:

  Grace A. Lewis and Patricia Lago. Characterization of Cyber-Foraging Usage Contexts. In proceedings of the 9th European Conference on Software Architecture, volume 9278 of Lecture Notes in Computer Science, pages 195-211. Springer International Publishing, 2015.

- Chapter 8: This chapter addressses research question RQ4 and presents a mapping of functional and non-functional requirements for cyber-foraging systems to the architectural tactics presented in Chapter 3. The result is a decision model for the development of cyber-foraging systems. Parts of this chapter have been published as:

  Grace A. Lewis, Patricia Lago and Paris Avgeriou. A Decision Model for Cyber-Foraging Systems. In proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture (WICSA 2016), IEEE, April 2016.

2

A Systematic Literature Review of Architectural Design Decisions for Cyber-Foraging Systems

This chapter presents the protocol and results of a Systematic Literature Review (SLR) to discover architectural design decisions in cyber-foraging systems. It includes an analysis of the identified primary studies using a categorization of architecture decisions related to what, when and where to offload computation and data from mobile devices. The results show that this is an area with many opportunities for research that will enable cyber-foraging systems to become widely adopted as a way to support the mobile applications of the present and the future.

2.1 Research Protocol

To identify work related to architectures for cyber-foraging a SLR was conducted following the guidelines proposed in [31] and [63]. The research question, search strategy, inclusion and exclusion criteria, and validation method are presented in the following subsections.

2.1.1 Research Question

The goal of the SLR is to identify work in cyber-foraging with a software architecture perspective. To achieve this goal, the following research question is defined:

What software architecture and design decisions for cyber-foraging from mobile devices can be identified in the literature?

2.1.2 Search Strategy

Three main keywords can be built from the research question: cyber-foraging, mobile devices, and software architecture. Each of these keywords has a set of related synonyms and alternative spellings. Based on these keywords and their related terms the following basic search string was defined:

(cyber foraging OR cyber-foraging OR code offload OR code offloading OR computation offload OR computation offloading OR data offload OR data staging) AND (mobile OR handheld OR smartphone) AND (software architecture OR software design OR system architecture)

The main data source was Google Scholar.[1] Snowballing was used to complement the set of primary studies. The advantage of using Google Scholar was that it included studies that are outside of software engineering, such as computer engineering and computer science, which is where many of the studies on cyber-foraging originate. The downside is that it returns many results which are irrelevant because it performs a full-text search and because there is no control process that ensures that all results are valid (i.e., are academic or industrial publications). To make sure that all relevant studies were identified, the dates were left open even though the term cyber-foraging was coined in 2001.

Details of each study were recorded using JabRef.[2] Separate JabRef databases were created for each round of the primary study identification process.

2.1.3 Inclusion and Exclusion Criteria

The inclusion and exclusion criteria shown in Table 2.1 were defined and applied to the search results.

2.1.4 Validation

The protocol was validated by executing it on Google Scholar without snowballing. The goal was to determine if it was rigorous enough and to improve it where necessary. The results of multiple iterations of the search string were

[1] `http://scholar.google.com/`
[2] `http://jabref.sourceforge.net/`

checked against a set of 17 known relevant studies in the field of cyber-foraging. This set was validated by an expert in the field. The search string was adjusted accordingly until it returned all 17 relevant studies either directly or as one of the references (first-level snowballing). The inclusion and exclusion criteria were reviewed during the process to ensure that the results were representative of software architecture and design of cyber-foraging systems.

Table 2.1: Inclusion and Exclusion Criteria

Inclusion Criteria	**Exclusion Criteria**
A study that proposes software architectures for cyber-foraging. *Rationale:* We are interested in studies that present software architecture and design of cyber-foraging systems in which mobile components, surrogate/server components, and offload elements are clearly defined. *Example:* A study that presents the software architecture and design of a cyber-foraging system for both the mobile device as well as the surrogate/server and clearly defines what computation or data is being offloaded.	A study that proposes a cyber-foraging system that does not present software architecture and design details. *Rationale:* If the study does not present architecture and design details, it does not contain information that can be abstracted into generic architectural tactics. *Examples:* A study that presents a cyber-foraging solution that discusses only the benefits of the solution but does not contain software architecture details will not be included. A study that surveys cyber-foraging solutions that have already been presented in other studies and does not propose a new cyber-foraging solution will not be included. A study that only discusses an offload algorithm and not a complete solution for cyber-foraging will not be included.
A study that proposes a cyber-foraging system for computation offload or data staging in which the mobile device is augmenting its computing power by using surrogates such as cloud resources. *Rationale:* A cyber-foraging system leverages surrogates to perform computation that would make sense to execute locally but if executed on the mobile device would drain resources or not provide adequate performance, or to stage data in transit to and from cloud resources and mobile devices. *Example:* A study that presents a cyber-foraging solution that uses surrogates to offload expensive computation or to store data temporarily until centralized resources become available.	A study that proposes a system in which mobile devices simply access cloud services or in which computation is partitioned across similar nodes. *Rationale:* A system that simply uses cloud services as parts of its functionality or that distributes computation among other mobile devices is not a case of cyber-foraging because it is not leveraging a more powerful surrogate to extend its computing power. *Example:* A study that presents a mobile cloud solution in which cloud services are accessed from mobile devices simply to fulfill part of its functionality or a study that represents distribution of computation across a mobile ad hoc network (MANET).

Continued on next page

Table 2.1 – *Continued from previous page*

Inclusion Criteria	Exclusion Criteria
A study that proposes solutions based on open technologies that contain enough detail to abstract the main software architecture components. *Rationale:* Studies that rely on open technologies are more likely to present solution details. *Example:* A study that presents software architecture views for a cyber-foraging solution that relies on open or readily-available technologies will be included.	A study proposed by a commercial vendor or that relies on proprietary hardware or network protocols. *Rationale:* Studies produced by vendors are unlikely to contain architecture information because it is part of their intellectual property. In addition, characteristics of solutions that rely on specific hardware or protocols will not be able to be abstracted into general architecture patterns and strategies. *Example:* A study that presents a cyber-foraging solution that only works if connected to a vendor's network or that requires special hardware, networking devices or protocols for communication will not be included.
A study that is in the form of a published scientific paper or industrial publication. *Rationale:* A scientific paper focuses on scientific content and follows a process to guarantee a good level of quality. Also, as solutions may have been devised by industrial organizations, broader industrial publications describing such solutions should be included. *Examples:* A study in a refereed journal that is part of a conference or a technical report that follows a standard publication template (i.e., abstract, introduction, description of the problem, proposed solutions, related work and references), a PhD or Masters thesis, or a study in an industrial publication that presents details of a cyber-foraging system or architecture will be included.	A study that is not in the form of a published scientific paper or that is in an industrial publication but only focuses on the commercial benefits of the solution. *Rationale:* Lack of scientific content and rigorous methods can lead to a low-quality outcome. In addition, studies in industrial publications targeted at increasing sales and that only highlight benefits of the solution do not add scientific value to the outcome of the review. *Examples:* Papers that have not been published, scientific papers that do not follow a standard publication template, keynote summaries, tables of contents, collections of abstracts, workshop summaries, project proposals, slide sets, and commercial product brochures will not be included.

2.2 Identification of Primary Studies

2.2.1 Round 1

The search string was last entered in Google Scholar on September 17, 2013 and returned 430 results. The complete list of results is available as online material at `http://www.andrew.cmu.edu/user/gritter/InitialStudies-CyberForaging.html`. The studies were evaluated against the inclusion and exclusion criteria *based on the title, abstract, keywords and an initial scan of*

the study. The results are shown in Table 2.2.

Table 2.2: Round 1 Results

Result	Studies	Description
Yes	91	Studies that met the inclusion and exclusion criteria based on the title, abstract, keywords and an initial scan of the study
No	297	Studies that did not meet the inclusion and exclusion criteria based on the title, abstract, keywords and an initial scan of the study
Maybe	23	Studies that did not fully meet the inclusion criteria based on the title, abstract, keywords and an initial scan of the study, but that warranted a full read due to the coverage of software architecture
Duplicate	18	Studies that were identical to other studies or were a subset of a larger study by the same author(s) (e.g., a paper that was cross-listed or a paper that is explicitly a chapter of a PhD or Masters thesis, in which case we included the thesis because it is the superset)
Plagiarism	1	Study that was copied from a conference paper that we co-authored in 2013.
TOTAL	**430**	
TOTAL FOR ROUND 2	**114**	Studies with Result = Yes and Result = Maybe

2.2.2 Round 2

The studies with Result = Yes and Result = Maybe from Round 1 were *fully read* and evaluated against the inclusion and exclusion criteria. The list of studies evaluated in Round 2 is available as online material at `http://www.andrew.cmu.edu/user/gritter/Round2Studies-CyberForaging.html`. The results of the evaluation are shown in Table 2.3.

2.2.3 Final Round

The references in each study with Result = Yes from Round 2 were evaluated against the inclusion criteria based on title, abstract and keywords as an initial round of snowballing. Those that passed based on this initial scan were fully read and included if they fully met the inclusion criteria. The results are shown in Table 2.4.

Table 2.3: Round 2 Results

Result	Studies	Description
Yes	50	Studies that met the inclusion and exclusion criteria based on fully reading the study
No	62	Studies that did not meet the inclusion and exclusion criteria after reading the study in full
Duplicate	12	Studies that were a subset of a larger study by the same author(s) (e.g., a paper that after a full read was determined to be part of a PhD or Masters thesis or a shorter version of a study that reports the same results from a software architecture perspective)
TOTAL	**114**	

Table 2.4: Final Round Results

Result	Studies	Description
Direct	50	Studies with Result = Yes from Round 2
Snowballing	8	Studies that correspond to references in the *Direct* results that met the inclusion and exclusion criteria based on fully reading the study
TOTAL	**58**	

The list of 58 primary studies is presented in Table 2.5. The *Primary Study* column contains the reference for the study. The *Type* column is the type of study which can be BC (Book Chapter), CP (Conference Paper), DD (Doctoral Dissertation), JA (Journal Article), MT (Masters Thesis), or TR (Technical Report). *System Name* refers to the name of the cyber-foraging system that is described in the study. The *Form* is the form of cyber-foraging which can be CO (Computation Offload) or (DS = Data Staging). The *Domain or Use Case* refers to the targeted domain or use case for the system. Finally, the *Source* column is the source of the study which is either GS (Google Scholar) or S (Snowballing).

Table 2.5: Primary Studies

Primary Study	Type	System Name	Form	Domain or Use Case	Source
Ahnn2013 [2]	JA	mHealthMon	CO	Healthcare	GS
Angin2013 [5]	JA	Mobile Agents	CO	Java applications	GS

Continued on next page

Table 2.5 – *Continued from previous page*

Primary Study	Type	System Name	Form	Domain or Use Case	Source
Armstrong2006 [6]	CP	Edge Proxy	DS	Web page updates	GS
Aucinas2012 [7]	CP	Clone-to-Clone (C2C)	CO	Intelligent transport systems, Mobile multi-player online games	GS
Bahrami2006 [8]	CP	Mobile Information Access Architecture for Occasionally-Connected Computing	DS	Occasionally-connected operations	GS
Balan2007 [9]	CP	Chroma	CO	Mobile interactive resource-intensive applications	S
Chang2011 [16]	JA	Collaborative Applications	CO	Speech recognition	GS
Chen2004 [18]	JA	Computation and Compilation Offload	CO	Image and video processing	GS
Cheng2013 [19]	TR	Cloud Media Services	CO	Hybrid Broadcast Broadband TV (HBB-TV)	GS
Chu2004 [20]	JA	Roam	CO	Seamless applications	GS
Chun2009 [22]	CP	CloneCloud	CO	Mobile applications in general	GS

Continued on next page

Table 2.5 – *Continued from previous page*

Primary Study	Type	System Name	Form	Domain or Use Case	Source
Cuervo2012 [26]	DD	MAUI (Mobile Assistance Using Infrastructure)	CO	Operations that consume and produce small amounts of information compared to their computational requirements	GS
		Kahawai	CO	Graphics applications that require high-end GPU rendering	
Duga2011 [30]	MT	HPC-as-a-Service	CO	High-Performance Computing (HPC)	GS
Endt2011 [33]	BC	OpenCL-Enabled Kernels	CO	Automotive	GS
Esteves2011 [35]	CP	Real Options Analysis	CO	Mobile applications in general	GS
Fjellheim2005 [39]	JA	3DMA	CO	Context-aware applications	GS
Flinn2002 [41]	CP	Spectra	CO	Mobile interactive resource-intensive applications	S

Continued on next page

Table 2.5 – *Continued from previous page*

Primary Study	Type	System Name	Form	Domain or Use Case	Source
Flinn2003 [42]	CP	Trusted and Unmanaged Data Staging Surrogates	DS	Distributed filesystems	GS
Giurgiu2009 [47]	CP	AlfredO	CO	Typical three-tiered applications implemented as OSGi[3] bundles for each tier	S
Goyal2011 [48]	DD	Collective Surrogates	CO	Mobile applications in general	GS
Guan2008 [51]	DD	Grid-enhanced mobile devices	CO	Ambient intelligence	GS
Ha2011 [52]	TR	Cloudlets	CO	Computation-intensive applications in hostile environments	GS
Hung2011 [55]	JA	Virtual Phone	CO	Mobile applications in general	GS
Imai2012 [56]	MT	Single-Server Offloading	CO	Moderately-slow, single-purpose, computation-intensive applications	GS

Continued on next page

[3]The Open Service Gateway Initiative, or OSGi, is a specification and Java framework for developing and dynamically deploying modular software programs and libraries (`http://www.osgi.org`).

Table 2.5 – *Continued from previous page*

Primary Study	Type	System Name	Form	Domain or Use Case	Source
		Cloud Operating System to Support Multi-Server Offloading	CO	Very computation-intensive mobile applications	
Iyer2012 [57]	CP	Android Extensions	CO, DS	Mobile applications that handle complex computations or large amounts of data	GS
Jarabek2012 [59]	CP	ThinAV	CO	Anti-malware scanning	GS
Kemp2012 [62]	CP	Cuckoo	CO	Mobile applications in general	GS
Kosta2012 [64]	CP	ThinkAir	CO	Mobile applications in general	GS
Kovachev2012 [65]	JA	MACS (Mobile Augmentation Cloud Services)	CO	Mobile applications in general	GS
Kristensen2010 [67]	DD	Scavenger	CO	Image manipulation, continuous speech recognition, augmented reality	GS
Kundu2007 [71]	JA	Telemedik	DS	Healthcare	GS

Continued on next page

Table 2.5 – *Continued from previous page*

Primary Study	Type	System Name	Form	Domain or Use Case	Source
Kwon2013 [72]	CP	AMCO (Adaptive, Multitarget Cloud Offloading)	CO	Java applications	GS
Lee2012 [74]	BC	MCo	CO	Java applications	GS
Matthews2011 [82]	CP	PowerSense	CO	Telemedicine (Image Processing for Dengue Detection)	GS
Messer2002 [83]	CP	AIDE	CO	Java applications	GS
Messinger2013 [84]	TR	Application Virtualization on Cloudlets	CO	Mobile applications in general	GS
Mohapatra2003 [88]	TR	PARM	CO	Mobile applications in general	GS
Ok2007 [92]	CP	Resource Furnishing System	CO	Computation-intensive mobile applications	GS
OSullivan2013 [93]	CP	Cloud Personal Assistant (CPA)	CO	Cloud Services	GS
Park2012 [96]	CP	SOME (Selective Offloading for a Mobile computing Environment)	CO	HTML5 web applications	S
Phokas2013 [98]	CP	Feel The World (FTW)	DS	Participatory sensing applications	GS
Pu2013 [100]	CP	SmartVirtCloud (SmartVC)	CO	Mobile applications in general	GS

Continued on next page

Table 2.5 – *Continued from previous page*

Primary Study	Type	System Name	Form	Domain or Use Case	Source
Ra2011 [101]	CP	Odessa	CO	Mobile interactive perception applications	S
Rachuri2012 [102]	DD	Smartphone-based social sensing	CO	Social sensing applications	GS
Rahimi2012 [103]	CP	MAPCloud	CO	Rich mobile applications	GS
Satyanarayanan2009 [108]	JA	VM-Based Cloudlets	CO	Computation-intensive mobile applications	S
Shi2013 [111]	TR	IC-Cloud	CO	Mobile applications in general	GS
Silva2008 [112]	CP	SPADE	CO	Mobile applications that perform lengthy tasks	GS
Su2005 [114]	CP	Slingshot	CO	Computation-intensive mobile applications	S
Verbelen2012 [117]	JA	AIOLOS	CO	Complex multimedia applications	S
Xiao2013 [119]	CP	Large-Scale Mobile Crowdsensing	DS	Crowdsensing applications	GS
Yang2008 [121]	JA	Offloading Toolkit and Service	CO	Java applications	GS

Continued on next page

Table 2.5 – *Continued from previous page*

Primary Study	Type	System Name	Form	Domain or Use Case	Source
Yang2012 [120]	TR	Sonora	DS	Continuous data streams	GS
Yang2013 [122]	JA	Mobile Data Stream Application Framework	CO	Data stream applications	GS
Zhang2009 [128]	CP	Heterogeneous Auto-Offloading Framework for Mobile Web Browsers	CO	Web pages with multimedia content	GS
Zhang2011 [127]	JA	Weblets	CO	Web applications	GS
Zhang2012 [129]	JA	DPartne	CO	Java applications	GS
Zhang2012a [126]	CP	Elastic HTML5	CO	Web applications	GS

2.3 Categorization of Primary Studies

2.3.1 Studies Per Type

As shown in Figure 2.1, most of the primary studies are papers published in conference proceedings (28) followed by journal articles (15). Even though the scope of the search included industry reports, of the six studies identified as Technical Reports, only one comes from industry. The others are from universities (4) and an FP7 project (1). In addition, there were two book chapters, two Masters Theses and five Doctoral Dissertations. This distribution shows that even though the topic is of potential interest to industry, most of the published work in this area comes from academia.

2.3.2 Studies Per Year

As shown in Figure 2.2, the number of primary studies per year has grown since the first study dated 2002. This shows that it is indeed a topic of interest, especially in the last three years.

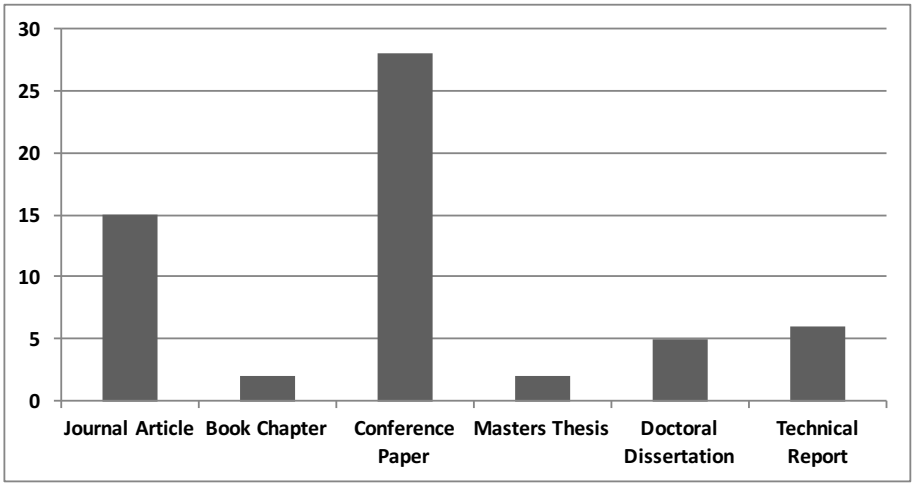

Figure 2.1: Number of Primary Studies Per Type

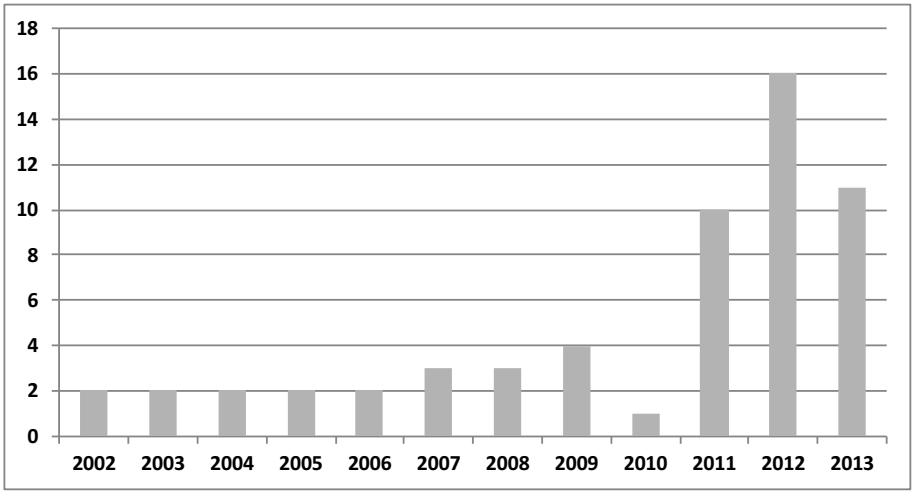

Figure 2.2: Number of Primary Studies Per Year

2.4 Threats to Validity

Google Scholar was the single data source for the primary studies, and was complemented by snowballing. The search string was adjusted until it returned the set of studies that was identified by an expert in the field as the set of seminal studies (either direct results or in the references). However, the problem is that if a study is not listed in Google Scholar it will not be returned in the results. For example, Google Scholar returned [22], which is one of the seminal studies, but did not return a later study on the same system [21].

In addition, the term software architecture (which is part of the search string) was not widely used until the mid 2000s, which is reflected in the years of the studies and Figure 2.2. This was mitigated by snowballing.

2.5 Analysis of Primary Studies

2.5.1 Categorization of Architecture Decisions

Defining an architecture for a system that uses cyber-foraging to enhance the computing power of mobile devices requires making decisions on where, when, and what to offload, from the perspective of the mobile device. In particular, the specific questions we identified are:

- Where to offload? Is computation and/or data offloaded to proximate (single-hop) resources or remote (multi-hop) resources?
- When to offload? With optimization in mind, when does it make sense to offload?
- What to offload? What is the granularity of the computation or data that is offloaded? What is the size and type of payload to use the computation or to offload data?

The answers to the three cyber-foraging questions (where, when and what) from the 58 primary studies were clustered based on similarity. If an answer did not belong to an existing category, a new category was created and the answers to the previously analyzed studies were revisited to see if they needed to be re-categorized. Figure 2.3 shows the resulting categorization for computation offload and Figure 2.4 shows the resulting categorization for data staging. The difference between the two categorizations is in the third question related to what to offload.

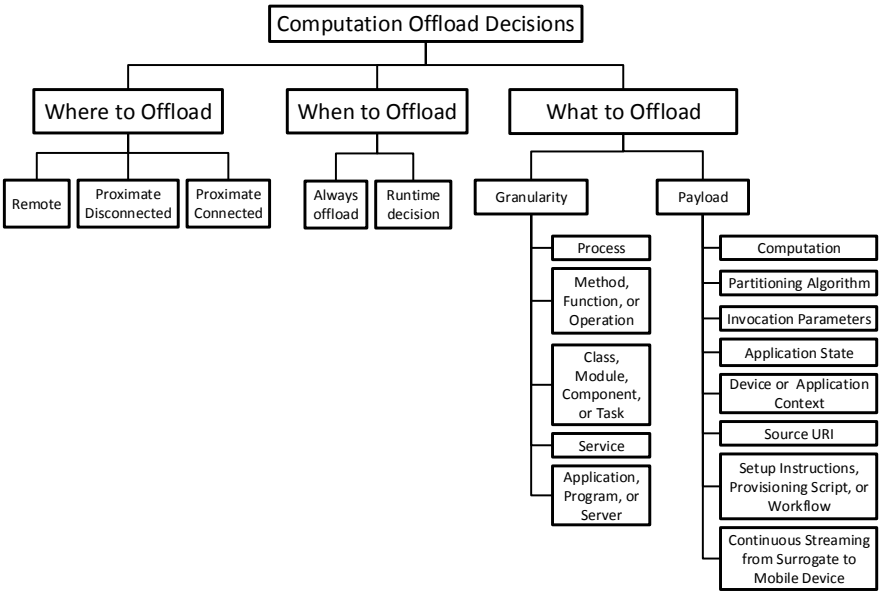

Figure 2.3: Categorization of Architecture Decisions for Computation Offload

2.5.1.1 Where to Offload

In cyber-foraging systems, computation or data is offloaded to resources with greater computing power. These resources are located in either single-hop or multi-hop proximity of the mobile devices that use them. For both computation offload and data staging, the answers to this question were grouped as follows. The answers map to the leaves under *Where to Offload* on the left side of Figures 2.3 and 2.4.

- Remote: Computation or data is offloaded to a remote resource such as an enterprise cloud or data center, as shown in Part (a) of Figure 2.5. In this case, it is a synchronous operation that requires multiple network hops between the mobile device and the enterprise cloud. Even though the bandwidth between the mobile device and the first hop (i.e., the first mile) and the last hop and the cloud resource (i.e., the last mile) is potentially high bandwidth, the bandwidth between the rest of the hops is likely low.

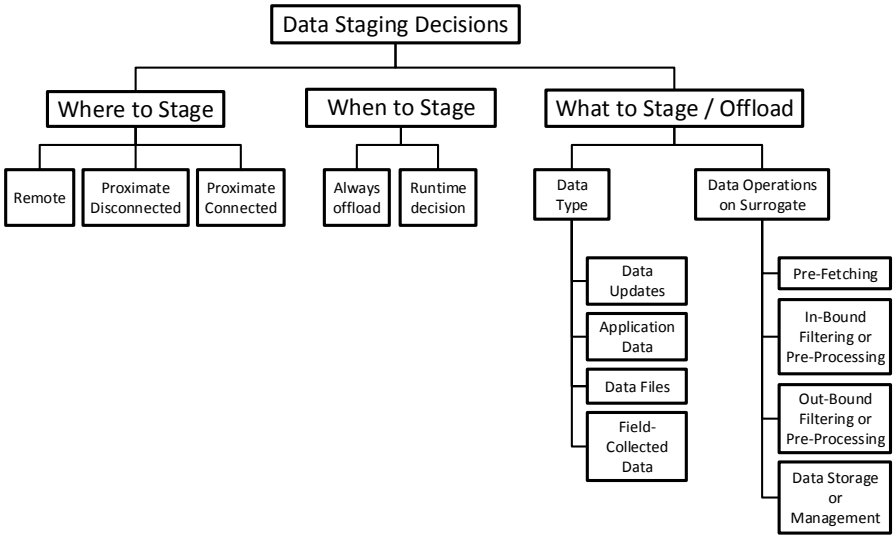

Figure 2.4: Categorization of Architecture Decisions for Data Staging

- Proximate Disconnected: Computation or data is offloaded to a surrogate located in single-hop proximity of the mobile device, and the surrogate can operate without being connected to the cloud resource, as shown in Part (b) of Figure 2.5. In this case, the offload operation is synchronous to a surrogate that is in single-hop proximity over a likely high bandwidth connection. The communication between the surrogate and the cloud resource is asynchronous and multi-hop over a likely lower bandwidth connection.

- Proximate Connected: As in the previous case, computation or data is offloaded to a surrogate located in single-hop proximity of the mobile device. However, as shown in Part (c) of Figure 2.5, the surrogate needs to be connected at runtime to the cloud resource. In this case, the offload operation is synchronous to a surrogate that is in single-hop proximity over a likely high bandwidth connection. The communication between the surrogate and the cloud resource is synchronous and multi-hop over a likely lower bandwidth connection but communication between the two is not necessary for every offload operation (i.e., necessary only for initial provisioning, data synchronization, or to fetch missing data).

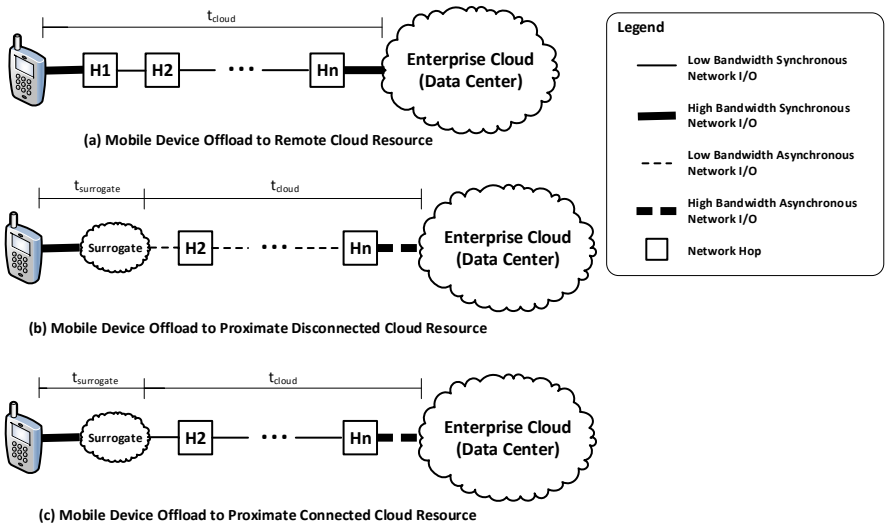

Figure 2.5: Mobile Device Offload to Proximate and Remote Cloud Resources

The reason why this is an important architectural decision is because if we assume that t = *roundtrip communication time* and $t_{surrogate}$ is less than t_{cloud}, as defined in Figure 2.5, nearby surrogates are a better option from an energy consumption and latency perspective [10].

2.5.1.2 When to Offload

In general, offloading is beneficial when large amounts of computation are needed with relatively small amounts of communication [70]. For both computation offload and data staging, the answers to this question were grouped as follows. The answers map to the leaves under *When to Offload* in the middle of Figures 2.3 and 2.4.

- Always Offload: Computation or data is always offloaded because either the computation is not available locally or the assumption is that offloading is more efficient from an energy consumption or latency perspective.
- Runtime Decision: Computation or data is only offloaded if remote execution is better than local execution according to a defined utility function based on a combination of parameters such as energy, latency and network bandwidth.

The reason why this is an important architectural decision is because a runtime offload decision implies that the code can execute both locally and remotely and that data is gathered either statically and/or dynamically to calculate the optimization function, whereas an always offload decision does not require a runtime calculation but does assume that the offload target is always available.

2.5.1.3 What to Offload

What to offload involves two architecture decisions, but these are different for computation offload and data staging. For computation offload, one decision has to do with the granularity of the computation that is offloaded to the surrogate or cloud resource, and another has to do with the payload that is sent from the client to the surrogate or cloud resource in order to execute the offloaded computation. Although these seem like low-level decisions, they have architectural implications because they determine the components that are needed on the client and the surrogate. For example, processes and methods create very tight coupling between the client and the surrogate in order to synchronize state, whereas applications, programs and scripts have looser coupling because all they require is an appropriate container or runtime environment. In addition, a technique based on methods, threads and module/components will probably require code to be annotated or re-written while services, applications, programs and scripts will likely require no changes to the software, especially if it follows a thin client/thick server paradigm. The answers to this question were grouped as follows. The answers map to the leaves under *What to Offload* on the right side of Figure 2.3.

- Granularity (ordered from smallest to largest)
  - Process: A process (or thread) that is executing in the context of the mobile device is offloaded for execution in an equivalent context on the surrogate or cloud resource.
  - Method, Function or Operation: A method, function or operation that is part of a larger programming construct (e.g., class, module, program) is offloaded for execution on the surrogate or cloud resource.
  - Class, Module, Component or Task: A class, module, component or task that is composed of methods, functions, sub-modules, subcomponents, or sub-tasks is offloaded for execution on the surrogate or cloud resource.

  - Service: A service that exposes a standardized interface is offloaded for execution on the surrogate or cloud resource.
  - Application, Program or Server: A complete application, program, or server portion of a client/server system is offloaded for execution on the surrogate or cloud resource.

- Payload
  - Computation: The actual computation is sent from the mobile device to the surrogate or cloud resource so that it can be remotely executed.
  - Partitioning Algorithm: The execution of a partitioning algorithm that is part of the *When to Offload* decision is offloaded to the surrogate or cloud resource to determine what portions of the computation should be executed on the mobile device and which should be executed on the surrogate or cloud resource.
  - Invocation Parameters: The computation already exists on the surrogate or cloud resource and therefore what is sent from the mobile device are the parameters use when invoking the computation.
  - Application State: The state of the application is sent to the computation that exists on the surrogate or cloud resource so that the remote computation can be executed with the same state as that of the application running on the mobile device.
  - Device Context: Parameters that describe the context of the device or the application are sent to the surrogate to provide additional information for the execution of the computation.
  - Source Location: What is sent from the mobile device to the surrogate is the location (e.g., URI) from which the computation can be retrieved so that it can be installed on the surrogate or simply executed.
  - Setup Instructions: Instructions on how to set up and/or assemble the computation are sent from the mobile device to the surrogate or cloud resource.
  - Continuous Data from Surrogate to Mobile Device: No payload is sent from the mobile device to the surrogate; the surrogate continuously executes the computation and sends the results to the mobile device.

For data staging, one architectural decision has to do with the type of data that is being staged and the other has to do with the operations that are offloaded to the surrogate or cloud resource to be performed on the data. As with computation offload, the answer to this question has architectural implications because it requires different components on both sides depending on how data is stored and forwarded. For data staging the answers to this question were grouped as follows. The answers map to the leaves under *What to Offload* on the right side of Figure 2.4.

- Data Type
  - Data Updates: Data is staged on a surrogate and updates are sent to the mobile device as defined by update policies established between the surrogate and the mobile device.
  - Application Data: Data used by an application on a mobile device is retrieved from a cloud resource and staged on a surrogate.
  - Data Files: Files needed by a mobile user are retrieved from a cloud resource and staged on a surrogate.
  - Field-Collected Data: Data collected in the field, such as sensor data, is sent from the mobile device to a surrogate as defined by data forwarding policies established between the surrogate and the mobile device.
- Data Operations On Surrogate
  - Pre-Fetching: Computation executes on the surrogate that attempts to determine the data that is likely to be used by the mobile devices that it is serving, and then retrieves that data from cloud resources and stores it in the surrogate so that it is available for use by the mobile devices when they need it.
  - In-Bound Filtering or Pre-Processing: Computation executes on the surrogate to process data that is retrieved or pushed from cloud resources so that the mobile devices that it serves receive data that is ready to be consumed, or filtered such that the mobile devices only receive the data that they need.
  - Out-Bound Filtering or Pre-Processing: Computation executes on the surrogate to process data that is received from mobile devices such that the data that is sent on to the cloud resource is processed and ready for consumption by the cloud resource (e.g., cleaned, filtered or merged data)

– Data Storage or Management: The surrogate serves as storage for data in transit between mobile devices and cloud resources. Operations to process that data (i.e., CRUD operations[4]) are also available on the surrogate.

The reason that this is an important architectural decision is because what to offload determines client and surrogate components.

2.5.2 Analysis Results

A mapping between the primary studies and the architectural decisions presented in Figures 2.3 and 2.4 is shown in Tables 2.6 and 2.7. Table 2.6 shows the mapping for the computation offload studies and Table 2.7 shows the mapping for the data staging studies. A study may appear in both tables if it presents architectures for both computation offload and data staging systems. If a study presents more than one architecture, a system identifier is included in parentheses after the primary study reference for each. Cuervo2012 [26] and Imai2012 [56] have two entries in Table 2.6 because they present two different systems for computation offload. Iyer2012 [57] has one entry in Table 2.6 and one in Table 2.7 because it presents systems for both computation offload and data staging. Therefore, the total of primary studies is 58 and the total of systems analyzed is 53 for computation offload and 8 for data staging for a total of 61 systems.

Table 2.6: Computation Offload Systems in Primary Studies

System	Where			When		What													
						Granularity					Payload								
	Prox. Disconnected	Prox. Connected	Remote	Runtime Decision	Always Offload	Process	Function	Component	Service	Application	Computation	Partitioning Algo.	Parameters	Application State	Device Context	Source Location	Setup Instructions	Continuous Data
mHealthMon [2]			X	X					X				X					
Mobile Agents [5]			X	X				X			X		X					
Clone-to-Clone (C2C) [7]	X			X			X						X					

Continued on next page

[4]CRUD is an acronym for Create Read Update Delete and refers to the main functions of data management applications.

Table 2.6 – *Continued from previous page*

System	Where			When		What												
						Granularity					Payload							
	Prox. Disconnected	Prox. Connected	Remote	Runtime Decision	Always Offload	Process	Function	Component	Service	Application	Computation	Partitioning Algo.	Parameters	Application State	Device Context	Source Location	Setup Instructions	Continuous Data
Chroma [9]	X			X			X						X					
Collaborative Applications [16]	X			X			X						X	X				
Computation and Compilation Offload [18]	X			X			X						X					
Cloud Media Services [19]			X		X			X							X			
Roam [20]		X	X	X				X					X	X		X		
CloneCloud [22]	X				X	X								X				
MAUI [26]	X		X	X			X						X					
Kahawai [26]	X				X					X								X
HPC-as-a-Service [30]	X		X		X				X				X					
OpenCL-Enabled Kernels [33]	X		X	X				X			X		X					
Real Options Analysis [35]	X		X	X			X						X					
3DMA [39]			X	X				X					X					
Spectra [41]	X			X			X						X					
AlfredO [47]			X	X				X				X	X					
Collective Surrogates [48]		X	X		X					X			X				X	
Grid-Enhanced Mobile Devices [51]		X	X		X			X					X					
Cloudlets [52]	X				X					X	X		X					
Virtual Phone [55]			X		X			X						X				
Single-Server Offloading [56]	X			X			X						X					
Cloud Operating System [56]		X			X			X			X		X					
Android Extensions [57]			X		X			X					X					
ThinAV [59]		X	X		X				X				X					
Cuckoo [62]	X		X	X				X			X		X					
ThinkAir [64]			X	X			X				X		X					
MACS [65]	X		X	X				X			X		X					
Scavenger [67]	X			X				X			X		X					
AMCO [72]	X		X	X				X					X	X				
MCo [74]			X		X			X			X		X					

Continued on next page

Table 2.6 – *Continued from previous page*

System	Where			When		What												
						Granularity					Payload							
	Prox. Disconnected	Prox. Connected	Remote	Runtime Decision	Always Offload	Process	Function	Component	Service	Application	Computation	Partitioning Algo.	Parameters	Application State	Device Context	Source Location	Setup Instructions	Continuous Data
PowerSense [82]	X			X					X				X					
AIDE [83]	X		X	X				X					X					
Application Virtualization [84]	X				X					X	X		X					
PARM [88]	X			X				X					X					
Resource Furnishing System [92]		X	X		X					X								X
Cloud Personal Assistant [93]		X	X		X				X				X					
SOME [96]			X		X		X						X					
SmartVirtCloud [100]		X		X			X				X		X					
Odessa [101]	X		X	X				X					X					
Smartphone-Based Social Sensing [102]	X		X	X				X					X					
MAPCloud [103]		X	X	X					X				X				X	
VM-Based Cloudlets [108]	X				X					X	X		X					
IC-Cloud [111]	X		X	X			X						X					
SPADE [112]			X		X			X					X					
Slingshot [114]		X			X					X			X					
AIOLOS [117]	X		X	X				X			X		X					
Offloading Toolkit and Service [121]	X		X	X				X			X		X					
Mobile Data Stream Application Framework [122]			X	X				X				X	X					
Heterogeneous Auto-Offloading Framework [128]			X	X				X					X					
Weblets [127]			X	X				X					X					
DPartner [129]	X		X	X				X					X					
Elastic HTML5 [126]	X		X	X				X					X			X		

2.5.2.1 Where to Offload

Figure 2.6 shows the distribution of systems in the studies for the architectural decision related to where to offload or stage data. Some of the systems have

Table 2.7: Data Staging Systems in Primary Studies

System	Where			When		What							
						Data Type				Data Operations			
	Prox. Disconnected	Prox. Connected	Remote	Runtime Decision	Always Offload	Data Updates	Application Data	Data Files	Field-Collected Data	Pre-Fetching	In-Bound Processing	Out-Bound Processing	Storage
Edge Proxy [6]		X			X	X					X		
Mobile Information Access Architecture for Occasionally-Connected Computing [8]	X				X		X			X			
Trusted and Unmanaged Data Staging Surrogates [42]	X				X			X		X			
Android Extensions [57]			X		X		X						X
Telemedik [71]	X		X		X		X				X		
Feel the World [98]	X	X	X		X				X				X
Large-Scale Mobile Crowdsensing [119]		X			X				X			X	
Sonora [120]	X	X	X		X				X			X	

more than one answer to the question, as indicated by multiple entries under *Where to Offload* in Table 2.6 and *Where to Stage* in Table 2.7. As an example, Edge Proxy [6] in Table 2.7 can offload to only proximate connected resources but Telemedik [71] in Table 2.7 can offload to proximate disconnected or remote resources. A category was created for each combination of answers:

- Proximate Disconnected (only)
- Proximate Connected (only)
- Remote (only)
- Remote or Proximate Disconnected
- Remote or Proximate Connected
- All: Proximate Disconnected, Proximate Connected, and Remote

Tied with the next category, most of the systems in the studies (16/61 or 26%) offload to only proximate disconnected resources, which is expected because of the advantages of lower latency and battery consumption that come

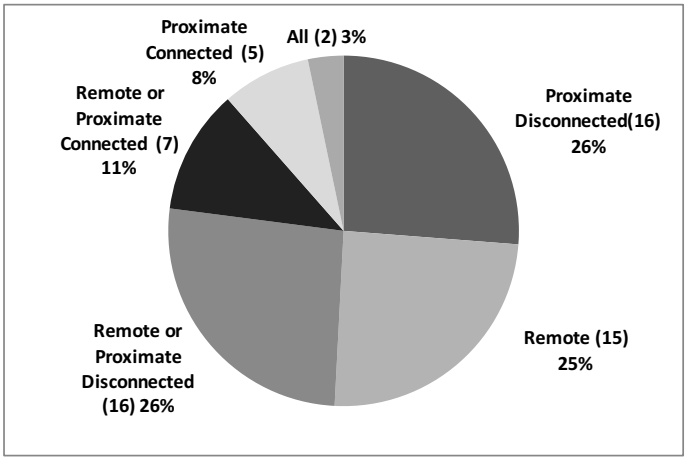

Figure 2.6: Distribution of Systems for the Architectural Decision *Where to Offload*

from using Wi-Fi or short-range radio instead of broadband wireless (e.g., 3G/4G) [10][75]. These systems include Mobile Information Access Architecture for Occasionally Connected Computing [8], Chroma [9], Collaborative Applications [16], Computation and Compilation Offload [18], CloneCloud [22], Kahawai [26], Spectra [41], Trusted and Unmanaged Data Staging Surrogates [42], Cloudlets [52], Single-Server Offloading [56], Scavenger [67], PowerSense [82], Application Virtualization on Cloudlets [84], PARM [88], and VM-Based Cloudlets [108]. Clone-to-Clone (C2C) [7] is a special case because even though the surrogate is proximate and not connected to the enterprise, it is part of an overlay network that enables communication with other mobile devices that are connected to the network. The systems presented in these studies assume that surrogates can function stand-alone (i.e., do not need to be connected to the enterprise in order to provide capabilities), whether they are pre-provisioned (i.e., at system deployment time) or provisioned at runtime from the mobile devices themselves. However, many of these systems could be adapted to work with remote cloud servers or any addressable offload target, but would lose the advantage of lower latency due to proximity.

Tied for the largest set of systems in the studies (also 16/61 or 26%) are those that offload to remote or proximate disconnected resources. These systems include MAUI [26], HPC-as-a-Service [30], OpenCL-Enabled Kernels [33], Real Options Analysis [35], Cuckoo [62], MACS [65], Telemedik [71], AMCO

[72], AIDE [83], Odessa [101], Smartphone-Based Social Sensing [102], IC-Cloud [111], AIOLOS [117], DPartner [129], and Elastic HTML5 [126]. The Offloading Toolkit and Service [121] is a special case because it assumes multiple connected surrogates on which applications can migrate in case of problems. In general, these systems have offload targets that can function stand-alone and are accessible over an IP network, whether local or remote.

The second largest set of systems in the studies (15/61 or 25%) offloads to remote resources. These systems include mHealthMon [2], Mobile Agents [5], Cloud Media Services [19], 3DMA [39], AlfredO [47], Virtual Phone [55], Android Extensions [57], ThinkAir [64], MCo [74], SOME [96], SPADE [112], Mobile Data Stream Application Framework [122], and Heterogeneous Auto-Offloading Framework for Mobile Web Browsers [128]. Weblets [127] is a special case because the architecture contains a cloud elasticity manager that manages a set of cloud nodes and decides where to offload. All these systems assume connectivity to remote cloud resources to offload computation or data. However, unless connectivity to an enterprise cloud is necessary for the system to work, these systems could also offload to proximate connected or disconnected nodes. In fact, the experimentation and validation for many of the solutions is done with a Wi-Fi-connected server instead of the remote enterprise cloud.

The next set of systems (7/61 or 11%) offloads to either remote or proximate connected resources. These systems include Roam [20], Collective Surrogates [48], Grid-Enhanced Mobile Devices [51], ThinAV [59], Resource Furnishing System [92], Cloud Personal Assistant [93], and MAPCloud [103]. The offload targets in these systems need access to a cloud resource in order to operate properly, whether to obtain the code to be offloaded [20], access application data [92], or to offload computation or data to other cloud resources (i.e., surrogate acts as an intermediary) [48][51][59] [93][103].

Five out of 61 systems (8%) offload to only proximate resources that are connected to cloud resources. Edge Proxy [6] requires access to the cloud to obtain data updates. SmartVC [100] and Large-Scale Mobile Crowdsensing [119] obtain the offload code from a cloud resource. Slingshot [114] connects a home server to replicas installed on surrogates so that the home server always has a safe copy of application state. Cloud Operating System [56] is a special case because the surrogate is not connected to the enterprise, but to other surrogates to load balance.

Finally, there are two data staging studies (2/61 or 3%) that can offload to all three options: remote, proximate connected, and proximate disconnected resources. Feel the World [98] and Sonora [120] simply need an addressable offload target, whether proximate or remote.

Most systems in the studies offload to a single known surrogate or cloud resource. The reason for this is that the focus of the studies is on demonstrating the validity or efficiency of portions of the architecture, such as optimization engines or partitioning algorithms, and not the operation of the full system.

Some systems include a component in the architecture to discover and select offload targets. In AlfredO [47] and VM-Based Cloudlets [108], the offload targets use broadcast methods to announce their presence. Mobile Agents [5] queries a cloud directory service to obtain a list of available offload targets in the cloud. Collective Surrogates [48] relies on a surrogate manager to manage available surrogates. Spectra [41] and SPADE [112] maintain a local offload target list. Grid-Enhanced Mobile Devices [51], Resource Furnishing System [92], CAP [93], MAPCloud [103], and Heterogeneous Auto-Offloading Framework for Mobile Web Browsers [128] rely on an application or service directory rather than an offload target directory.

2.5.2.2 When to Offload

Figure 2.7 shows the distribution of systems in the studies for the architectural decision related to where to offload or stage data.

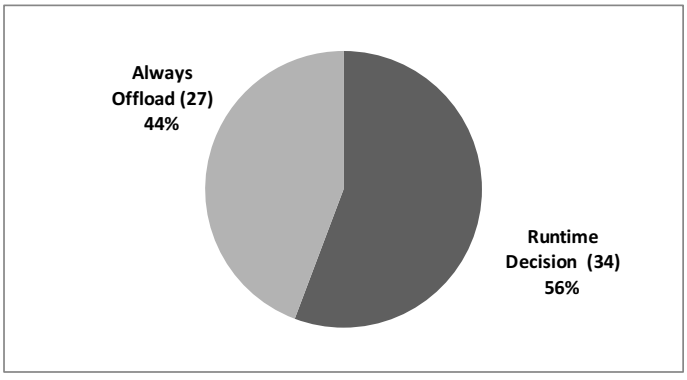

Figure 2.7: Distribution of Systems for the Architectural Decision *When to Offload*

For most of the systems in the studies (34/61 or 56%) offloading is a runtime decision. The majority of these systems perform a runtime calculation (often called a utility function) to determine whether it is better to execute locally or to offload computation by comparing predicted local execution cost against predicted remote execution cost. Local execution cost typically takes

into consideration the energy consumed by local execution as well as the local execution time. Remote execution cost typically considers the energy consumed by communication based on payload size and network conditions, the communication time based on payload size and network conditions, and remote execution time. If local execution cost is lower than remote execution cost then the computation is executed locally; otherwise, it is executed remotely (i.e., offloaded). In systems such as ThinkAir [64] and PowerSense [82], the user can decide to optimize for energy or execution time (e.g., prefer lower energy consumption over lower execution time). In Smartphone-Based Social Sensing [102], the user can also optimize for bandwidth usage.

There are many variations of how a system decides when to offload. Some systems do not perform a runtime calculation, but instead check environmental conditions to determine if offloading should take place. For example, Cuckoo [62] simply checks if the offload target is reachable and, if so, offloads the computation. Roam [20] checks whether a component can be run locally or not due to device constraints and, if not, it offloads the component to a reachable resource that can execute it. Other systems have much more complex methods. For example, Collaborative Applications [16] uses a performance model, a power model, a model and status of offload servers, location of files that are needed by the computation, and network conditions to make the offload decision.

The systems that perform runtime calculations require developer input or static profiling to obtain the initial values or models that are used in the calculation, such as required compute cycles, payload size based on input and output parameters, and required energy for execution and communication. Other parameters such as current network conditions or load of the mobile device and offload target are obtained at runtime.

In addition, some systems use runtime profiling to collect data at runtime to adjust the initial values. The goal is to obtain more realistic values based on actual execution data. SmartVC [100] collects execution time and power consumption and Odessa [101] collects network measurements as historical data to improve the offloading decision. MAUI [26] does continuous device, network, and application profiling to optimize energy efficiency. In Single-Server Offloading [56], values are updated at runtime based on real data using the least squares method only if the prediction error is greater than a certain threshold. In AMCO [72], the runtime system continuously improves its effectiveness due to a feedback-loop mechanism.

The rest of the systems in the studies (27/61 or 44%) always offload computation or data. For computation offload systems, the parts of the system that are considered computation-intensive, or that simply cannot run on a mobile

device, are pre-determined and executed on offload targets. All the data staging systems fall in this category, which is expected, because by definition the idea is for the mobile device to send and receive data to and from an enterprise cloud, either directly or via a surrogate. The decision-making process is not whether it is efficient or not to stage data but when is the right time to do so.

2.5.2.3 What to Offload

As mentioned earlier, there are two architectural decisions related to what to offload for computation offload systems; one is the granularity of the computation that is offloaded and another is the payload that is sent from the mobile device to the offload target in order to execute the offloaded computation. Figure 2.8 shows the distribution of computation offload systems in the studies for the architectural decision related to what to offload for granularity.

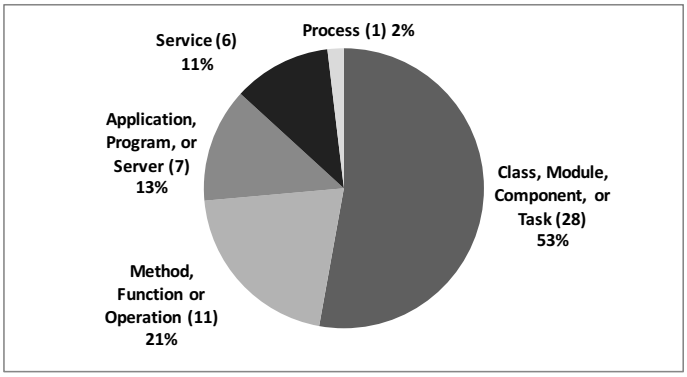

Figure 2.8: Distribution of Systems for the Architectural Decision *What to Offload: Granularity*

All the systems in the studies have an *offload client* that runs on the mobile device and an *offload server* that runs on the offload target that together coordinate the offload operation. The majority of the systems are designed such that the applications at runtime are not aware that computation is being offloaded. What changes between systems based on granularity are the development, build, and runtime dependencies between the offload client and target, as well as the amount of state synchronization required to guarantee the correct execution of applications.

As far as granularity, most systems offload at the class, module, component, or task level (28/53 or 53%). The type of element that is offloaded varies

greatly between systems, but in general they are software elements that execute inside specific containers or runtime environments such as Java Virtual Machines (JVMs), OSGi platforms, or custom-built environments that enable migration between local and remote execution. The advantage of offloading at this level of granularity is that for the most part these are self-contained elements, meaning that they store their own state. Once an element is offloaded there is no need to synchronize state with the local device unless the execution is returning to the local device. However, except for the systems that rely on more standard environments, such as JVMs and OSGi platforms, there are very tight dependencies between the mobile execution environment and the execution environment on the offload target, as shown in Table 2.8. This creates limitations in terms of programming languages and increases the effort required for application reuse because of the need to use specific libraries and constructs to enable computation offload.

Table 2.8: Computation Offload Systems that Offload Classes, Modules, Components or Tasks with Offload Details and Constraints

System	Offload Element	Constraints/Requirements
Mobile Agents [5]	Java classes and/or methods that have been manually marked by the developer as remoteable and packaged as mobile agents	Applications have to be written as mobile agent applications using a development environment such as JADE (Java Agent Development Environment).
Cloud Media Services [19]	Media processing tasks	There are separate specialized VMs running on the offload target that execute media processing tasks (one per task).
Roam [20]	Roamlets (Java components with well-defined interfaces)	All devices have resource descriptions that are used by the runtime optimization algorithm. The developer partitions the application into components and determines how many implementations of the component there will be, and then implements each as a Roamlet.
OpenCL-Enabled Kernels [33]	Kernels (program parts) with OpenCL interfaces	Offload targets are OpenCL-capable.
Real Options Analysis [35]	Tasks (any component with an API)	The mobile device runs the ROA (real options analysis) framework.

Continued on next page

Table 2.8 – *Continued from previous page*

System	Offload Element	Constraints/Requirements
3DMA [39]	Components implemented as Active Objects	Active Object Spaces are created in all mobile devices and servers. In addition, specialized components called workers are implemented in the system to coordinate communication between active objects.
AlfredO [47]	OSGi bundles	The mobile device and offload target are running R-OSGi.
Grid-Enhanced Mobile Devices [51]	Tasks	There is a specialized grid middleware running on the offload target.
Virtual Phone [55]	Android activities	A VM image that matches the mobile device OS is available on the offload target. A FUSE[5] filesystem client and server are installed on the client and offload target respectively.
Cloud Operating System (COS) [56]	Application modules implemented as SALSA actors that can automatically or manually migrate between machines	Applications are written in SALSA and all interconnected surrogates run COS.
Android Extensions [57]	Android programming constructs (activity, service, content provider, or broadcast receiver)	A Mobile Cloud Manager runs on the mobile device and is part of the Android stack. A Mobile Cloud Server runs on the offload target.
Cuckoo [62]	Android services	Applications have to be written and built using the Cuckoo framework. Offloaded code has to be implemented as an Android service. The Ibis middleware is used for remote communication.
MACS [65]	Android services	Applications have to be developed using the MACS (Mobile Augmentation Cloud Services) development framework. Offloaded code has to be implemented as an Android service.
Scavenger [67]	Tasks written in Python, annotated with the tag @scavenge	A daemon that runs on the offload target enables task offload (Stackless Python).

Continued on next page

[5]FUSE stands for Filesystem in Userspace; a mechanism that enables a user to create a filesystem without editing kernel code

Table 2.8 – *Continued from previous page*

System	Offload Element	Constraints/Requirements
AMCO [72]	Components (Java methods, classes, or packages) marked by the programmer as energy hotspots	Programmers know the energy hotspots and annotate the code. Components have loose coupling and high cohesion to avoid back-and-forth dependencies between local and remote code execution.
MCo [74]	Java classes	Mobile applications need to be developed using a development toolkit that provides capabilities for offload to a surrogate called a Master Node that receives computation offload requests from mobile devices and forwards the requests to appropriate Worker Nodes on which migrated classes actually run.
AIDE [83]	Java classes	Requires a modified JVM.
PARM [88]	PARM (power-aware middleware) components	Applications need to be written using the PARM (power-aware middleware) API. Energy profiles are known for all components. A server is set up as a power broker. Devices need to register with the nearest offload target and send state information that is used by the power broker.
Odessa [101]	Data processing stages as defined by the Sprout framework	Applications are implemented using the Sprout framework which structures applications as data flow graphs composed of self-contained stages.
Smartphone-Based Social Sensing [102]	Data classification tasks and subtasks	Applications are implemented using the computation offloading API.
SPADE [112]	Tasks that require file processing that can be run from a command line on the remote machine	Offload target runs the SPADE middleware.
AIOLOS [117]	Java classes that contain methods marked as offloadable	Both the mobile device and the surrogate need to run the OSGi platform because at build time the AIOLOS plugin generates OSGi bundles that publish the annotated classes as OSGi services.
Offloading Toolkit and Service [121]	Java classes	Shadow classes are generated for each instrumented class. Offload targets are interconnected and capable of supporting VM migration.

Continued on next page

Table 2.8 – *Continued from previous page*

System	Offload Element	Constraints/Requirements
Mobile Data Stream Application Framework [122]	Data processing components and application partitioning algorithm	Applications have a pipe-and-filter architecture. There is a centralized resource manager that controls all offload targets. The resource manager assigns an offload target as an application master for a mobile application.
Weblets [127]	Weblets, which are autonomous software entities that run either on the device or the offload target and expose RESTful web service interfaces via HTTP	Weblets are self-contained. The application is developed as specified by Weblet framework. A Cloud Elasticity Manager manages a set of cloud nodes that each run one or more Weblet containers.
DPartner [129]	Java classes packaged as OSGi bundles	The OSGi platform runs on the device and the offload target. A proxy needs to be created for every module.
Elastic HTML5 [126]	Components and functions that leverage the HTML5 concept of web workers, which are background processes that run JavaScript on a web page without blocking the user interface	Proxy web workers exist on the device and the offload target.

The second largest set of systems offloads at the method, function or operation level (11/53 or 21%). In systems such as Collaborative Applications [16], Computation and Compilation Offload [18], MAUI [26], Spectra [41], ThinkAir [64], SOME [96], SmartVC [100] and IC-Cloud [111], developers manually mark the methods that they consider offloadable. Chroma [9] uses a tactic plan to specify the configuration or set of methods that should be offloaded to obtain a particular result (i.e., optimize for a particular parameter). In C2C [7], all code is considered offloadable. In addition to the same types of constraints and requirements for applications and offload targets outlined for the first set of systems, the challenge for this type of system is guaranteeing fidelity of results, which means that executing locally or remotely should produce the same results. Methods, functions, and operations are part of larger programming constructs such as classes or programs that maintain state at runtime, typically expressed as class attributes or global variables. This means that the system has to synchronize state such that it is the same locally and remotely, either periodically or by sending it as an additional input/output of the offload operation.

Systems that offload full applications, programs, or servers of a clien-

t/server system represent the third largest set of systems in the studies (7/53 or 13%). Kahawai [26] offloads the game engine that performs all the GPU computations. Collective Surrogates [48], Cloudlets [52], Application Virtualizaton on Cloudlets [84], VM-Based Cloudlets [108], and Slingshot [114] offload the server portion of a client/server system. Resource Furnishing System [92] offloads full applications and interacts with them using a VNC client.[6] The advantage of offloading at this level of granularity is that execution environments are much more generic, such as virtual machines or application servers. This also increases application reuse because servers do not have to be adapted to run on mobile devices. Clients are very thin and perform the functionality that cannot be offloaded, such as user interface and sensor operations. However, the rest of the computation is always offloaded, regardless of whether or not it would be more efficient to execute on the mobile device.

The fourth largest set of systems in the studies offload services (6/53 or 11%). Services in these studies are coarse-grained capabilities accessed via standardized interfaces that have been identified by system developers as computation-intensive. In mHealthMon [2], these correspond to services that can analyze sensor data. In HPC-as-a-Service [30] these are high-performance-computing (HPC) services. In ThinAV [59], these are antivirus services; because the service is long-running in addition to computation-intensive, the server has a call back to the mobile device. In PowerSense [82], these are image processing services. In CPA [93] they are general cloud services. In MAPCloud [103], services are part of an application request modeled as a workflow. These systems do not have the requirements or constraints of the systems that offload methods or components because by definition services are self-contained. Once a decision is made to offload, the service is invoked and the system either waits for a reply or receives the reply when it is ready.

Finally, CloneCloud [22] is the only system that offloads at the process level (1/53 or 2%). In this system, the mobile device is fully cloned inside a VM running on the offload target. When the system encounters a computation block that is marked for offload, the process enters into a sleep state and process state is transferred from the mobile device to the clone VM. The clone VM integrates the process state, executes the computation block from beginning to end, and then transfers its process state back to the mobile device. The mobile device reintegrates the process state and wakes up the sleeping process to continue its execution. This system allows very fine-grained control of what portions of an application to offload, but requires a very stable network

[6] VNC stands for Virtual Network Computing and is a protocol for remote access to graphical user interfaces

connection to support state synchronization.

Figure 2.9 shows the distribution of computation offload systems in the studies for the architecture decision related to what to offload for payload. Some of the systems have more than one answer to the question, as indicated by multiple entries under What to Offload – Payload in Table 2.6. As an example, the payload in Chroma [9] in Table 2.6 is only invocation parameters but the payload for Collaborative Applications [16] in Table 2.6 is invocation parameters and application state. A category was created for each combination of answers:

- Invocation Parameters
- Computation and Invocation Parameters
- Application State
- Setup Instructions and Invocation Parameters
- Continuous Data from Offload Target to Mobile Device
- Partitioning Algorithm and Invocation Parameters
- Application State and Invocation Parameters
- Device Context and Invocation Parameters
- Source Location, Application State and Invocation Parameters
- Source Location and Invocation Parameters

For the majority of the systems, the payload is the Invocation Parameters used to execute the remote computation (26/53 or 49%). All these systems assume that the offloaded computation already exists on the offload target, which leads to a small payload that simply depends on the size of the parameter data types. However, the systems completely rely on the existence and currency of the offloaded computation on the offload target, which in turn would require more complex deployment processes.

For the next largest set of systems, the payload is Computation and Invocation Parameters (14/53 or 26%). This means that both the actual computation and its invocation parameters are sent from the mobile device to the offload target. What differs between systems is the protocol for getting the computation ready on the offload target. Cuckoo [62], MACS [65], Scavenger [67], SmartVC [100], and AIOLOS [117] first check if the offload target has the computation; if not, the computation is sent for deployment. The rest of the

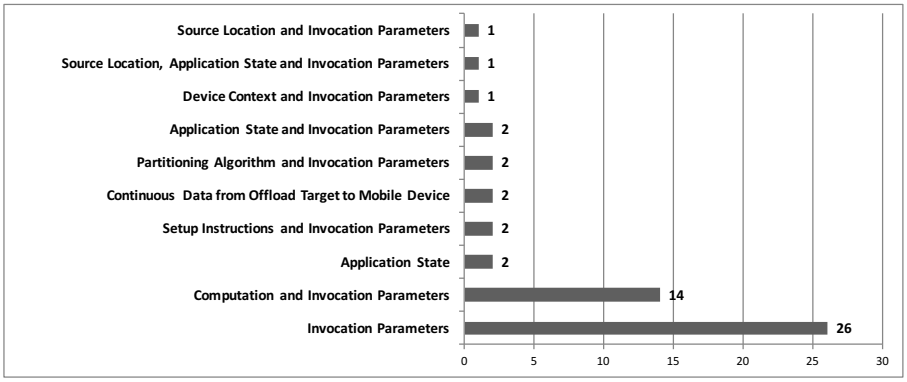

Figure 2.9: Distribution of Systems for the Architectural Decision *What to Offload: Payload*

systems simply send the computation for deployment on the offload target. The offload target deploys the computation inside a container or execution environment, executes it directly in a runtime environment, or distributes it to other offload targets for deployment. Once the computation is running, the mobile device sends the invocation parameters for the actual execution. The exception is the Mobile Agents systems [5] in which what is offloaded is an agent that already contains its invocation parameters.

For the next set of systems, the payload is Application State (2/53 or 4%). In CloneCloud [22], the payload is the application state so that the offloaded process can execute in the offload target with the same state as if it executed on the mobile device. In Virtual Phone [55], Android activity state is saved to a user space file system based on FUSE when an activity is paused for offload. The FUSE client that runs on the mobile device synchronizes state with the FUSE server that runs on the offload target so that the activity can be resumed on the offload target with the same state. In both of these systems, the execution returns to the mobile device and state is synchronized back in the same way.

For a small set of systems, the payload is Setup Instructions and Invocation Parameters (2/53 or 4%). This means that the initial payload is not the computation itself but the instructions of how to set up the computation on the offload target. In Collective Surrogates [48], the payload is a small program which is simply a script that offloads code from the Internet, installs, and runs it. In MAPCloud [103], the payload is an application request modeled as a

workflow of tasks. The offload target (broker) locates offload targets that can perform the tasks and returns a service plan with the URL of each offloaded workflow task. Once the computation is running, the mobile device sends the Invocation Parameters for the actual execution.

In the next set of systems (2/53 or 4%), the payload is Continuous Data from Offload Target to Mobile Device. In Kahawai [26], a system targeted at GPU-intensive applications such as games, the offload target maintains a high-fidelity version of the graphics and a low fidelity version that matches the fidelity of the mobile device. It compares both and sends a compressed video stream of delta frames to the mobile device. The mobile device decompresses the stream and applies the deltas to the frames that it renders locally. In Resource Furnishing System [92], the interaction with the offload target is done via a VNC client which means that GUI updates are continuously sent from the offload target to mobile devices and applied locally.

In addition to Invocation Parameters, two systems offload the Partitioning Algorithm that determines what computation executes locally and what computation is offloaded (2/53 or 4%). These systems are AlfredO [47] and Mobile Data Stream Application Framework [122].

For two systems, the initial payload is local Application State for the mobile device and offload target to synchronize state before invoking the offloaded computation (2/53 or 4%). AMCO [72] sends the state of the nodes that are going to be involved in the computation to the offload target. The offload target synchronizes its state with the received state. The mobile device then invokes the offloaded computation. When a decision is made to return execution to the mobile device, the offload target sends its state to the mobile device so that it is updated. Collaborative Applications [16] uses a suspend/resume approach in which the VM on the mobile device is suspended, state is transferred to the offload target, and then resumed on the server. Once the computation is running, the mobile device sends the Invocation Parameters for the actual execution.

For one system, the initial payload is the Device Context (1/53 or 2%). Cloud Media Services [19] is a system for media processing. The payload is the device context (device type, browser type, supported codecs, screen size, network bandwidth, and latency) such that the appropriate media processing components are selected. Once the computation is running, the mobile device sends the Invocation Parameters for the actual execution.

For one system (1/53 or 2%), Roam [20], the initial payload is the Source Location, or where to obtain the offload computation for installation on the offload target. Application State is then transferred from the mobile device to the offload target. Once the computation is running and the state is syn-

chronized, the mobile device sends the Invocation Parameters for the actual execution.

Finally, for one system, Elastic HTML5 [126], the initial payload is the Source Location (URL) of the offload computation (web worker) and the Invocation Parameters (1/53 or 2%).

As mentioned earlier, there are two architectural decisions related to what to offload for data staging systems; one is the type of data that is being staged and the other is the operations that are offloaded to the offload target to be performed on the data. Figure 2.10 shows the distribution of data staging systems in the studies for the architecture decision related to *What to Offload – Data Type*.

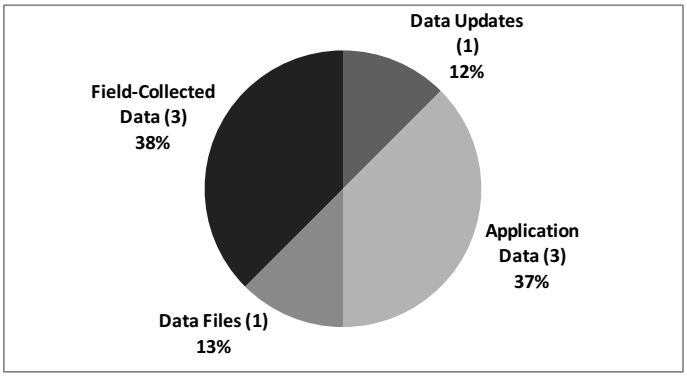

Figure 2.10: Distribution of Data Staging Systems for the Architectural Decision *What to Offload: Data Type*

Field-Collected Data is sent to an offload target for staging in three of the systems (3/8 or 38%). Feel the World [98] and Sonora [120] offload data collected from sensors, either raw or pre-processed. Large-Scale Mobile Crowdsensing [119] offloads raw data collected from sensors. Staging sensor data addresses storage limitations on mobile devices. In addition, data collected by a surrogate can be shared by other mobile devices connected to the same surrogate or can be fused or pre-processed before sending it to the enterprise.

Application Data is staged in three of the systems (3/8 or 38%). Mobile Information Access Architecture for Occasionally Connected Computing [8] stages data from the enterprise that is likely to be used by applications running on the mobile device; it also stages data going to the enterprise in case of disconnected operations. Android Extensions [57] stages data used by ap-

plications on a surrogate that handles all access to the cloud. Telemedik [71] stages data that is likely to be used by applications based on usage patterns or data alerts. The advantage in this case is lower latency because the data resides in a nearby surrogate and not in a remote cloud.

In Edge Proxy [6], the surrogate informs the mobile device when marked areas of a web page have changed. This is a case of data staging from the enterprise in which the mobile device is only notified when there are updates (1/8 or 13%).

Finally, in Trusted and Unmanaged Data Staging Surrogates [42], a surrogate stages files that might be needed by the mobile device (1/8 or 13%). The advantage, as in staging application data, is lower latency because the files reside on a nearby surrogate and not in a remote server. Access to the remote server is done by the surrogate and only when the file is not available on the surrogate (similar to a cache miss) or when data on the surrogate has changed and needs to be consolidated with the data in the remote server.

Figure 2.11 shows the distribution of data staging systems in the studies for the architectural decision related to What to Offload – Data Operations on Surrogate.

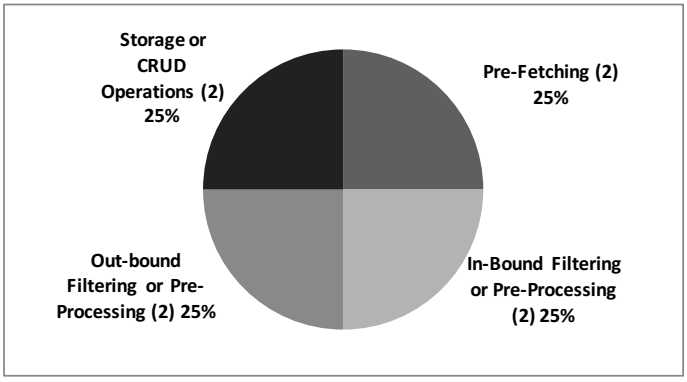

Figure 2.11: Distribution of Data Staging Systems for the Architectural Decision *What to Offload: Data Operations on Surrogate*

Mobile Information Access Architecture for Occasionally Connected Computing [8] and Trusted and Unmanaged Data Staging Surrogates [42] are the two systems that perform pre-fetching operations on the surrogate (2/8 or 25%). Trusted and Unmanaged Data Staging Surrogates [42] executes a prediction algorithm on the surrogate to determine files that are likely needed by

the mobile device using file clusters defined by the user as input (files that are often used together).

Edge Proxy [6] and Telemedik [71] are the two systems that perform inbound filtering or pre-processing of data that flows from the enterprise (or cloud) to the mobile device. The advantage is that the heavy computation or communication to remote servers happens on the surrogates and not on the mobile device. In Edge Proxy [6], the mobile device specifies interest in changes to specific parts of web pages. The surrogate polls the web servers involved, and if relevant changes have occurred, it aggregates the updates as one batch that is sent to the mobile device. In Telemedik [71], a healthcare system, a "context-sensitive priority-based text fragmentation algorithm" determines when and what information to display to the user. This is an example of using data staging to address limited screen size of mobile devices in addition to latency.

Large-Scale Mobile Crowdsensing [119] and Sonora [120] are the two systems that perform out-bound filtering or pre-processing of data that flows from the mobile device to the enterprise (or cloud). In Large-Scale Mobile Crowdsensing [119], sensor data is sent to VMs running on surrogates that can share sensor data with other applications running on the surrogate or format the data to send to applications running in the cloud. Sonora [120] uses a construct called a sync stream that executes on the surrogate and supports disconnected operations, batching, filtering, and compression of data in transit to a remote server.

Finally, Android Extensions [57] and Feel the World [98] use the offload target as an extension of the mobile device's storage system for Data Storage (CRUD operations). Android Extensions [57] provides a set of libraries that can be used to invoke remote services as if they were local services by leveraging the content provider and broadcast receiver Android programming constructs. In Feel the World [98], collected sensor values can be aggregated and/or transformed locally on the client and uploaded to the data staging server in real-time or at a later time. The surrogate gathers and processes the collected values and can provide visualizations of the collected data.

2.6 Main Observations and Findings from Primary Studies

The primary studies show very different and novel computation offload and data staging systems targeted at guaranteeing fidelity of results, and optimiz-

ing attributes such as energy consumption, network bandwidth usage, and performance/response time. For computation offload systems, the offload mechanisms range from dynamic approaches in which the computation is provisioned from the mobile device to more static approaches in which the computation already exists on the offload target. For data staging systems, the capabilities of the offload target range from an extension of the mobile device's storage to sophisticated algorithms that predict and stage the data that will likely be needed by the mobile device. As far as distribution, the number of computation offload systems (53) is much larger than the number of data staging systems (8).

Analysis of the data shows the following gaps and opportunities for architectural strategies for cyber-foraging systems.

- Lack of understanding of non-functional requirements beyond energy, performance, network usage, and fidelity of results: One of the main challenges of building cyber-foraging systems is the dynamic nature of the environments in which they operate. For example, the connection to an external resource may not be available when needed or may become unavailable during a computation offload or data staging operation. As another example, multiple external resources may be available for a cyber-foraging system, but not all have the required capabilities. Many of the cyber-foraging systems, especially those that perform runtime partitioning and offloading decisions, have very complex and thorough algorithms for guaranteeing fidelity of results, and optimizing energy consumption, network bandwidth usage, and performance/response time. Disconnected operations and fault tolerance are supported by some systems in which the local computation is a fallback mechanism if the remote computation fails. However, there is very little consideration to other non-functional requirements that are relevant to cyber-foraging systems, such as ease of distribution and installation, resiliency, and security. Understanding these equally-important non-functional requirements is key to reasoning about the behavior of a cyber-foraging system in light of an uncertain operating environment.

- Lack of focus on system-level concerns: Related to the previous point, the systems in the studies tend to have a very narrow focus on proving that cyber-foraging is possible between one mobile device and one offload target, which is a very limited view of a real, operational cyber-foraging system. There is very little discussion of system-level concerns that have to be addressed when moving from experimental prototypes to operational systems; for example:

  - How do the systems perform when there are multiple devices trying to offload to the same target?
  - If there are multiple offload targets available, how does the mobile device select the target that best fits its requirements?
  - What happens if the mobile device loses connectivity to the offload target?
  - In those mechanisms that require custom infrastructures or middleware, what are the mechanisms for ensuring currency and compatibility of mobile-side and server-side components if these may not have the same distribution mechanisms?
  - How does a mobile device know that a discovered surrogate is trustworthy?
  - What are the tradeoffs between the non-functional requirements promoted by the system and other non-functional requirements such as ease of distribution and installation, resiliency, and security?

- Lack of large-scale evaluations: Most of the studies have very limited case studies or evaluations. For example, even though studies talk about mobile cloud computing, the experiments are done in controlled environments over Wi-Fi connections, which is not representative of a real mobile cloud environment with disconnections, high latency, and multiple heterogeneous users and devices. Large-scale evaluations or simulations would generate knowledge that would enable developers of cyber-foraging systems to understand the implications and design decisions to deal with operational environments.

- Small number of studies on architectures for data staging systems: The low number of primary studies related to architectures for data staging, combined with an increasing number of data collection devices in the field, show that it is a potential area for developing architectural patterns or tactics that can be leveraged by software architects and developers of these types of systems.

2.7 Related Work

There are several studies that survey the field of mobile cloud computing and identify cyber-foraging as a research area and challenge, but are not systematic literature reviews and do not have an architectural focus. Abolfazli et al

[1] present a survey of cloud-based mobile augmentation (CMA) approaches, one of which is cyber-foraging. One of the challenges stated by this work is the lack of a reference architecture for CMA. Dinh at al [28] present a survey on mobile cloud computing (MCC). Computation offload is discussed as a technique for extending battery lifetime of mobile devices and listed as one of the challenges for MCC. Fernando et al [38] present a more complete survey on MCC. Some of the research that addresses efficient computation offload and distribution to the cloud and how it differs from traditional distributed systems is discussed in this paper. Lomotey at al [80] present an additional survey on MCC and start introducing some of the challenges of ubiquitous cloud computing (UCC), defined as consistency in cloud service access from multiple mobile devices owned by a single user. Computational offloading from mobile nodes to middle-tier servers (i.e., surrogates) is mentioned as one way to overcome energy and latency limitations of offloading to remote clouds in this paper. Kumar et al [69] present a survey on computation offloading, but focus primarily on the algorithms used to partition and offload programs in order to improve performance or save energy. Finally, Yu et al [123] present a survey on seamless application mobility, which is the continuous or uninterrupted computing experience as a user moves across devices. Code offloading is mentioned as a future direction for seamless application mobility.

The work that is most similar to ours is by Flinn et al [40] that presents a discussion of representative cyber-foraging systems and their characteristics. However, it is limited to a small number of systems and does not follow a systematic process. To the best of our knowledge, ours is the first systematic literature review related to architectures for cyber-foraging.

2.8 Summary and Conclusions

This chapter presented the results of an SLR in architectures for cyber-foraging systems in the context of RQ1, which is to determine what software architecture design decisions for cyber-foraging systems can be identified in the literature.

We identified 58 primary studies, containing a total of 61 systems (53 computation offload systems and 8 data staging systems). The systems were analyzed using a categorization of architecture decisions related to what, when, and where to offload computation and data from mobile devices. While most of these systems presented very novel methods for computation offload and making runtime decisions, there was very little detail on how the systems would function in a real operational setting. In particular, the analysis allowed us to

identify gaps and opportunities for research in (1) non-functional requirements that are relevant to operational cyber-foraging systems, such as ease of distribution and installation, resiliency, and security, (2) system-level architecture analysis, (3) large-scale evaluations, and (4) architectures for data staging systems. Of particular interest was the small number of data staging systems in the studies. Given the data presented in Section 1.1 related to trends such as IoT, demanding content types, and increasing mobile traffic, we would have expected to see more data staging systems in the studies addressing some of these problems.

The next chapter presents a set of architectural tactics derived from the architecture decisions identified in the primary studies. The goal of the tactics is to provide reusable elements for architects of cyber-foraging systems to reason about quality attributes necessary for deployment in operational environments, beyond energy efficiency, response time, and fidelity of results.

3

Architectural Tactics for Cyber-Foraging

This chapter presents a catalog of architectural tactics for cyber-foraging that was derived from the results of the systematic literature review on architectures for cyber-foraging systems presented in Chapter 2. Elements of the architectures identified in the primary studies were codified in the form of Architectural Tactics for Cyber-Foraging. These tactics will help architects extend their design reasoning towards cyber-foraging as a way to support the mobile applications of the present and the future.

3.1 Introduction

Architectural tactics are design decisions that influence the achievement of a quality attribute response [13]. The tactics in this chapter were extracted from the primary studies based on (1) common components found in the studies, (2) quality attributes explicitly stated in the studies, and (3) quality attributes inferred from system and component descriptions.

Figure 3.1 presents the set of identified tactics. The top levels of the figure are the tactic categories. The boxes with solid lines under each category are the tactics. A box with a dashed line under a tactic is a variation of that tactic. Each tactic is described using the following template:

- Motivation: rationale behind the tactic
- Description: components introduced by the tactic and explanation of their roles
- Constraints: necessary conditions for applying the tactic in an existing software architecture

- Example: application of the tactic in one or more systems; the example(s) will map back to the elements of the architecture diagram presented in the description
- Dependencies: other tactics required by the tactic
- Variations (Optional): slight variations of the tactic

The tactics are divided into functional and non-functional tactics. Functional tactics are broad and basic in nature and correspond to the architectural elements that are necessary to meet cyber-foraging functional requirements. Non-functional tactics are more specific and correspond to architecture decisions made to promote certain quality attributes. Non-functional tactics have to be used in conjunction with functional tactics.

The tactics described in this section will include a surrogate as the offload target, as depicted in Parts (b) and (c) of Figure 2.5. The notion is that the elements of the tactic that apply to the surrogate will also apply to a remote cloud server.

3.2 Functional Architectural Tactics for Cyber-Foraging

Functional architectural tactics correspond to basic capabilities of a cyber-foraging system. All the systems in the studies presented in Chapter 2 contained at least the following combination of tactics:

$$(\textit{Computation Offload} \vee \textit{Data Staging}) \wedge \textit{Surrogate Provisioning} \wedge \textit{Surrogate Discovery}$$

This means that all cyber-foraging systems contain

- A tactic for computation offload or data staging (or both)
- A tactic for provisioning the surrogate with the offloaded computation or data processing capabilities
- A tactic for a mobile device to locate a surrogate for offload or data staging

The following sections describe these basic functional architectural tactics.

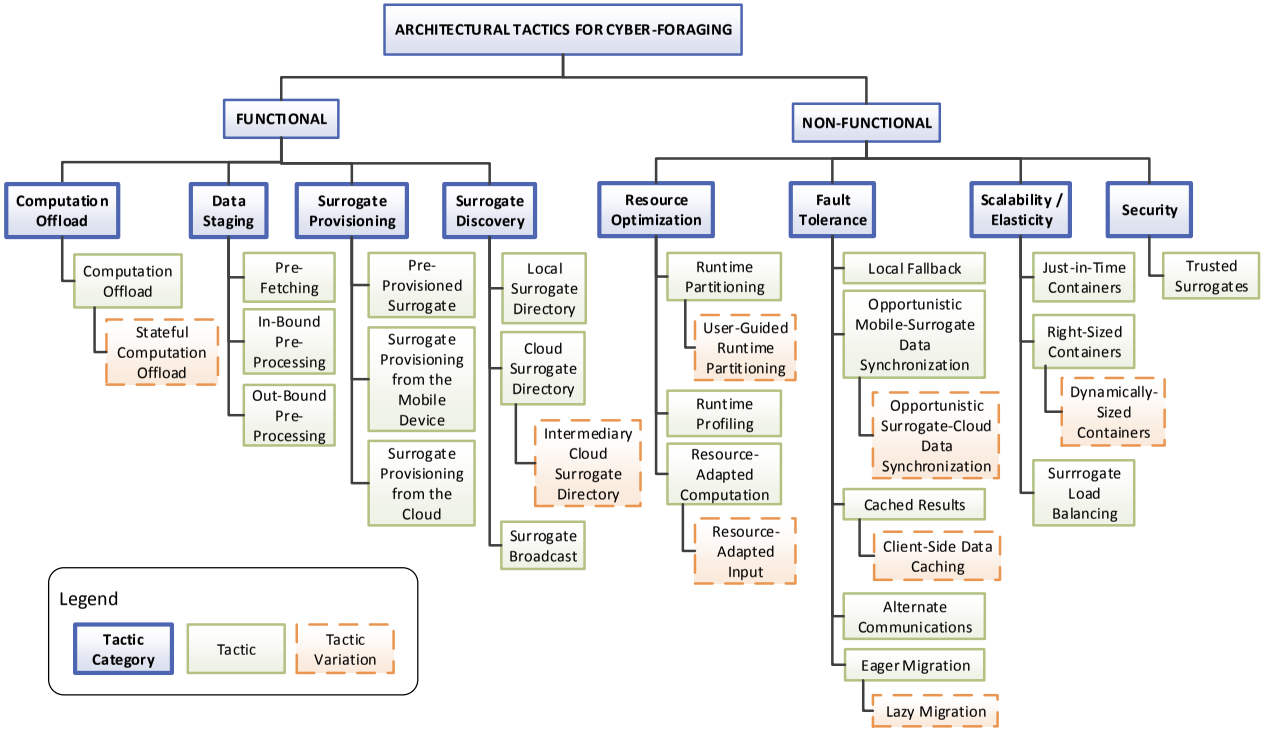

Figure 3.1: Architectural Tactics for Cyber-Foraging

3.2.1 Computation Offload

A scenario for Computation Offload from a mobile device to a surrogate is the following: The user of a mobile device executes a cyber-foraging-enabled mobile application. The application offloads the computation to a nearby surrogate with minimal disruption to the mobile device user.

The Computation Offload tactic can be found in the computation offload systems shown in Table 2.6 for which *What to Offload - Granularity* is Component, Service, or Application. It can also be mapped to the data staging systems in Table 2.7 for which *What to Offload - Data Operations* corresponds to Storage because even though these systems are using the surrogate for extended storage, what they are really offloading is the data management computation.

Motivation. Mobile devices still do not have the computing power and battery life that will allow them to perform effectively over long periods of time, or to execute applications that require extensive communication or computation. Computation Offload extends battery life by offloading computation-intensive portions of an application to nearby surrogates with greater computation power. In addition, the single-hop proximity of surrogates combined with the use of WiFi or short-range radio instead of broadband wireless (e.g., 3G/4G) also decreases latency [10][75] and improves the user experience especially for highly-interactive applications.

Description. Figure 3.2 shows the main components of this tactic with numbers that indicate the sequence of operations. The Computation Offload tactic requires an *Offload Client* running on the *Mobile Device* and an *Offload Server* running on the *Surrogate*. This pair of components communicates to coordinate the offload operation. The *Cyber-Foraging Enabled Mobile App* invokes the *Offload Client* when it encounters a portion of code that has been identified as offloadable computation and passes it any *App Metadata* that is required to set up the *Offloaded Code*. The *Offload Client* then coordinates with the *Offload Server* to set up the*Offloaded Code* so that it can be invoked by the *Cyber-Foraging Enabled Mobile App*. The *Offloaded Code* runs inside a *Container* on the *Surrogate*. Examples of a *Container* are a virtual machine, application server, web server, or the operating system. Figure 3.2 shows the *Cyber-Foraging Enabled Mobile App* communicating directly with the *Offloaded Code*. An alternative is for the *Cyber-Foraging Enabled Mobile App* to always communicate through the *Offload Client*. This latter alternative has the potential for performance problems as the number of mobile clients using the surrogate increases. This is because the *Offload Server* becomes a bottleneck as all communication would go through this component.

However, some systems that implement Fault Tolerance tactics (Section 3.3.2) place the responsibility of detecting and managing disconnections in the *Offload Client* and *Offload Server* which therefore benefit from the single point of communication of the latter alternative.

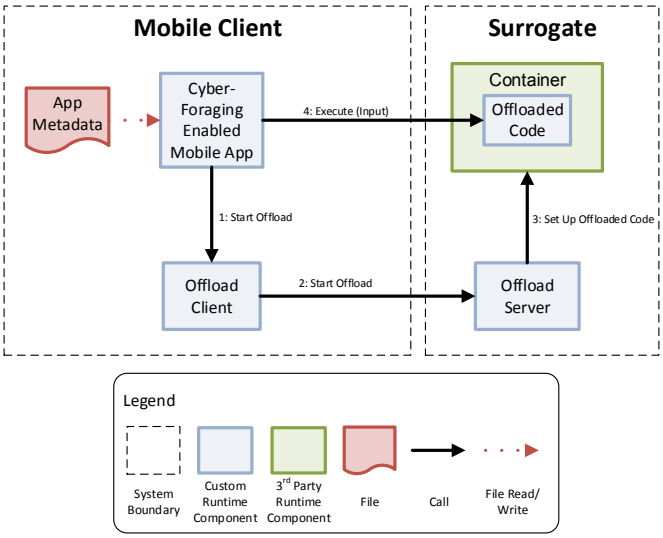

Figure 3.2: Computation Offload Tactic

Constraints. The tactic as described assumes that offloaded computation already exists on the surrogate (provisioned via the application of a Surrogate Provisioning tactic (Section 3.2.3)) and that the surrogate is always available.
Example. An example of how to apply the Computation Offload tactic is the Mobile Agents system [5] shown in Figure 3.3. In the Mobile Agents system applications are manually partitioned into components that have to be executed locally and components that can be offloaded. These offloadable components are set up as *Mobile Agents* using the Java Agent Development Environment (JADE). At runtime, the *Execution Manager* determines if the agent marked as offloadable should be offloaded based on a comparison of local and remote execution times (Section 3.3.1.1 contains details on runtime partitioning). If so, the *Execution Manager* sends the *Mobile Agent* (which carries its input parameters) to the *Agent Management System* so that it can migrate the *Mobile Agent* to the *JVM Container* in the *Cloud Host*. After migration, the offloaded component starts executing and communicates directly with the

Mobile App.

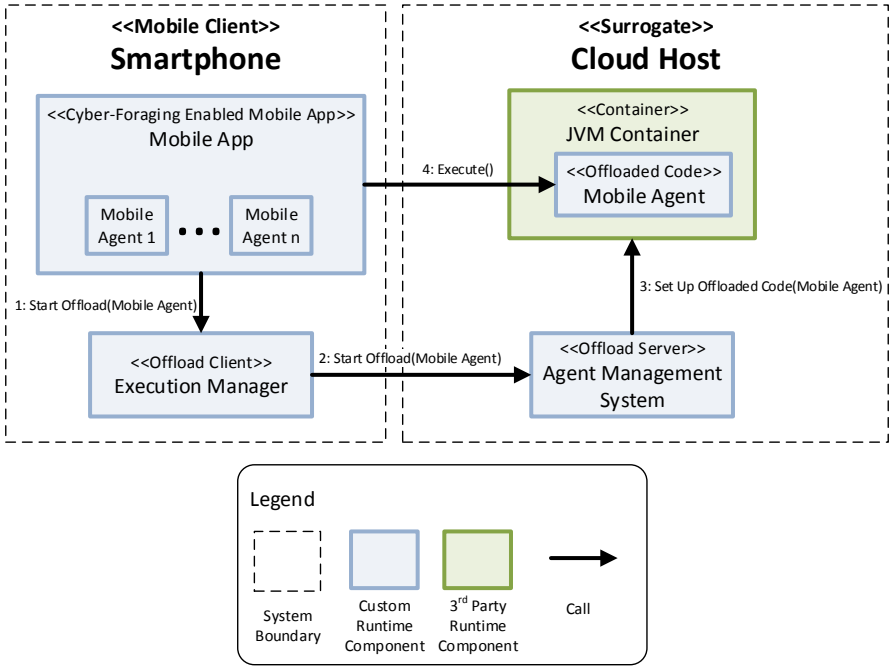

Figure 3.3: Mobile Agents as an Example of the Computation Offload Tactic

Dependencies. The Computation Offload tactic needs to be combined with a Surrogate Provisioning tactic (Section 3.2.3) that prepares the surrogate for computation offload. It also needs to be combined with a Surrogate Discovery tactic (Section 3.2.4) to discover surrogates in the environment. This tactic is also often combined with non-functional tactics to achieve desired system qualities. For example, it is often combined with Resource Optimization tactics (Section 3.3.1) to make better decisions on resource usage and with Fault Tolerance tactics (Section 3.3.2) to attempt to provide continued operations.
Variation: Stateful Computation Offload. The tactic as described assumes that the offload operation is stateless, which means that no mobile app state needs to be transferred between the *Offload Client* and the *Offload Server* during the offload operation. This is what happens when the granularity of the offload operation is a module or class, a service, or a complete application (or server portion of an application) because offloaded code is self-

contained. When the granularity of the offload operation is at the process or at the method level, the state of the program or object that contains the process or method being offloaded has to be transferred to the equivalent program or object on the surrogate. In this case, a state synchronization operation in a *State Manager* component that is invoked either periodically or on-demand has to execute before the offloaded code is executed to guarantee that the state is equivalent on both sides. This stateful variation of the tactic can be mapped to the computation offload systems in Table 2.6 for which *What to Offload - Granularity* corresponds to Process or Function.

An example of how to apply the Stateful Computation Offload tactic is the CloneCloud system [22] shown in Figure 3.4, marked with numbers that indicate the sequence of operations. In CloneCloud, the *Container* on the *Surrogate* is a *Clone Application VM* of the *Application VM* that is executing on the mobile device. At runtime, when a computation block of the *Instrumented Mobile App* is marked for offload, a *Migrator* component running in the VM is invoked that puts the running process into a sleep state and transfers this state to the *Clone Application VM* via the pair of *Node Managers* running on both the mobile device and the surrogate. The *Migrator* in the *Clone Application VM* creates a new process with the received state and marks it as runnable so it executes. The cloned process executes from the beginning of the computation block until it reaches the end of the computation block. The *Migrator* on the cloned VM then transfers the new process state back to the mobile device. The *Migrator* on the mobile device receives the new process state, merges it with the sleeping process, and then wakes up the sleeping process to continue its execution.

3.2.2 Data Staging

A scenario for Data Staging is the following: A mobile application is being used by multiple users to collect data in the field. Upon detection that it is close to a surrogate, the mobile application offloads the collected data. When the operation is complete, the mobile device deletes the transmitted data to free up storage space. In addition, when the surrogate establishes connectivity to the main data center in the cloud, it forwards the data that was collected by the multiple users, where it is integrated into the enterprise data repository. An additional capability of the application is to provide data visualizations pertaining to the data collected by the user, the data collected in the region that is served by the surrogate, and the data collected by the entire set of users. Therefore, data is pushed from the enterprise data center to the surrogate either on-demand or periodically so that the data is closer to the user and

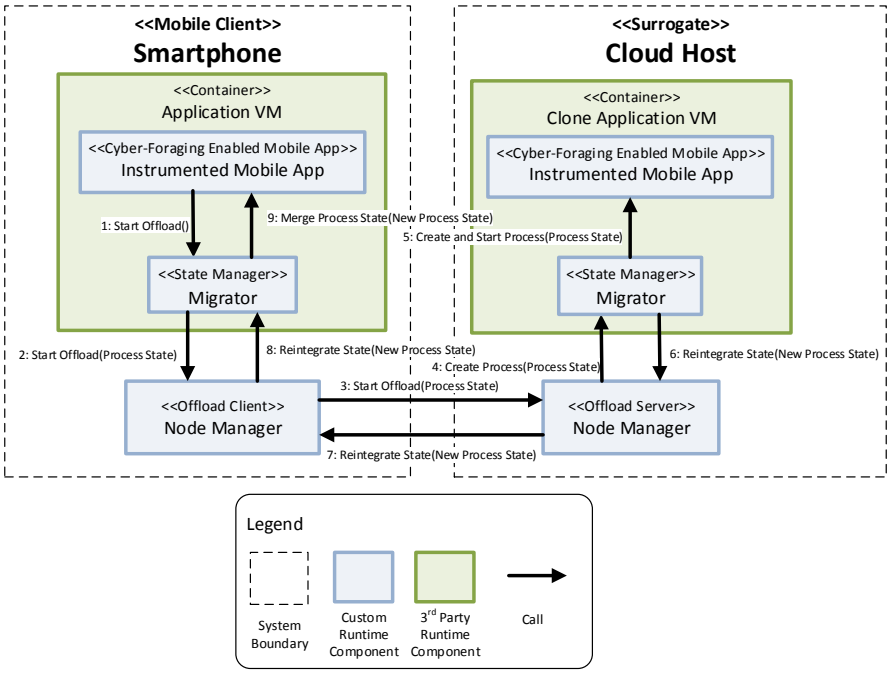

Figure 3.4: CloneCloud as an Example of the the Stateful Computation Offload Tactic

accessible even if the surrogate is disconnected from the enterprise.

The Data Staging tactics require a configuration such as the one shown in Part(c) of Figure 2.5 in which the mobile device is connected to a surrogate and the surrogate is connected to the enterprise or cloud data center, even if connectivity is intermittent or periodic.

3.2.2.1 Pre-Fetching

The Pre-Fetching tactic can be found in the data staging systems shown in Table 2.7 for which *What to Offload - Data Operations* is Pre-Fetching.
Motivation. Data-intensive mobile apps often rely on data located in the cloud. However, access to this data is likely over a lower-bandwidth and multi-hop connection, compared to the higher-bandwidth, single-hop connection that exists between a mobile device and a surrogate. Pre-fetching anticipates data

needs in order to minimize communication to the cloud and reduce latency. 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. Access to the cloud is therefore only necessary when the data is not already available on the surrogate.

Description. Figure 3.5 presents the main components of this tactic. The Pre-Fetching tactic requires a *Data Staging Client* that runs on the *Mobile Client* and a *Data Staging Manager* that runs on the *Surrogate*. The *Data Staging Client* handles all data operations on behalf of a *Cyber-Foraging-Enabled Mobile App*. Before sending the data operation to the *Data Staging Manager*, the *Data Staging Client* captures and also sends along any *Pre-Fetch Hints* that are used by the *Pre-Fetch Algorithm* to determine and anticipate data needs. Examples of pre-fetching hints include mobile device location, user profile and preferences, and the user's schedule. The*Data Staging Manager* first executes the data operation against the local *Cache*. If the operation is successful it returns the results of the data operation. If the operation is not successful the *Data Staging Manager* obtains the data from the *Cloud Data Repository* in the *Enterprise Cloud* (or the equivalent of a master data repository), stores it in the local *Cache*, and returns the results of the data operation to the *Mobile Client*. Asynchronously, either periodically or triggered by certain conditions, the *Data Staging Manager* will use the *Pre-Fetch Hints* from the *Mobile Client* and any local data such as the user's access history as parameters to a *Pre-Fetch Algorithm* that will calculate the data set that is likely to be needed next by the *Cyber-Foraging Enabled Mobile App*. It will then retrieve this data set from the *Cloud Data Repository* and store it in the local *Cache* so that it is available when it is needed by the *Cyber-Foraging Enabled Mobile App*. Similarly, either periodically or in response to certain conditions, that *Data Staging Manager* will sync the *Cache* with the *Cloud Data Repository* to ensure that data is consistent locally and remotely.

Constraints. The tactic as presented requires connectivity between the mobile device and the surrogate for access to any data that is being staged, and eventual connectivity between the surrogate and the enterprise cloud to serve cache misses and synchronize data. The tactic also assumes that there is a mechanism in place, either manual or automatic, to resolve any synchronization conflicts between the Cache and the Cloud Data Repository, especially if cached data is not read-only.

Example. An example of how to apply the Pre-Fetching tactic is the Trusted and Unmanaged Data Staging Surrogates system [42] shown in Figure 3.6. Data is staged on a *Staging Server* in the *Surrogate*. A *Client Proxy* running on the *Wimpy Client* intercepts all data operations. If it detects high latency

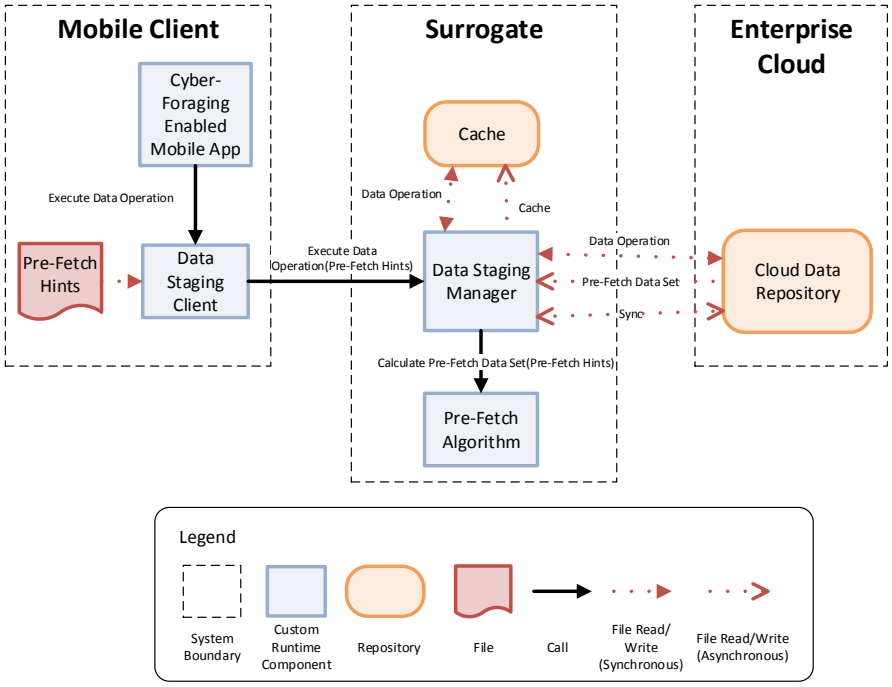

Figure 3.5: Pre-Fetching Tactic

it sends the data operation to the *Surrogate*, which then uses a pre-defined *User Role* to determine the initial set of files that the user is going to need based on this role. The *User Role* basically establishes the set of files that are commonly used together. The *Staging Server* obtains the set of files from the *File Server* and caches them on the surrogate.[1] After the *Cache* has been loaded with the initial data set, all data operations are routed to the *Staging Server*. If the requested file exists in the *Cache* then the data operation takes place locally on the *Surrogate*. If the file is not available in the *Cache* it obtains the file from the *File Server* and stores it in the *Cache*, along with any other files that are predicted to be required based on the request.

Dependencies. The Pre-Fetching tactic needs to be combined with a Surrogate Provisioning tactic (Section 3.2.3) that prepares the surrogate for data

[1]For simplicity, the desktop and its trusted authority role are not included in the discussion of this tactic but are addressed in Section 3.3.4.1.

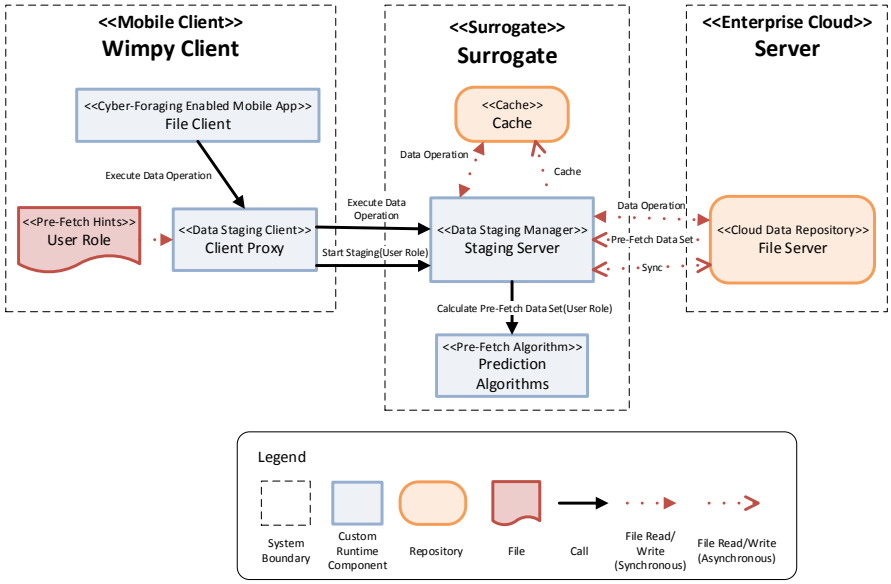

Figure 3.6: Trusted and Unmanaged Data Staging Surrogates as an Example of the Pre-Fetching Tactic

staging and with a Surrogate Discovery tactic (Section 3.2.4) to discover surrogates in the environment. This tactic is also often combined with other non-functional tactics to achieve desired system qualities. For example, it is often combined with Fault Tolerance tactics (Section 3.3.2) to attempt to provide continued operations.

3.2.2.2 In-Bound Pre-Processing

The In-Bound Pre-Processing tactic can be found in the data staging systems shown in Table 2.7 for which *What to Offload - Data Operations* is In-Bound Processing.

Motivation. Data-intensive mobile apps often rely on data that resides in the cloud. However, access to this data is likely over a lower-bandwidth and multi-hop connection, that in addition consumes more energy than the single-hop connection that exists between a mobile device and a surrogate. In order to reduce the amount of data received by the mobile device, avoid direct communication to the cloud for every data operation, and avoid the computation

costs of processing this data for visualization on mobile devices, the surrogate pre-processes the data that is retrieved or pushed from the enterprise cloud. The mobile device receives data that is ready to be consumed, or filtered such that it only receives data of interest or relevance.

Description. Figure 3.7 shows the main elements of the In-Bound Processing tactic. This tactic requires a *Communications Manager* that runs on the *Mobile Client* and handles all communication with the *Data Processor* on the *Surrogate*. The *Mobile Client* can request data on demand or periodically (synchronous) or can register with the surrogate for data of interest (asynchronous). In the case of synchronous requests, as shown by the S# operations in Figure 3.7, the *Cyber-Foraging-Enabled Mobile App* requests data via the *Communications Manager*. The *Data Processor* retrieves the data from the *Cloud Data Repository* and pre-processes it according to defined algorithms/rules before sending the data to the mobile app. The *Data Processor* may store data in its local *Cache* for additional processing, to serve additional requests based on the same data, or if the algorithm/rules involve partitioning or priorization of data such that it is sent incrementally upon request. In case of asynchronous requests, as shown by the A# operations in Figure 3.7, the *Cyber-Foraging-Enabled Mobile App* registers for data of interest via the *Communications Manager*. The *Data Processor* periodically polls the *Enterprise Cloud* for the data of interest (e.g., new data, updated data, data conditions satisfied) and when conditions are met it sends the data asynchronously back to the mobile app using some form of callback mechanism.

Constraints. The tactic as presented requires connectivity between the mobile device and the surrogate for access to any data that is being staged, and connectivity between the surrogate and the enterprise cloud to receive data as required.

Example. An example of how to apply the In-Bound Pre-Processing tactic is the Edge Proxy system [6] shown in Figure 3.8. The Edge Proxy system uses a surrogate called an *Edge Server* to monitor changes in web pages on behalf of a *Web Browser* running on the *Mobile Device*. The user marks areas of interest on a web page (e.g., stock prices, temperature, news) and sends them to an *Edge Proxy* running on the *Edge Server* via the *Mobile Proxy*. The *Edge Proxy* saves the current state of the web page along with the areas of interest in its *Cache*. The *Edge Proxy* then does high-frequency polling of the web page on the *Web Server* and notifies the *Mobile Device* if it detects a change in the areas of interest compared to the cached web page. Instead of sending separate messages for the web page and its embedded objects, the *Edge Proxy* bundles the web page with all its embedded objects in a single batch update message, further reducing the amount of communication between the *Mobile*

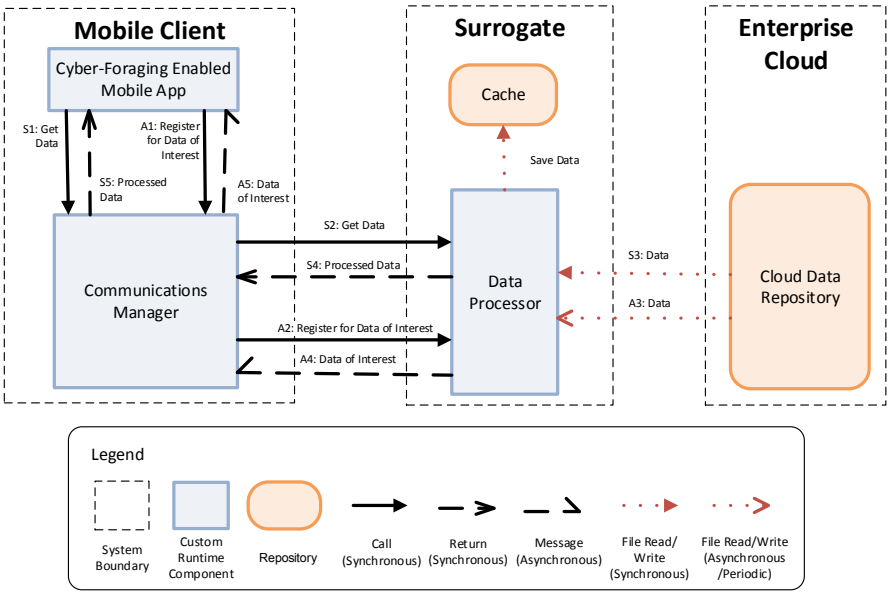

Figure 3.7: In-Bound Pre-Processing Tactic

Device and the *Surrogate*.

Dependencies. The In-Bound Pre-Processing tactic requires a Surrogate Provisioning tactic (Section 3.2.3) that prepares the surrogate for data staging.

3.2.2.3 Out-Bound Pre-Processing

The Out-Bound Pre-Processing tactic can be found in the data staging systems shown in Table 2.7 for which *What to Offload - Data Operations* is Out-Bound Processing.

Motivation. Data-intensive mobile apps are often used to collect data in the field, where Internet connectivity might not be available to mobile devices or might be costly. In addition, although the field-collected data is valuable, it might be overwhelming for a device to transmit all data collected to the enterprise, especially if Internet connectivity is a scarce resource. In these cases, a surrogate can pre-process – clean, filter, summarize, or merge – the data that is received from the mobile devices that it serves such that the data that is sent on to the enterprise cloud is ready for consumption and serves an immediate need. Complete data from the mobile device and/or the surrogate

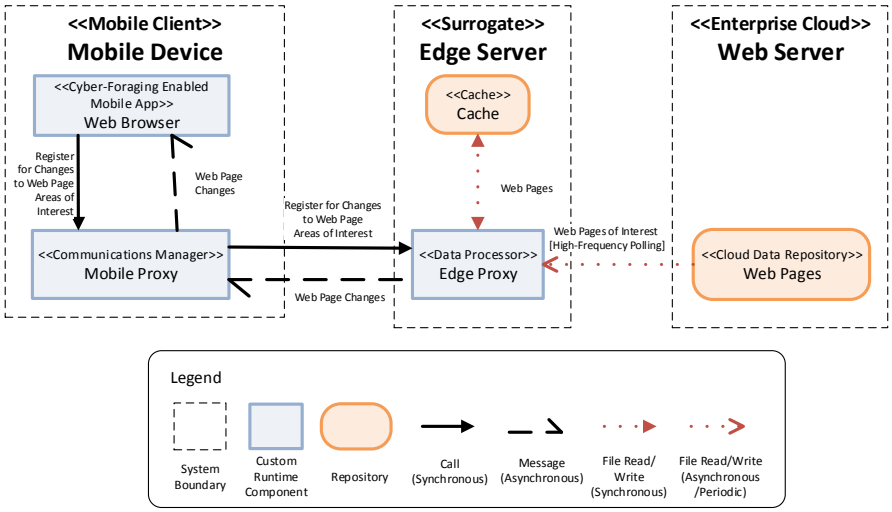

Figure 3.8: Edge Proxy an Example of the In-Bound Pre-Processing Tactic

can be uploaded to the cloud when network connectivity is available.

Description. Figure 3.9 shows the main components of the Out-Bound Pre-Processing tactic. This tactic requires a *Mobile Sensing App* that uses a *Communications Manager* on the mobile device to buffer data to send to its counterpart on the *Surrogate*. The *Communications Manager* can also batch data according to user or application preferences to conserve the energy spent on turning the radio on and off for communication. The *Communications Manager* on the *Surrogate* receives the data and stores it in a local *Cache*. One or more *Data Processing Applications* on the *Surrogate* can either subscribe to data coming in from the *Mobile Device*, perform continuous processing and forwarding of the data as it is coming in, or provide on-demand capabilities to other mobile devices being served by the same surrogate or cloud applications.

Constraints. The tactic as presented requires eventual connectivity between the mobile device and the surrogate to offload data captured in the field and eventual connectivity between the surrogate and the enterprise cloud to offload data that is staged on the surrogate.

Example. An example of how to apply the Out-Bound Processing tactic is the Large-Scale Mobile Crowdsensing system [119]. Crowdsensing refers to individuals using mobile devices with sensors that share information about an event or task of interest such as environmental monitoring, public safety, traffic

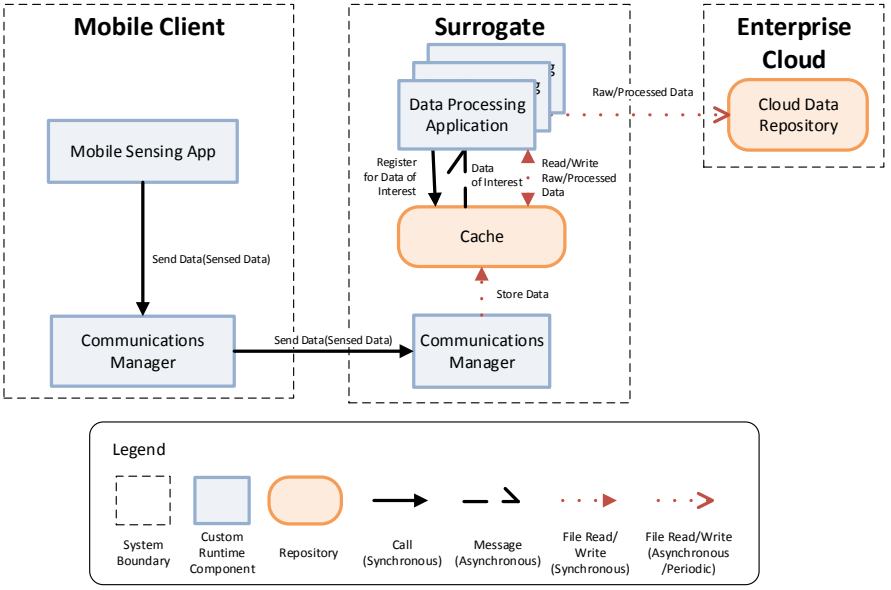

Figure 3.9: Out-Bound Pre-Processing Tactic

monitoring, or collaborative searches. As shown in Figure 3.10, the Large-Scale Mobile Crowdsensing system relies on a single *Crowdsensing Participation App* to gather data from one or more sensors on the *Mobile Device* and create a *Data Sensing Stream* that is sent to a *Proxy VM* on a surrogate called a *Cloudlet*. The *Proxy VM* serves the role of both *Communications Manager* and *Cache* and is essentially a proxy of the mobile device that handles all requests for sensor data on behalf of the mobile device. A *Cloudlet* can run one or more *Proxy VMs* that each corresponds to a mobile device that is participating in a crowdsensing task. In addition, the *Proxy VM* can perform processing on the data sensing stream to for example enforce privacy settings. One or more *Crowdsensing Application VMs* that also run on the surrogate access the *Proxy VM* to obtain the sensed data to process locally or to format and send the data to applications running in an *Application Server* in the cloud.

Dependencies. The In-Bound Pre-Processing tactic requires a Surrogate Provisioning tactic (Section 3.2.3) that prepares the surrogate for data staging.

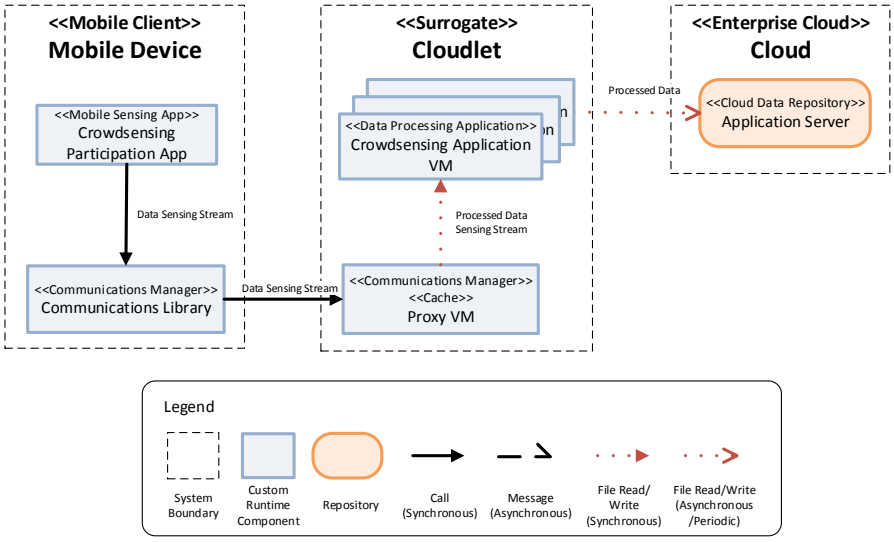

Figure 3.10: Large-Scale Mobile Crowdsensing as an Example of the Out-Bound Pre-Processing Tactic

3.2.3 Surrogate Provisioning

To be able to use a surrogate for cyber-foraging, it has to be provisioned with the offloaded computation and/or the computational elements that enable data staging. A scenario for surrogate provisioning is as follows: a mobile device needs to execute a computation-intensive task. Instead of executing the task locally, it locates a surrogate and sends it a request to execute the computation on its behalf. The surrogate first checks if it already has the computation to support the task. Because it does not, it sees if it can locate the computation in a cloud repository. Because the surrogate is not able to locate the capability in the cloud, the mobile device sends the computation to the surrogate for installation. Once the surrogate installs and starts the computation it notifies the mobile device that it is ready, executes the computation, and sends back the results of the computation.

3.2.3.1 Pre-Provisioned Surrogate

Many of the systems described in the primary studies assume that the offloaded computation and/or data staging elements are already installed (pre-

provisioned) on the surrogate at deployment time. The computation offload systems shown in Table 2.6 that make this assumption are those for which *What to Offload -Payload* is (1) Parameters but not Computation, Source Location nor Setup Instructions, (2) Application State, (3) Device Context, or (4) Continuous Data. It is also true of all the data staging systems shown in Table 2.7. However, for these systems, there is no detail of how the surrogates were provisioned with the necessary offloaded computation and or data staging elements. This observation relates to the SLR finding in Section 2.6 that states that most systems tend to focus on the algorithms and implementation details for enabling cyber-foraging and not on system-level attributes such as ease of distribution and installation that have to be considered when moving from experimental prototypes to operational systems. Indeed, a cyber-foraging system could be implemented with a static, hard-coded connection between the mobile device and the offloaded computation or data staging elements in the surrogate. However, this static link between mobile device and surrogate does not enable the flexibility that is implied by cyber-foraging as the opportunistic leveraging of resource-rich surrogates.

Motivation. Pre-provisioned surrogates have the advantage of shorter response time to offload requests from mobile devices because the offloaded computation or data staging elements already reside on the surrogate. In an operational setting in which surrogates support multiple clients, a surrogate should have minimal management capabilities that (1) help surrogate administrators to install capabilities (offloaded computation and data staging computing elements) and appropriate execution containers, and (2) maintain a list of these capabilities (similar to a service registry).

Description. Figure 3.11 shows the main components of the Pre-Provisioned Surrogate tactic. This tactic requires a *Surrogate Manager* that acts as a management component for the *Surrogate*. The *Surrogate Manager* is accessed by a system administrator from a *Local User Interface* running on the *Surrogate* or a *Remote User Interface* that resides on an external *Admin Client* (e.g., laptop, desktop, mobile device). When a system administrator uses the *Surrogate Manager* to install a new offload or data staging capability on the *Surrogate*, the capability is stored in a *Capabilities Repository* such as a file system or database. The *Capabilities Repository* contains the set of capabilities that are either started when the *Surrogate* is started, or started on demand when the*Offload Server* (from the Computation Offload tactic (Section 3.2.1)) or the *Data Staging Manager* (from the Data Staging tactics (Section 3.2.2)) receive a request from a mobile device. In the latter case, the *Capability Metadata* contains metadata that enables the setup of these capabilities on-demand, such as resource requirements, installation scripts, and configuration data. Installed

capabilities are then registered in a *Capability Registry* that is used by Surrogate Discovery tactics (Section 3.2.4) for advertising capabilities to mobile cyber-foraging clients.

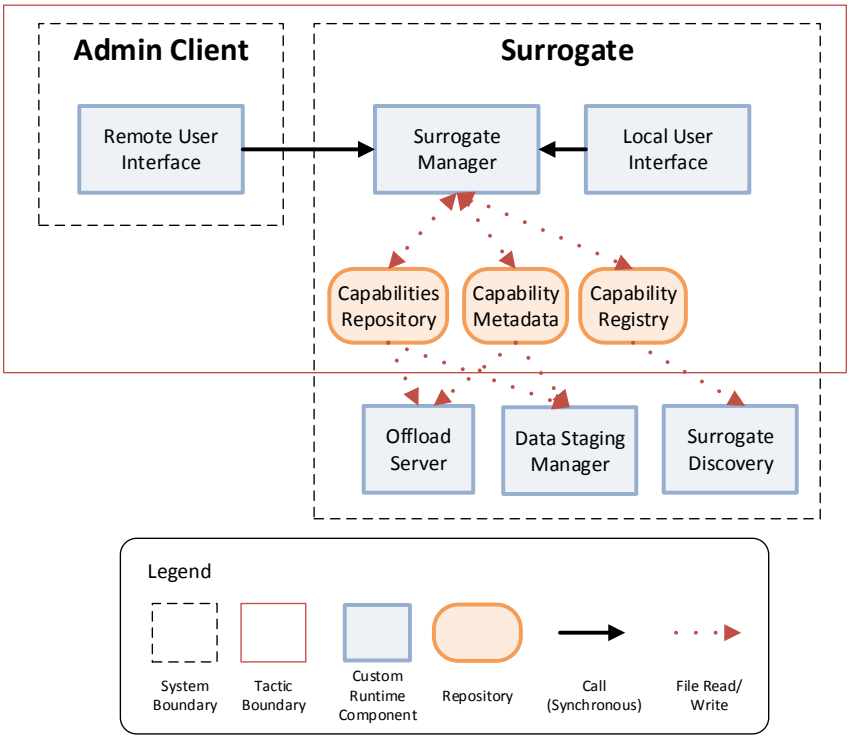

Figure 3.11: Pre-Provisioned Surrogate Tactic

Examples. This tactic is not present in any of the systems, but could be integrated into any of the cyber-foraging systems in the primary studies that assume that offloaded computation and/or data staging elements are already available on the surrogate at runtime. What would vary between pre-provisioned systems that implement this tactic is the form of the capabilities that are stored in the repository and capability metadata, which depend on the *What to Offload - Granularity* architecture decision from Figure 2.3.

- For systems that offload at the process level, such as CloneCloud [22] shown in Figure 3.4, the capabilities take the form of a container to

which the process and its state can migrate. For CloneCloud this is an Application VM.

- For systems that offload at the Method, Function or Operation level the capabilities take the form of the larger programming construct that these are a part of (i.e., class, module or program). As an example, if the MAUI system [26] would implement this tactic, the capabilities would take the form of .NET component classes that are stored in the Capabilities Repository and at runtime would be deployed inside a .NET CLR environment (i.e., execution container).

- For systems that offload at the Class, Module, Component, Task, Service, Application, Program, or Server level, the capabilities take this exact form because they are self contained. As an example, if the AIDE system [83] implemented this tactic the capabilities would take the form of Java classes that at runtime would be deployed inside a JVM.

In addition, something that would also vary across these systems is whether the offloaded computation is started once and always running, as in the mHealthMon system [2], or if it is started upon offload request as in the Grid-Enhanced Mobile Devices system [51]. In mHealthMon the services that correspond to offloaded computation are running and waiting for requests from mobile clients. Even though it is not explicitly stated in the study, starting up the system would involve starting all the services. If mHealthMon implemented this tactic, a startup process would start all the services in the Capability Repository. In Grid-Enhanced Mobile Devices, upon an offload request an object called a deputy object is created on the surrogate to manage all the mobile device's requests and then destroyed when the mobile device terminates the connection. This latter approach also promotes scalability and elasticity, as shown in the Just-In Time Containers tactic (Section 3.3.3.1).

3.2.3.2 Surrogate Provisioning from the Mobile Device

The Surrogate Provisioning from the Mobile Device tactic can be found in the computation offload systems shown in Table 2.6 for which *What to Offload - Payload* is Computation.

Motivation. In Pre-Provisioned Surrogates (Section 3.2.3.1) a mobile device can only execute applications that already exist on the surrogate. Provisioning the surrogate from the mobile device has the advantage of enabling the execution of a greater number of applications because surrogates are provisioned at runtime. The mobile device sends the offloaded computation to the surrogate

at runtime from the mobile devices that use them. The surrogate installs the computation inside an execution container and starts the application on behalf of the mobile device.

Description. Figure 3.12 shows the main elements of the Surrogate Provisioning from the Mobile Device tactic with numbers that indicate the sequence of operations. In this tactic each *Cyber-Foraging-Enabled Mobile App* has one or more files that correspond to *Offloaded Code for Cyber-Foraging-Enabled Mobile App*, such as a class, module or application. The *Cyber-Foraging-Enabled Mobile App* starts the offload process. The *Offload Client* sends the *Offloaded Code for Cyber-Foraging-Enabled Mobile App* to the *Offload Server* on the *Surrogate*. The *Offload Server* installs the offloaded code in an execution *Container* and notifies the mobile app that it is ready for execution. At this point the *Cyber-Foraging-Enabled Mobile App* starts the execution of the offloaded code.

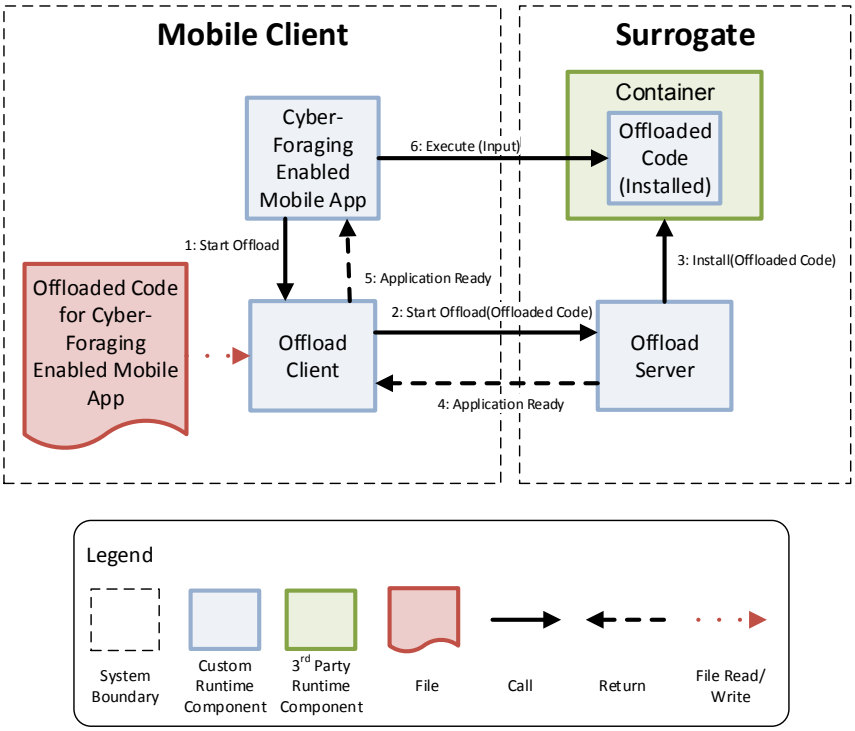

Figure 3.12: Surrogate Provisioning from the Mobile Device Tactic

Constraints. The tactic as presented requires a pre-established agreement between mobile devices and surrogates on the format of the offloaded code (e.g., Java class, Python script, Windows application). In addition, depending on the size of the offloaded code (i.e., payload), the tactic may require additional components on the mobile device and surrogate to manage and provide reliable communications during the transmission of the offloaded code.
Example. An example of how to apply the Surrogate Provisioning from the Mobile Device tactic is the VM-Based Cloudlets system [108]. In this system, an *Application Overlay* is created for each cyber-foraging-enabled mobile app by starting a *Base VM* (a minimally configured VM with a guest (OS) installed), installing the application in the *Base VM*, and then suspending the VM. The binary difference is calculated between the resulting VM image file and the *Base VM* and saved as an *Application Overlay*. As shown in Figure 3.13, at runtime the *Application Overlay* is sent by the *KCM Client* to the *KCM Server*. The *KCM Server* performs VM Synthesis by taking the same *Base VM* from which the *Application Overlay* was created and applying the overlay to it in order to recreate the VM with the installed application. The resulting VM is called a *Launch VM* and is started within a *VM Manager* (in this system it is VirtualBox[2]). Once the *Launch VM* is started and ready, the *KCM Client* is notified that the application is ready for execution. The user then interacts with the application via a *VNC Client*.

3.2.3.3 Surrogate Provisioning from the Cloud

The Surrogate Provisioning from the Cloud tactic can be found in the computation offload systems shown in Table 2.6 for which *What to Offload - Payload* is Source Location, which are the Roam [20] and the Elastic HTML5 [126] systems. For these two systems the payload is the URL of the location of the offloaded computation. It can also be found in the Collective Surrogates [48] and MAPCloud [103] systems for which *What to Offload - Payload* is Setup Instructions. In the first system the payload is a script that obtains the offloaded computation from the cloud; in the second system it is an application request that is modeled as a workflow of tasks to be located in the cloud.
Motivation. Provisioning surrogates from the mobile device has the advantage of enabling the execution of a greater number of applications (Section 3.2.3.2) compared to pre-provisioned surrogates (Section 3.2.3.1). However, the size of the computation that is sent to the surrogate at runtime can be significant. In the examples for the MAUI system [26], the size of the .NET

[2] `https://www.virtualbox.org/`

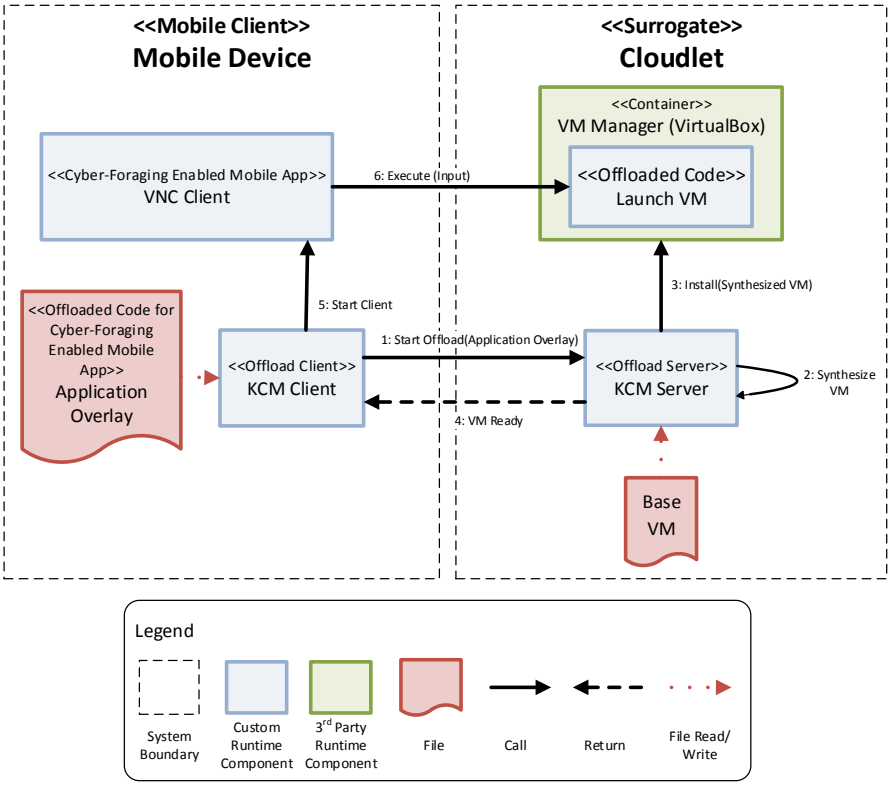

Figure 3.13: VM-Based Cloudlets as an Example of the Surrogate Provisioning from the Mobile Device Tactic

components transmitted at runtime is between 0.2 MB and 13.8 MB. In the examples for the VM-Based Cloudlets system [108], the size of an application overlay is between 63 MB and 196 MB. An alternative is to send the location of the computation in the form of a URL for the surrogate to download and install. The payload in this case is almost insignificant but the time to provision may be longer due to potentially higher and unpredictable latency between the cloud and the surrogate. However, the mobile device is not consuming battery due to high transmission costs. In addition, because the computation exists in a defined place in the cloud it is easier to update because it does not have to be sent to each mobile device after patches or upgrades.

Description. Figure 3.14 shows the main elements of the Surrogate Provisioning from the Cloud tactic with numbers that indicate the sequence of operations. In this tactic the Cyber-Foraging-Enabled Mobile App contains the URL that indicates the location from which the offloaded code has to be downloaded. The *Cyber-Foraging-Enabled Mobile App* starts the offload process by sending the URL to the *Offload Client*, which in turn sends it to the *Offload Server* on the *Surrogate*. The *Offload Server* downloads the offload code from an *Offload Code Repository* at the URL, installs it in an execution *Container* and notifies the mobile app that it is ready for execution. At this point the *Cyber-Foraging-Enabled Mobile App* starts the execution of the *Offloaded Code*.

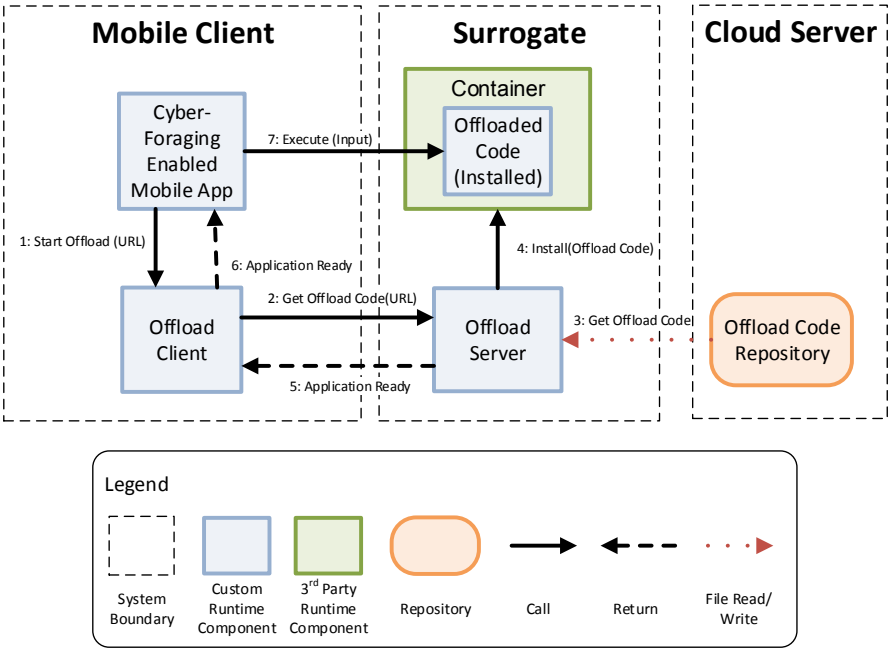

Figure 3.14: Surrogate Provisioning from the Cloud

Constraints. The tactic as presented requires connectivity between the surrogate and the cloud and potentially additional components on the surrogate and cloud server to manage and provide reliable communications during the transmission of the offloaded code. The computation has to exist at the indi-

cated location. In addition, it requires a pre-established agreement between surrogates and the cloud servers on the format of the offloaded code (e.g., Java class, Python script, Windows application).

Example. An example of how to apply the Surrogate Provisioning from the Cloud tactic is the Collective Surrogates system [48]. As shown in Figure 3.15, at runtime once a *Participating Node* is assigned to an offload operation, the *Offload Client* sends a shell script to a *Daemon* running on the *Participating Node* which executes the script on behalf of the client. The script downloads the application that corresponds to the offloaded code from an *Application Repository* on an *Internet Server*, installs the application and starts it. Once the *Application* is started and ready, the *Offload Client* is notified that the application is ready for execution. The user then interacts with the application via a *Client Interface*.

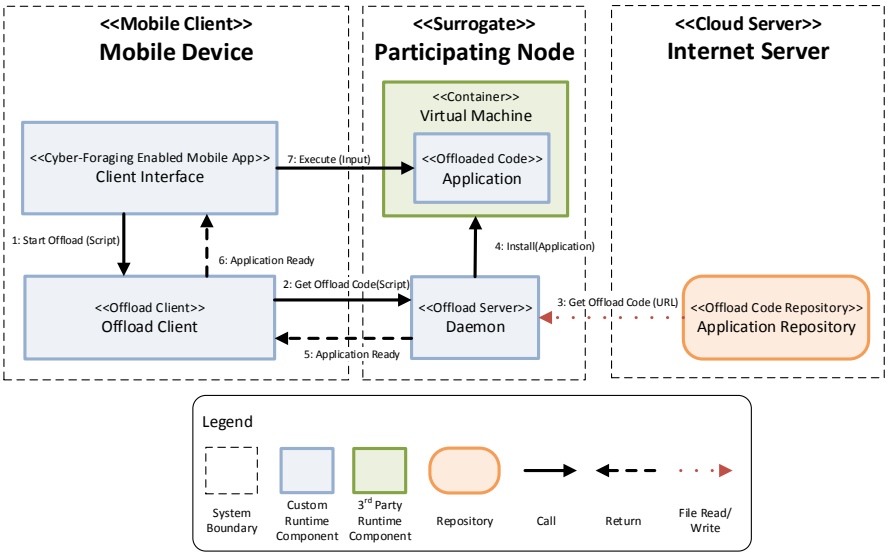

Figure 3.15: Collective Surrogates as an Example of the Surrogate Provisioning from the Cloud Tactic

3.2.4 Surrogate Discovery

In order to leverage cyber-foraging, mobile devices need to be able to locate available surrogates on which to offload computation or stage data. A sce-

nario for surrogate discovery is as follows: a mobile device needs to execute a computation-intensive task and has already decided that it will offload the task to a surrogate. The mobile device is able to locate all nearby surrogates and selects the surrogate that is the best match for the offloaded task.

The Surrogate Discovery tactics are a pre-requisite for Data Staging (Section 3.2.2) and Computation Offload (Section 3.2.1) tactics. The surrogate discovery protocol becomes the initial part of the offload process. Surrogate Discovery tactics need to be matched with a Surrogate Provisioning tactic (Section 3.2.3) that prepares the surrogate for cyber-foraging.

3.2.4.1 Local Surrogate Directory

The Local Surrogate Directory tactic can be found in six systems that maintain a list of potential surrogates on which to offload computation or stage data: Roam [20], Spectra [41], Cuckoo [62], SPADE [112], Offloading Toolkit and Service [121], and Heterogeneous Auto-Offloading Framework for Mobile Web Browsers [128].

Motivation. For mobile devices to leverage nearby surrogates they need to know where the surrogates are located; that is, they need to know their network address (i.e., surrogate IP address or URL). A simple solution is for mobile devices to maintain a list of potential surrogates with their network addresses or URLs, in addition to any information that can help the mobile device to select the best offload target in case more than one is available. The list can be static, or updated based on network conditions or offload execution data. An advantage of a local list is that it will potentially include only surrogates that are trusted by the mobile device.

Description. The Local Surrogate Directory Tactic has two parts. One part involves the *Surrogate Directory UI* which populates and maintains the *Surrogate Directory*. The other part involves the components that interact during the offload process as shown in Figure 3.16 with numbers that indicate the sequence of operations. At runtime, the *Cyber-Foraging Mobile App* calls the *Offload Client* to start the offload process. The *Offload Client* obtains that list of potential surrogates from the *Surrogate Directory* and pings each *Surrogate* to see if it is available for offload. The *Offload Server* of each available *Surrogate* responds to the *Offload Client* with any *Surrogate Metadata* required by the discovery protocol, such as current load or available capabilities. Based on this information and any network information available, the *Offload Client* selects the best surrogate for offload and starts the actual offload operation with the selected *Surrogate*. Optionally, the *Offload Client* may update the *Surrogate Directory* based on the availability and performance of the selected

surrogate.

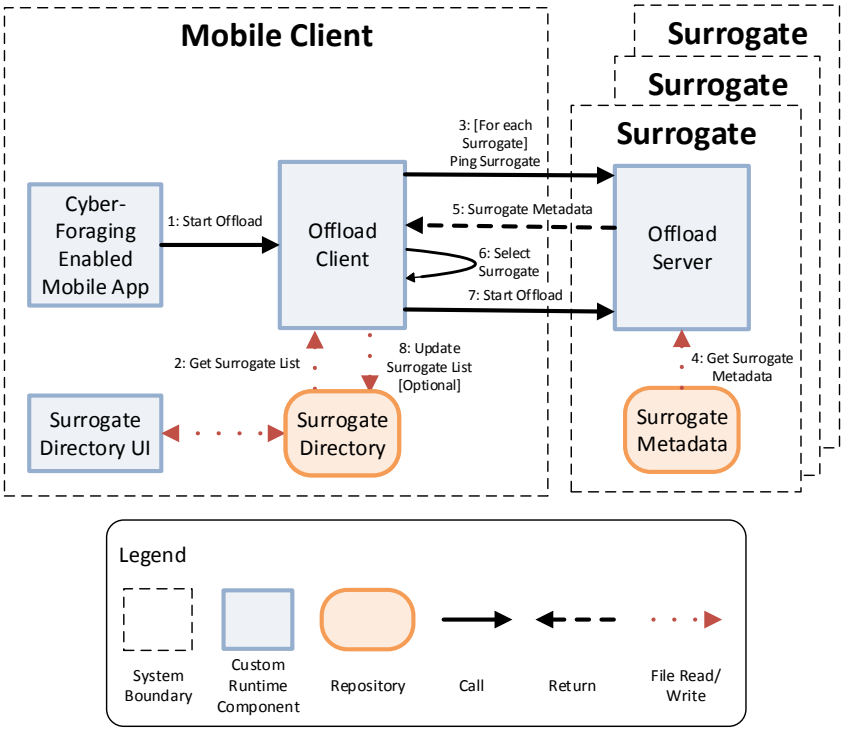

Figure 3.16: Local Surrogate Directory

Constraints. The tactic as presented places the responsibility of surrogate identification on the mobile device user. If surrogate metadata changes or new surrogates are made available, a cyber-foraging system will not have an automated way of updating the surrogate directory.
Examples. The six systems that implement the Local Surrogate Directory tactic maintain a list of potential surrogates for offload. What varies between systems is how the list is populated and whether or not the list is updated based on network conditions or offload execution data.

- Roam [20] maintains a list of servers that can accept offloadable components along with their characteristics. These characteristics are used at runtime to determine an appropriate offload target.

- Spectra [41] keeps a list of surrogates that are willing to host computation in a configuration file. As the system executes, the status of each surrogate is updated (e.g., availability, CPU load, file cache state).
- Cuckoo [62] has a component called a *Resource Manager* that maintains a list of surrogates. If the surrogate has a visual display, upon loading it shows a QR code[3] that is read by the mobile device and then added to the list of resources (surrogates) it can use for offload. If it does not have a visual display, the resource description file for the surrogate has to be copied to the mobile device so that it can be added to the list.
- SPADE [112] users have to associate remote computers called *Cycle Providers* to specific tasks that are part of a job. At runtime, the mobile device uses this list to locate cycle providers based on each of the tasks that it needs to execute. An interesting aspect of this system is that surrogates have functionality to discover other surrogates on the same network and can provide this list back to the mobile device. However, the mobile device does not have capabilities to discover surrogates on its own. Details of this system are shown as an example in Figure 3.17. A single *User Interface* acts as the UI for maintaining the *Cycle Provider List* and for starting an offload job. The *Job Manager* selects a *Cycle Provider* for each task and starts the offload for each in a separate process so that tasks can execute in parallel.
- Offloading Toolkit and Service [121] maintains a list of surrogates (service providers) that are queried at runtime for desired capabilities. Each surrogate maintains its own service registry.
- Heterogeneous Auto-Offloading Framework for Mobile Web Browsers [128] queries all potential surrogates on its list for matching required capabilities. Each matching surrogate sends back quality information (e.g., server capability and network bandwidth) and the client decides whether to offload the computation to a matching surrogate or execute locally.

3.2.4.2 Cloud Surrogate Directory

The Cloud Surrogate Directory tactic can be found in 12 systems in which the mobile device contacts a cloud server that maintains a list of potential

[3] A QR code, or Quick Response Code, is a machine-readable code consisting of an array of black and white squares that typically contains URLs or other information that can be read by the camera on a smartphone (`http://www.qrcode.com/en/`).

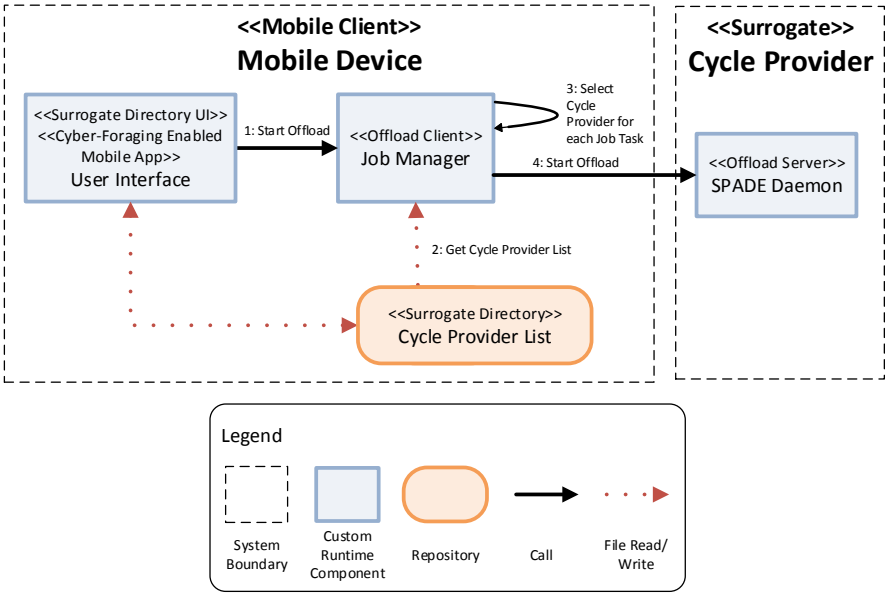

Figure 3.17: SPADE as an Example of the Local Surrogate Directory Tactic

surrogates on which to offload computation or stage data: Mobile Agents [5], HPC-as-a-Service [30], Collective Surrogates [48], Grid-Enhanced Mobile Devices [51], ThinAV [59], MCo [74], Resource Furnishing System [92], Cloud Personal Assistant (CPA) [93], MAPCloud [103], Large-Scale Mobile Crowdsensing [119], Mobile Data Stream Application Framework [122], and Weblets [127].

Motivation. In the Local Surrogate Directory tactic (Section 3.2.4.1) the mobile device is responsible for populating and maintaining the list of surrogates on which it can offload computation. This is a rather static solution because as more surrogates become available in the environment there is no automated way of discovering these new surrogates or updating their metadata as changes occur. Maintaining the surrogate directory in the cloud has the advantage of a centralized location for surrogate registration. All surrogate metadata is populated and updated in this central repository. All the mobile device needs to know is the network address of the cloud server that manages the surrogate directory. In addition, optimal surrogate selection algorithms can run in the cloud, which is an additional offload operation that can lead to battery savings

on the mobile device. Regarding trust, in this tactic the mobile device only needs to trust the cloud surrogate directory server assuming that the directory only contains trusted surrogates (Section 3.3.4.1).

Description. In the Cloud Surrogate Directory Tactic the *Surrogate Directory* is located in a *Cloud Server*. Figure 3.18 shows the main elements of the tactic with numbers that indicate the sequence of operations. The *Cyber-Foraging-Enabled Mobile App* starts the offload process by querying the *Surrogate Directory* via the *Surrogate Directory Interface*. This is the same interface that would be used by any program that populates and maintains the *Surrogate Directory* or by *Surrogates* that provide live data. The *Surrogate Directory Interface* selects the optimal surrogate from the directory based on data such as mobile device characteristics, type of offload request, surrogate availability, surrogate load, or any other data that is available in the directory or was provided by the mobile device as query parameters. The *Surrogate Directory Interface* then sends the *Offload Client* the data for the selected surrogate which includes the surrogate address. The *Offload Client* contacts the *Offload Server* of the selected *Surrogate* to continue the offload process.

Constraints. The tactic as presented requires the mobile device to know the address of the cloud server that holds the surrogate directory. The cloud server can become a single-point-of-failure if it becomes unavailable to mobile devices. In the cases that the cloud server acts as an intermediary it also becomes a potential bottleneck. Cloud servers that perform service discovery instead of simply maintaining a surrogate directory suffer from the traditional challenges of service discovery in service-oriented computing [95].

Examples. The 12 systems that implement the Cloud Surrogate Directory tactic maintain a list of potential surrogates on a centralized cloud server. What varies between systems is (1) the parameters that are used for surrogate selection, (2) whether the surrogate selection algorithm runs on the cloud server or the mobile device, (3) whether the surrogate directory maintains a list of surrogates or a list of services that are hosted on each surrogate, and (4) whether the cloud server returns a surrogate address or forwards the offload request to the surrogate therefore acting as an intermediary (a variation of this tactic).

- Mobile Agents [5]: As shown in Figure 3.19, the *Execution Manager* on the mobile device contacts a *Cloud Directory Service* to get a list of available surrogates and selects the one with the highest communication link speed with the mobile device as well as the highest computing power.
- HPC-as-a-Service [30]: The mobile device queries a centralized repository

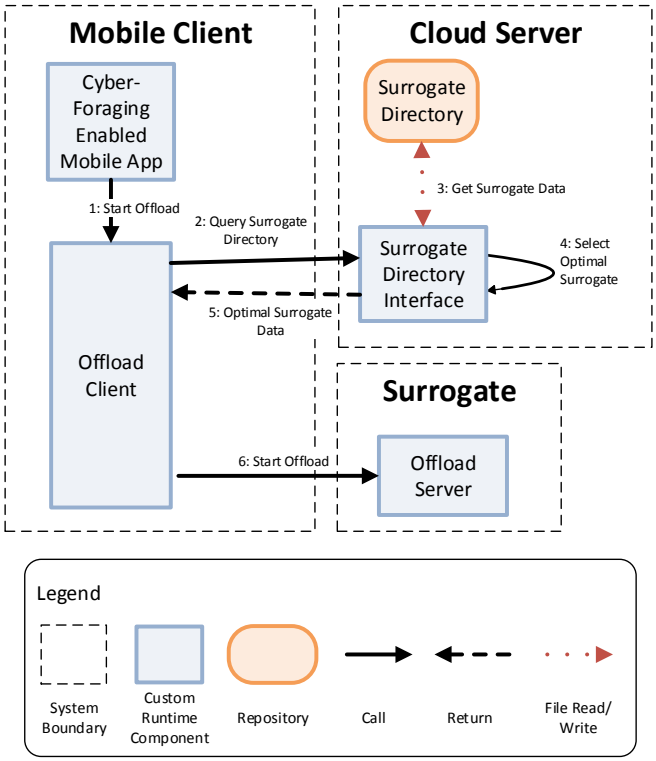

Figure 3.18: Cloud Surrogate Directory

of HPC (high-performance computing) services to locate a service with given characteristics.

- Collective Surrogates [48]: The mobile device contacts a *Collective Manager* that manages a set of surrogates (participating nodes) and uses profile and historic information to determine the specific surrogate on which the computation will be offloaded.
- Grid-Enhanced Mobile Devices [51]: Mobile devices contact the *Grid Gateway* which locates Grid services available on surrogates and then forwards the offload request, acting as as intermediary.
- ThinAV [59]: The cloud server (ThinAV Server) submits received off-

load requests to surrogates and returns results to mobile clients when available. The *ThinAV Server* acts as an intermediary.

- MCo [74]: Upon receipt of computation offloading request from a mobile device, the cloud server (Master Node) searches its list of surrogates (Worker Nodes) on which computation can be offloaded. Once a *Worker Node* is selected the offload request is forwarded. The *Master Node* acts as an intermediary.

- Resource Furnishing System [92]: A *Dispatching Surrogate* maintains the software list of known surrogates (application servers), and selects an application server based on the contents of the request packet and application server load.

- Cloud Personal Assistant (CPA) [93]: CPA receives a set of tasks to execute from a mobile device, discovers the necessary cloud services, invokes them and then delivers the results back to the mobile device, acting as an intermediary.

- MAPCloud [103]: For each offload request (modeled as a workflow of tasks) from a mobile device, the *Broker* consults the registry of available surrogates and services and returns the addresses of services that can execute each task.

- Large-Scale Mobile Crowdsensing [119]: A cloud server (Application Server) consults a global registry for a list of surrogates (Cloudlets) that are located in a certain area.

- Mobile Data Stream Application Framework [122]: Mobile devices send offload requests to a cloud server (Resource Manager) which then assigns a surrogate (Application Master) to handle the request.

- Weblets [127]: A *Cloud Elasticity Service (CES)* allocates surrogates to offload requests based on usage information (e.g., compute power, bandwidth and storage).

Variation: Intermediary Cloud Surrogate Directory. The tactic as described returns the address of the selected surrogate to the mobile device, which then contacts the surrogate directly. In Grid-Enhanced Mobile Devices [51], ThinAV [59], MCo [74], Cloud Personal Assistant (CPA) [93], and Large-Scale Mobile Crowdsensing [119] the *Cloud Server* does not return the surrogate address to the mobile device, but rather forwards the offload request

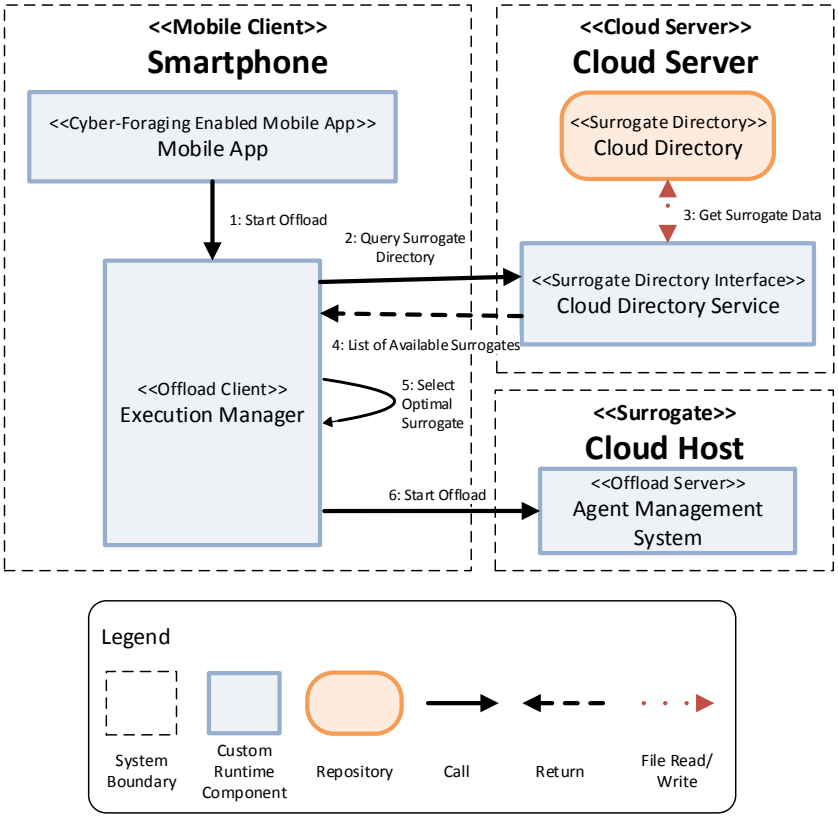

Figure 3.19: Mobile Agents as an Example of the Cloud Surrogate Directory Tactic

to the selected *Surrogate* and then returns the results to the mobile device. In this variation the *Cloud Server* acts as an intermediary between the *Mobile Device* and the *Surrogate*.

3.2.4.3 Surrogate Broadcast

The Surrogate Broadcast tactic can be found in five systems in which surrogates broadcast or advertise their presence to mobile devices: Scavenger [67], Real Options Analysis [35], Application Virtualization on Cloudlets [84],

VM-Based Cloudlets [108], and Slingshot [114].

Motivation. The Local Surrogate Directory (Section 3.2.4.1) and Cloud Surrogate Directory (Section 3.2.4.2) tactics require a directory of potential surrogates to be maintained either on the mobile device or on a cloud server, respectively. Having surrogates broadcast their availability and metadata to mobile devices removes the burden of having to maintain surrogate directories up to date. 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.

Description. As shown in Figure 3.20, in the Surrogate Broadcast tactic all *Surrogates* broadcast selected metadata using a *Broadcast Component*. The numbers in the figure indicate the sequence of operations, starting with the broadcast operation as 0 to mean that it occurs in advance of the offload request. The *Cyber-Foraging-Enabled Mobile App* initiates the offload request. The *Offload Client* finds available surrogates by analyzing broadcast information which will include at least the surrogate address. The *Offload Client* then selects the optimal surrogate and starts the offload process by contacting the *Offload Server* of the selected surrogate. In addition to basic surrogate metadata such as surrogate address, the surrogate can also broadcast data retrieved from a *Capability Metadata* repository on the *Surrogate* as described in the Pre-Provisioning tactic (Section 3.2.3.1).

Constraints. The tactic as described requires an agreement between mobile devices and surrogates on the broadcast protocol. Regarding trust, mobile devices will require additional components to determine whether broadcast information is coming from a valid, trusted surrogate (Section 3.3.4.1).

Examples. The surrogates in the five systems that implement the Surrogate Broadcast tactic broadcast their availability and selected metadata to mobile devices for discovery. What varies between systems is the broadcast mechanism and the information or metadata that they broadcast.

- Scavenger [67]: Surrogates periodically broadcast their service descriptions using UDP broadcast.[4] As shown in Figure 3.21, a *Presence Daemon* running on each *Surrogate* periodically packs all its service descriptions into a single UDP packet and broadcasts it onto the local subnet. An *Application* running on a mobile device uses the *Scavenger Library* to find available surrogates, select the optimal surrogate on which to offload, and finally contact the *Scavenger Front-End* of the selected surrogate.

[4] UDP stands for User Datagram Protocol and is one of the core protocols of the IP suite. UDP broadcast is the broadcasting of UDP packets to an entire subnet.

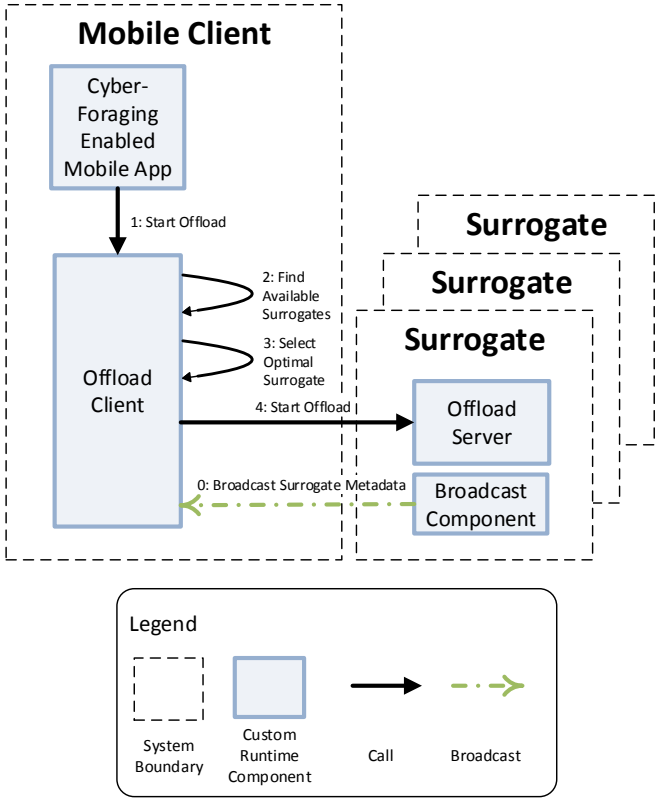

Figure 3.20: Surrogate Broadcast

- Real Options Analysis [35]: As surrogates come online, they broadcast their availability and address over a broadcast channel.
- Application Virtualization on Cloudlets [84] and VM-Based Cloudlets [108]: Surrogate information that includes surrogate address is broadcast using an implementation of Zeroconf.[5]

[5]Zerconf stands for Zero Configuration Networking and is a set technologies that enables automated network configuration of devices and services without the use of central services such as DNS or DHCP (`www.zeroconf.org`).

- Slingshot [114]: This system uses UPnP[6] to discover new surrogates in its surrounding network environment.

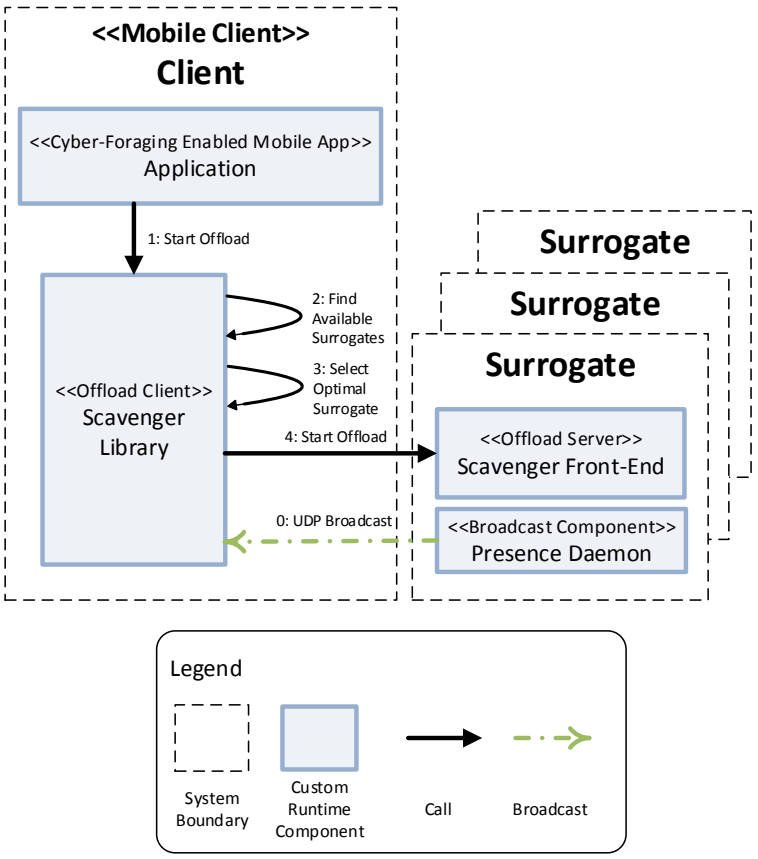

Figure 3.21: Scavenger as an Example of the Surrogate Broadcast Tactic

[6]UPnP stands for Universal Plug and Play and is a set of networking protocols that enable networked devices to seamlessly discover each other's presence on the network and establish functional network services (`www.upnp.org`).

3.3 Non-Functional Architectural Tactics for Cyber-Foraging

The non-functional architectural tactics described in this section are used in combination with the functional architectural tactics described in Section 3.2 to meet additional requirements placed on cyber-foraging systems.

3.3.1 Resource Optimization

A scenario for Resource Optimization is the following: A mobile app is enabled for cyber-foraging. Upon request for execution of computation that has been targeted for offload, the mobile app first checks if it is better from a performance and latency perspective to execute the computation locally or remotely. Given that the the network conditions between the mobile device and the surrogate are not ideal, the computation is executed locally instead of offloaded to the surrogate.

3.3.1.1 Runtime Partitioning

The Runtime Partitioning tactic can be found in the computation offload systems shown in Table 2.6 for which *When to Offload* is Runtime Decision.
Motivation. In general, offloading is beneficial when large amounts of computation are needed with relatively small amounts of communication [70]. Runtime Partitioning enables mobile devices to make runtime decisions regarding the benefits of offloading. Computation is offloaded only if remote execution is better than local execution according to a defined optimization function (often called a utility function). Local execution cost typically takes into consideration the energy consumed by local execution as well as the local execution time. Remote execution cost typically considers the energy consumed by communication based on payload size and network conditions, the communication time based on payload size and network conditions, and remote execution time. If local execution cost is lower than remote execution cost then the computation is executed locally; if not, it is executed remotely (i.e., offloaded).
Description. Figure 3.22 shows the main components of the Runtime Partitioning tactic with numbers to indicate the sequence of operations. In addition to the components required by the Computation Offload tactic, the Runtime Partitioning tactic requires an *Offload Decision Engine* component that compares predicted local execution cost against predicted remote execution cost. The *Offload Decision Engine* uses *App Metadata* such as required compute

cycles, payload size based on input and output parameters, and required energy for execution and communication. Even though the *App Metadata* is depicted in Figure 3.22 as an external file, this data can also reside within the code as annotations. Upon a request for execution of a computational element that is marked for offload, the *Cyber-Foraging Enabled Mobile App* invokes the *Offload Decision Engine*, passing it the necessary metadata for the *Offloadable Element*. In addition, although optional, the *Offload Decision Engine* can also make use of *Environment Monitors* to obtain runtime environment data such as network conditions or load of the mobile device and surrogate if these are required by the defined optimization function. It can also make use of *Cost Models* (e.g., an energy model for the mobile device) as input to the optimization function. Based on the results of the optimization function, the *Cyber-Foraging-Enabled Mobile App* invokes the local copy of the *Offloadable Element* or invokes the *Offload Client* in order to invoke the remote copy of the *Offloadable Element* running on the *Surrogate*.

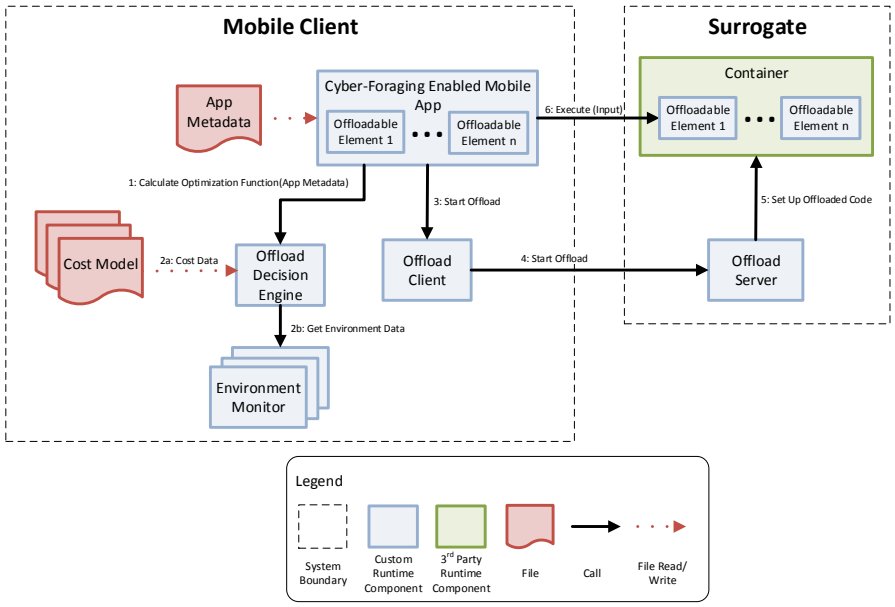

Figure 3.22: Runtime Partitioning Tactic

Constraints. The Runtime Partitioning tactic assumes that there is equivalent code for the offloaded computation on both the mobile device and the

surrogate. This aspect limits the direct reusability of legacy code because a version would have to be written for the mobile device or surrogate depending on the original platform of the legacy code. In addition, the optimization function should not be a computation-intensive task because it would then cancel the benefits of cyber-foraging. Finally, data collection of app metadata to be used as optimization function parameters has to be gathered in advance using techniques such as static profiling.

Example. An example of how to apply the Runtime Partitioning tactic is the MACS system [65], as shown in Figure 3.23. In MACS, *Cyber-Foraging Enabled Mobile Apps* contain offloadable elements defined as *Services*. Each service has *Service Metadata* related to memory size, code size, and input/output parameter size. When the mobile app is going to execute a service, the *Performance and Context Monitor* is invoked to determine the feasibility of remote execution as well as to compare the cost of local execution of the service against the cost of remote execution. The *Performance and Context Monitor* uses a *Mobile Device Monitor* implemented as calls to the Android API to obtain available memory information, CPU load and remaining battery. It also uses a *Network Monitor* to obtain connectivity and bandwidth information. In addition, based on a pre-built *Energy Model* it calculates the energy cost of local vs. remote execution using the service metadata. If the decision is to offload, the *Offload Manager* and *Remote Execution Manager* coordinate to set up the offloaded service for remote execution.

Dependencies. The Runtime Partitioning tactic requires the Computation Offload tactic (Section 3.2.1) as the infrastructure for computation offload.

Variation: User-Guided Runtime Partitioning. The tactic as described assumes a static optimization function. However, in some systems what to optimize is determined based on user preferences or input. In the PowerSense system [82] the user can select a *Time Saver* option to minimize processing time or an *Energy Saver* option to minimize energy consumption. The ThinkAir system [64] offers four optimization options (profiles) to users: execution time only; energy consumption only; execution time and energy consumption; execution time, energy consumption and cost of cloud services. These systems have a user interface on the mobile device to set these preferences.

3.3.1.2 Runtime Profiling

The Runtime Profiling tactic can be found in ten systems: MAUI [26], Real Options Analysis[35], Single-Server Offloading [56], ThinkAir [64], AMCO [72], SmartVC [100], Odessa [101], IC-Cloud [111], AIOLOS [117], and Mobile Data Stream Application Framework [122].

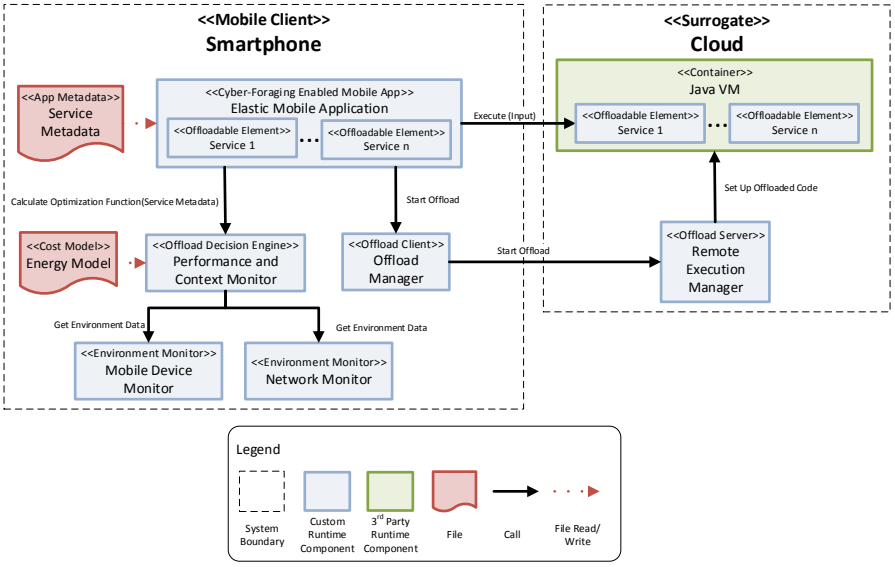

Figure 3.23: MACS as an Example of the Runtime Partitioning Tactic

Motivation. Systems that implement the Runtime Partitioning tactic (Section 3.3.1.1) require developer input or static profiling to obtain the values or models that are used in the calculation of the optimization function that determines whether code should run locally or remotely. However, models tend to be inaccurate because (1) applications are not deterministic, (2) smartphones scale the CPU's voltage dynamically to save energy (i.e., dynamic voltage scaling), (3) energy models highly depend on hardware configuration, usage, and even the battery model of a mobile device, and (4) network quality is highly variable and often unpredictable [29]. To account for this variability and take into consideration current conditions, once the offload operation ends, or periodically, the system updates the profiling data and models that are used by the optimization functions.

Description. Figure 3.24 shows the main components of the Runtime Profiling tactic. The difference between the Runtime Profiling tactic and the Runtime Partitioning tactic (Section 3.3.1.1) is the data that is used in the offload decision and what happens after the offloading process ends. The *Cyber-Foraging Enabled Mobile App* invokes the *Offload Decision Engine*, passing it the necessary metadata for the *Offloadable Element*. In addition to run-

time data obtained from *Environment Monitors* and *Cost Models*, the *Offload Decision Engine* uses *Historical Execution Data* as input to the optimization function. Large differences between estimated and historic cost data might trigger the *Offload Decision Engine* to adjust the *Cost Models*. Based on the results of the optimization function, the *Cyber-Foraging-Enabled Mobile App* invokes the local copy of the *Offloadable Element* or invokes the *Offload Client* in order to invoke the remote copy of the *Offloadable Element* running on the *Surrogate*. After the offload process is completed, the *Offload Client* saves current execution data for the offloadable element such as timestamp, input parameters, energy consumption, network quality, and execution time in the *Historical Execution Data* repository. In addition, although optional, the *Environment Monitors* may store environment data periodically in the *Historical Execution Data* repository.

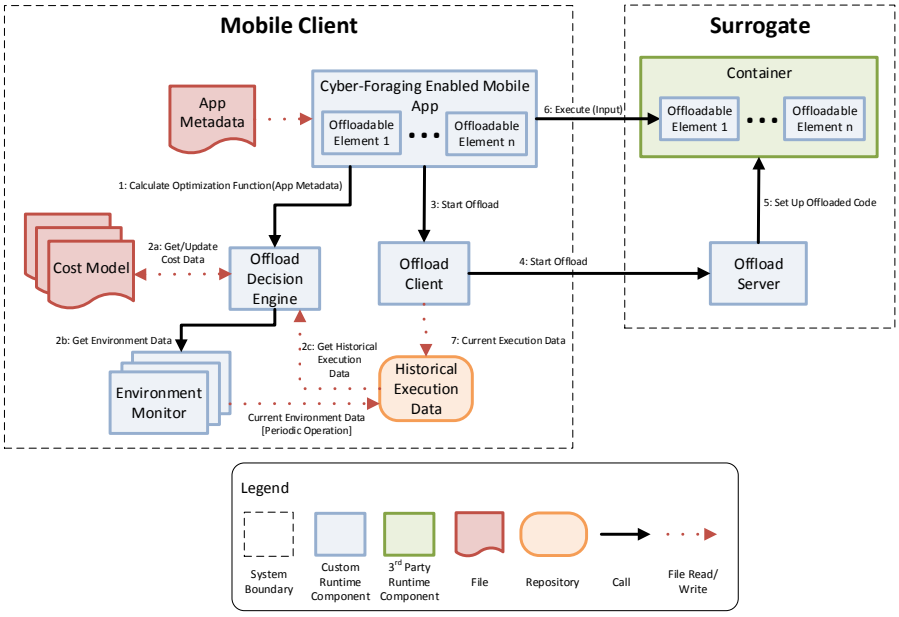

Figure 3.24: Runtime Profiling Tactic

Constraints. As in the Runtime Partitioning tactic (Section 3.3.1.1), the Runtime Profiling tactic assumes that there is equivalent code for the offloaded computation on both the mobile device and the surrogate. In addition, the cost of profiling is not negligible and can impact overall application performance

[26]. System designers need to consider the type and frequency of data to capture at runtime.

Examples. The ten systems that implement the Runtime Profiling tactic update the data that is used by the optimization function based on current execution data and environmental conditions. What varies between systems is the type of data that is captured and and the frequency of data capture.

- MAUI [26]: As shown in Figure 3.25, the *Solver+Profiler* uses data from the annotated method (inputs, outputs and CPU cycles), the *Device Energy Model*, network data obtained via a *Network Monitor*, and *Past Program Execution and Network Data* to compute an energy-efficient program partition. Once an offloaded method terminates, the *Client Proxy* updates the *Past Program Execution and Network Data* to better predict whether future invocations of the method should be offloaded.

- Real Options Analysis[35]: The system maintains a list of accessible servers and estimates the network delay to each of them using the default routing. Once offload completes, the network traffic model is updated.

- Single-Server Offloading [56]: Remote execution time is calculated for the first execution as communication time plus remote computation time. The latter is sent back from the surrogate as part of the results. From the second execution on, the model predicts local and remote execution time and offloads only if remote execution time is less than local execution time. The system updates the execution time parameters from actual computation results only if the difference between predicted and actual execution times (local and remote) is greater than an established threshold.

- ThinkAir [64]: When a method is encountered for the first time,the decision to offload is based only on environmental parameters such as network quality. From that point on, the profilers start collecting execution and energy consumption data for that method. If the method is invoked again, the decision to offload is based on the method's past execution times and energy consumed.

- AMCO [72]: Based on a feedback-loop mechanism, energy consumption data is updated after the execution of code portions marked as "energy hotspots" and used in the calculation of future energy consumption which drives offload decisions.

- SmartVC [100]: The system records the execution time and power consumption for each method as historical data to better inform future offloading decisions.
- Odessa [101]: The system's decision engine uses the recent history of network measurements to determine if offloading or increasing the level of parallelism will improve performance.
- IC-Cloud [111]: The system uses signal strength and historical information of network states to obtain a coarse-grained estimation of network access quality that influences the offload decision.
- AIOLOS [117]: The system updates the surrogate and network state data used by the estimation model after every offload operation.
- Mobile Data Stream Application Framework [122]: The profiler on the mobile device measures the device's characteristics at startup and continuously monitors its CPU workload and wireless network bandwidth. If any of the parameters varies by a value exceeding an established threshold, a new partitioning is generated for the application.

Dependencies. The Runtime Profiling tactic requires the Runtime Partitioning tactic (Section 3.3.1.1) to enable the system to make a runtime decision on whether or not to offload computation. It also requires the Computation Offload tactic (Section 3.2.1) to establish the infrastructure for computation offload.

3.3.1.3 Resource-Adapted Computation

The Resource-Adapted Computation tactic can be found in the Cuckoo system [62]. Cuckoo has elements that enable it to use different versions of offloadable elements to match the resource characteristics of mobile devices and surrogates, depending on whether code executes locally or remotely.
Motivation. In the Runtime Partitioning tactic (Section 3.3.1.1) a decision is made at runtime to execute code locally or remotely depending on an optimization function. In this tactic the local and remote code are identical. Even though this makes development and versioning easier, computation ends up being limited to what can execute on the mobile device, which will always lag behind static elements such as surrogates in terms of compute resources (power, CPU, memory, storage) [107]. Resource-Adapted Computation enables cyber-foraging systems to fully take advantage of the computing power of surrogates by adapting the computation to the resource on which it will be

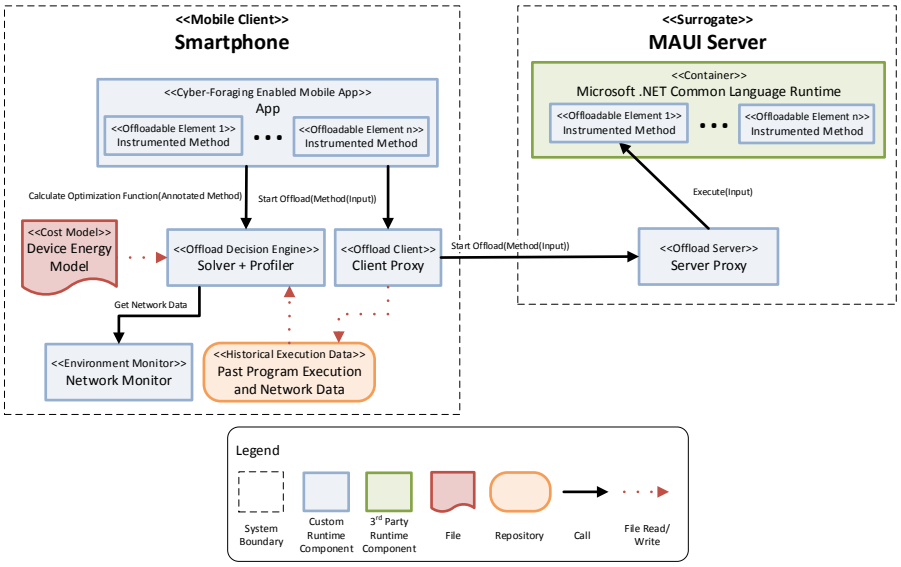

Figure 3.25: MAUI as an Example of the Runtime Profiling Tactic

executing. In an image processing scenario, the object recognition algorithm that runs on the surrogate can be much more computation-intensive than the one that runs on the mobile device and can therefore deliver a much more precise result.

Description. Figure 3.26 shows a simplified representation of the Runtime Partitioning tactic (Section 3.3.1.1) with additional elements that describe the Resource-Adapted Computation tactic. At runtime, the *Offload Decision Engine* calculates the optimization function for the *Offloadable Element*. If the decision is to execute locally, the *Cyber-Foraging Enabled Mobile App* executes the *Offloadable Element (Mobile Version)* that is adapted to the resource characteristics of the mobile device. However, if the decision is to execute remotely, the *Offloadable Element (Surrogate Version)* is executed to take advantage of the more powerful resources of the *Surrogate*.

Constraints. The Resource-Adapted Computation tactic requires developing, profiling and maintaining different versions of offloadable elements.

Example. Cuckoo [62] is an example of a system that implements the Resource-Adapted Computation tactic. The Cuckoo Framework generates an implementation of the same interface for a local and a remote service. Initially, the

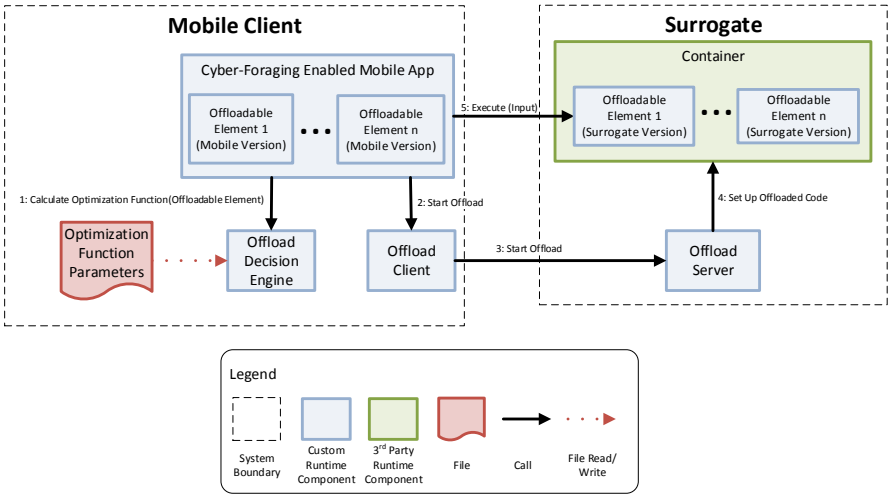

Figure 3.26: Resource-Adapted Computation

remote implementation will contain dummy method implementations, which the developer has to replace with real method implementations that can be executed at the remote location.The real methods can be identical to the local service implementation, but may also be completely different, because the remote implementation can run a different algorithm, use a different library, or take advantage of parallelization on the more powerful surrogate. Figure 3.27 shows the Cuckoo system at runtime with numbers to indicate the sequence of operations. The *Cuckoo Framework* intercepts all service calls. It then uses the *Cuckoo Resource Manager* to decide whether to execute the local or the remote implementation of the service. In the current implementation it will execute the remote implementation if a surrogate is available (details of how it locates surrogates are in Section 3.2.4.1. If a surrogate (*Cuckoo Server*) is not available, the *Local Service Implementation* is executed. If a surrogate is available, it uses the *Ibis Middleware* to invoke the *Remote Service Implementation*.

Dependencies. The Resource-Adapted Computation tactic requires the Runtime Partitioning tactic (Section 3.3.1.1) to enable the system to make a runtime decision on whether or not to offload computation. It also requires the Computation Offload tactic (Section 3.2.1) to establish the infrastructure for computation offload.

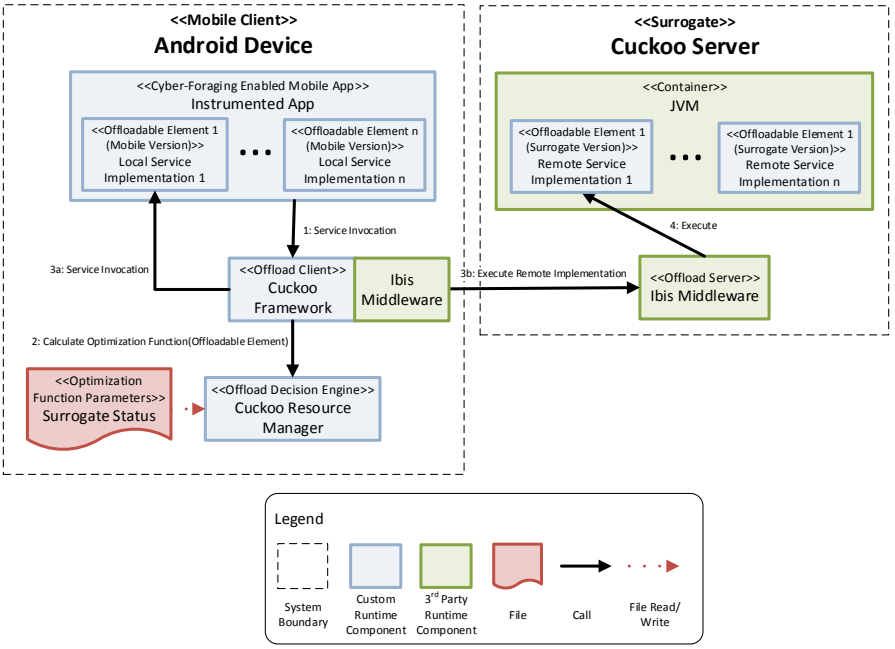

Figure 3.27: Cuckoo as an Example of the Resource-Adapted Computation Tactic

Variation: Resource-Adapted Input. A variation of this tactic is for the *Offloadable Element (Mobile Version)* and the *Offloadable Element (Surrogate Version)* to be identical, but what varies is the input parameters. The enabler is that different input parameters will lead to different resource consumption. PowerSense [82] is an image processing system for dengue detection that implements this variation of the tactic. PowerSense uses the same algorithm (implementation) locally and remotely for image processing, but uses images of lower resolution if processed locally and higher resolution if processed remotely because processing these higher quality images requires greater computing power.

3.3.2 Fault Tolerance

A scenario for Fault Tolerance is the following: A mobile app is enabled for cyber-foraging and is leveraging a surrogate for computation offload. During

the execution of the remote computation the mobile device loses connectivity to the surrogate. The mobile device detects the situation and executes the local copy of the computation instead with minimal effect on user experience.

3.3.2.1 Local Fallback

The Local Fallback tactic can be found in the MAUI [26] and ThinkAir [64] systems. These systems have elements that enable them to use the local copy of the offloadable computation in case the connectivity to the surrogate is lost.
Motivation. Due to movement of a mobile device to an area with no connectivity to the surrogate, problems with network quality, or service disruption, the mobile device may lose connectivity to the surrogate during the computation offload or data staging process. The *Local Fallback* tactic enables the cyber-foraging enabled mobile app to detect loss of connectivity and revert to local execution of the offloaded element.
Description. Figure 3.28 is an extension of the Computation Offload tactic (Section 3.2.1) marked with numbers that indicate the sequence of operations that trigger the local fallback. The *Cyber-Foraging Enabled Mobile App* starts the computation offload process by contacting the *Offload Client* which in turn contacts the *Offload Server* that sets up the *Offloaded Code* on the *Surrogate*. Upon completion of the setup process the *Cyber-Foraging Enabled Mobile App* starts execution of the *Offloaded Code* on the *Surrogate*. During execution the *Cyber-Foraging Enabled Mobile App* detects a timeout in the communication with the *Surrogate* (or a network monitor detects loss of connectivity). At this point the *Cyber-Foraging Enabled Mobile App* executes the local version of the offloaded code.
Constraints. The Local Fallback tactic assumes that there is equivalent code for the offloaded computation on both the mobile device and the surrogate. Because disconnection may happen at any point in the offload process, this tactic is best fit for stateless request-response operations that can be restarted on the mobile device if the operation fails. For stateful operations, program state has to be synchronized between the local and remote versions of the computation. In cases of data staging, results would need to be cached locally until connectivity is available and would have to use local data that can potentially be out-of-date. For systems that implement the Just-In-Time Containers tactic (Section 3.3.3.1) with the Local Fallback tactic, these systems would require a component or a periodic clean-up process that destroys containers that are not being used in order to reduce the load on the surrogate.
Examples. The following two examples illustrate the Local Fallback tactic:

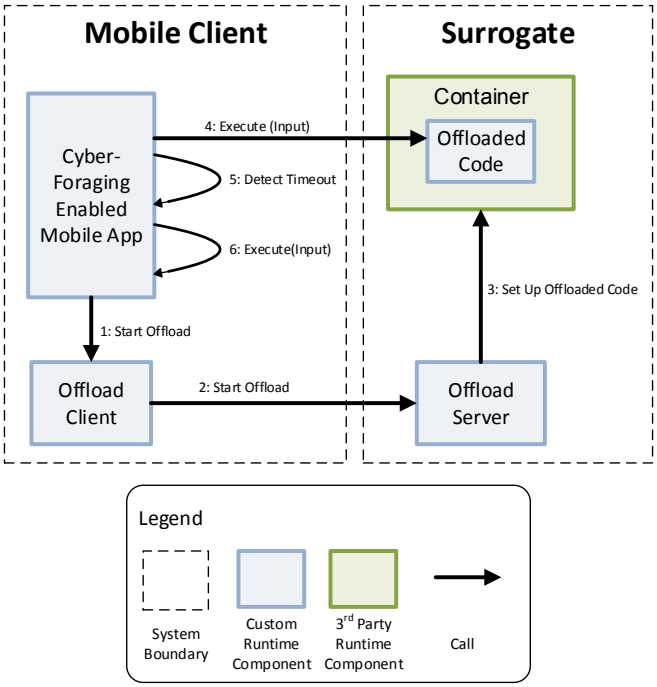

Figure 3.28: Local Fallback

- MAUI [26]: MAUI detects failures using a simple timeout feature that returns control back to the mobile device. If a disconnect occurs, MAUI resumes running the method on the local smartphone. After every offload operation, MAUI returns program state as part of the results, which is applied to the local computation so that state is synchronized between the local and remote computation. Figure 3.29 is based on Figure 3.25 to reflect what occurs in the MAUI system after the remote execution decision has been made. The *App* starts the offload process by invoking the *Client Proxy* which invokes the *Server Proxy* that invokes the remote method. When the *Client Proxy* detects a timeout, it invokes the local method.

- ThinkAir [64]: If the connection fails for any reason during remote execution, the framework falls back to local execution, discarding any data

collected by the profiler. There is no need to synchronize state because an offload request includes the computation itself along with its state and parameters.

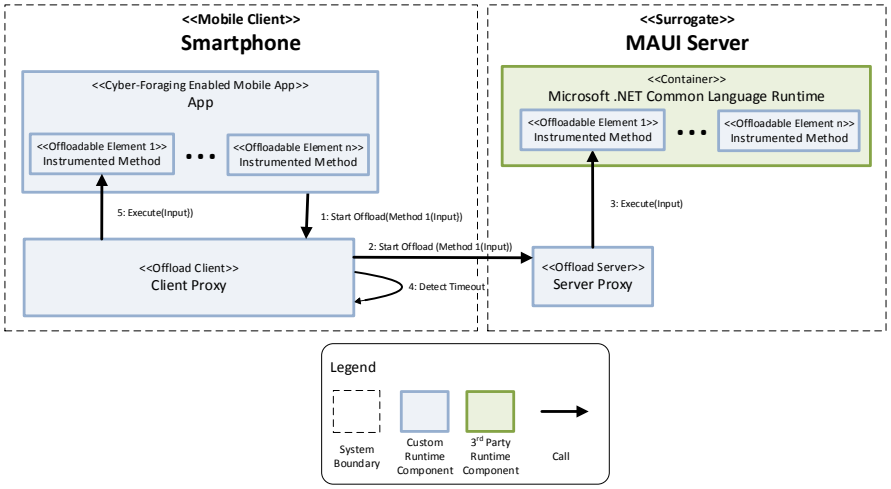

Figure 3.29: MAUI as an Example of the Local Fallback Tactic

Dependencies. The Local Fallback tactic requires a Surrogate Provisioning tactic (Section 3.2.3) to enable the surrogate for computation offload or data staging, and a Computation Offload tactic (Section 3.2.1) or Data Staging tactic (Section 3.2.2) to enable the actual computation offload or data staging process.

3.3.2.2 Opportunistic Mobile-Surrogate Data Synchronization

The Opportunistic Mobile-Surrogate Data Synchronization tactic for fault tolerance is not present in any of the cyber-foraging systems in the primary studies. However, the Collaborative Applications [16] and Virtual Phone [55] systems could easily implement this tactic.

Motivation. Data-reliant cyber-foraging systems, as their name indicates, rely on stored data to fulfill their operations. As in the Local Fallback tactic (Section 3.3.2.1), the mobile device may lose connectivity to the surrogate during the computation offload or data staging process. The Opportunistic Mobile-Surrogate Data Synchronization tactic keeps data synchronized during

periods of connection such that the system can continue operating in periods of disconnection.

Description. Figure 3.30 shows the main elements of the tactic. The data synchronization process can be triggered by the *Cyber-Foraging Enabled Mobile App* right before computation offload by synchronously invoking the *Data Synchronization Client* that ensures that *App Data* is synchronized. It can also be started by the *Data Synchronization Client* asynchronously according to pre-defined *Data Synchronization Policies* that determine an optimal time for synchronization such as periodic synchronization, optimal bandwidth, or detection of re-connection.

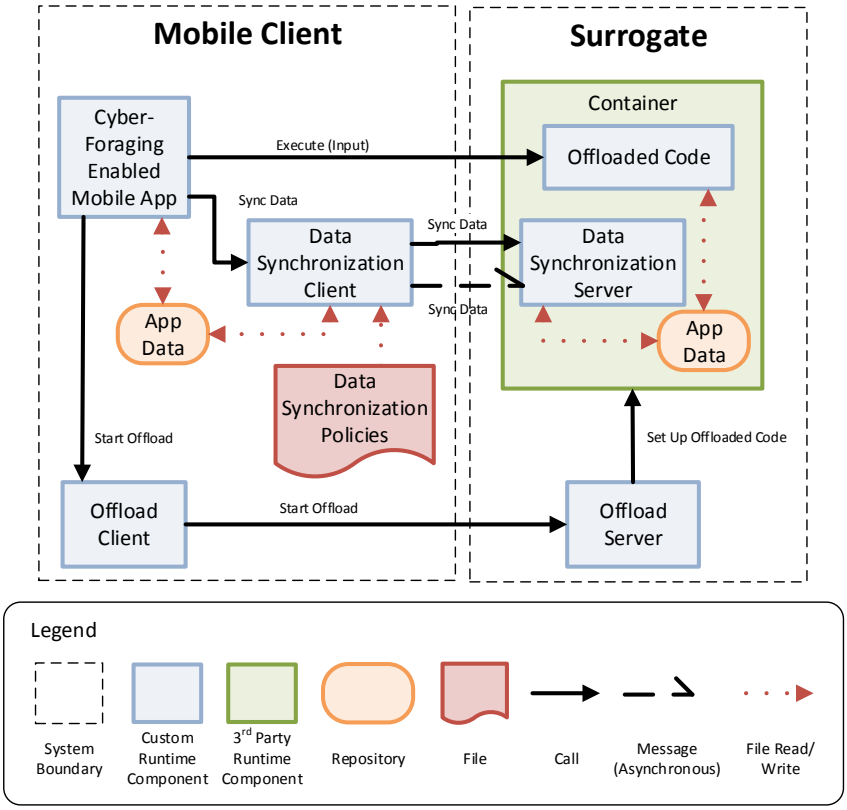

Figure 3.30: Opportunistic Mobile-Surrogate Data Synchronization

Constraints. Systems that implement this tactic need to be aware of the energy consumption on the mobile device for keeping data synchronized. Also, while disconnected, it is possible that data may not be up-to-date, which may lead to incorrect results for applications that operate on time-sensitive data. Finally, like in any distributed data system, conflict resolution between systems that update data simultaneously is challenging.

Examples. As mentioned earlier, there are no systems in the primary studies that implement the Opportunistic Mobile-Surrogate Data Synchronization tactic for fault tolerance as described, but the principle of using distributed storage is the same: to opportunistically keep data/state synchronized without placing the responsibility on the actual applications. The Collaborative Applications [16] and Virtual Phone [55] are computation offload systems that use FUSE for state synchronization between the mobile device and the surrogate to guarantee fidelity of results, meaning that the local and remote computation produce identical results because they are operating on the same state.

Dependencies. The Opportunistic Mobile-Surrogate Data Synchronization tactic requires a Surrogate Provisioning tactic (Section 3.2.3) to enable the surrogate for computation offload or data staging, and a Computation Offload tactic (Section 3.2.1) or Data Staging tactic (Section 3.2.2) to enable the computation offload or data staging process.

Variation: Opportunistic Surrogate-Cloud Data Synchronization. The principles of the Opportunistic Mobile-Surrogate Data Synchronization technique can also be applied to handle disconnection between the surrogate and the cloud, especially for data staging systems. Opportunistic Surrogate-Cloud Data Synchronization enables a system to continue operating in the event of disconnection between the surrogate and the cloud and to synchronize data when reconnection occurs. To support this tactic, the *Data Synchronization Client* runs on the *Surrogate* and the *Data Synchronization Server* runs in the cloud. The Trusted and Unmanaged Data Staging Surrogates [42] is a data staging system that implements this tactic. It uses a distributed filesystem based on Coda[7] between the surrogate and the cloud that supports disconnected operations to maintain data opportunistically synchronized such that it is available on the surrogate when needed. In Figure 3.6 the *Staging Server* includes a *Coda Client* and the *File Server* includes a Coda Server.

[7]Coda is a an advanced networked filesystem that supports disconnected operations. More information is available at `http://www.coda.cs.cmu.edu/`

3.3.2.3 Cached Results

The Cached Results tactic can be found in the Mobile Agents [5], 3DMA [39], Grid-Enhanced Mobile Devices [51], CPA [93], and Sonora [120] systems. These systems contain elements that enable them to cache results on the surrogate that can be delivered to, or retrieved by a mobile device after a disconnection.

Motivation. Offload requests from mobile devices are not always as simple as request-response interactions. Some requests may take a long time to execute or may rely on data that has been gathered and maintained over time. In the case of disconnection between a mobile device and a surrogate during an offload operation, restarting the offload request or losing data is not desired. The Cached Results tactic enables a system to cache results and state on a surrogate until the mobile device is able to reconnect.

Description. Figure 3.31 shows the main elements of the Cached Results tactic with numbers that indicate the sequence of operations. Steps 1 through 4 describe the basic computation offload process. Starting at Step 5, the *Offloaded Code* on the *Surrogate* executes the offloaded operation and tries to send the results back to the *Cyber-Foraging Enabled Mobile App*. However, it detects that the mobile device is disconnected and therefore saves the results in the *Results Cache* along with information that associates the results with a particular mobile client/user. When the *Mobile Client* reconnects to the *Offloaded Code* on the *Surrogate*, the *Offloaded Code* retrieves the results from the *Results Cache* and send them back to the *Cyber-Foraging Enabled Mobile App*. Detecting disconnection could be implemented using assured delivery mechanisms that require receipt acknowledgment, or an external component that detects when a mobile device has been disconnected. In systems that always go through the *Offload Client* and the *Offload Server* for interaction, the disconnection detection mechanism and the interaction with the *Results Cache* would be the responsibility of the *Offload Server*. As another option, using message-oriented middleware for communications would enable the results to be delivered automatically to the *Mobile Client* upon reconnection without requiring a *Results Cache*.

Constraints. The tactic as described is best fit for asynchronous interactions between mobile devices and surrogates or applications that are not time-sensitive or require immediate results. In addition, the tactic requires a mechanism for detecting disconnection from mobile devices.

Examples. The following systems implement the Cached Results tactic:

- Grid-Enhanced Mobile Devices [51]: An example of how this system implements the tactic is shown in Figure 3.32 with numbers to indicate

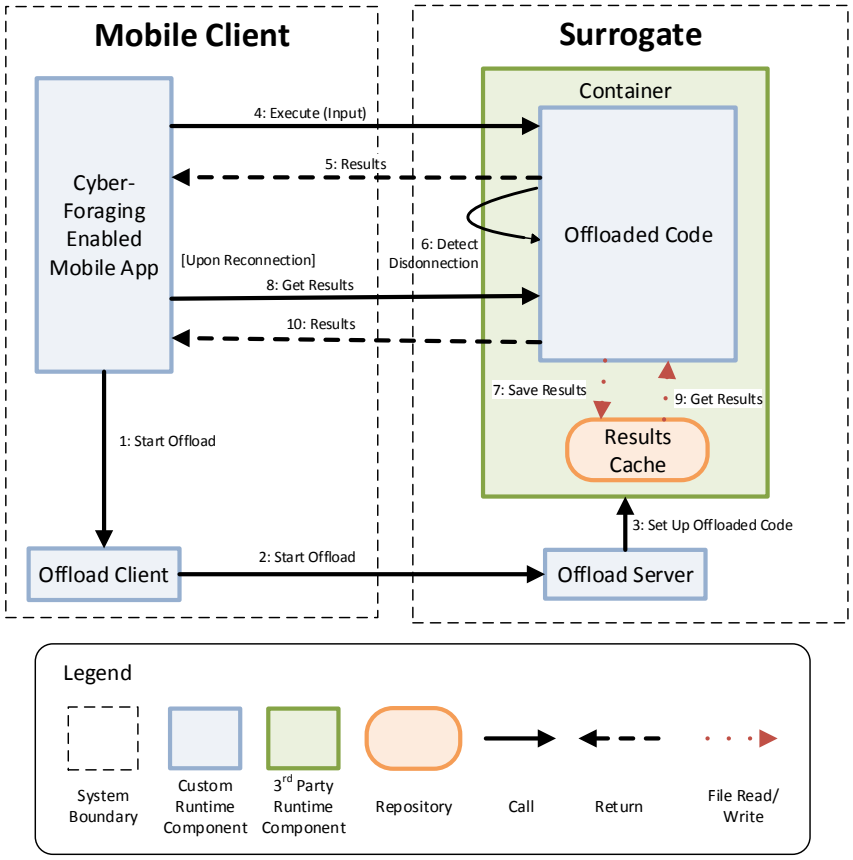

Figure 3.31: Cached Results

sequence of operations. The *User Interface* starts the offload process by invoking the *Connection Manager* with the task to be offloaded. The *Connection Manager* contacts the *Grid Gateway Adapter* on the *Surrogate* which locates a *Grid Service* that can execute the task. Periodically, the *Connection Manager* sends a keep-alive message to the *Grid Gateway Adapter*. If the mobile device fails to send a keep-alive message, after a certain period the *Grid Gateway Adapter* assumes that the mobile device has disconnected, whether voluntarily or involuntarily, and informs the *Device Monitor* to update the device status as disconnected.

When the results from the *Grid Service* come back, the *Grid Gateway Adapter* first checks the device status. If it is disconnected, it saves the results in the *Cache*. When the mobile device is re-connected, the *Grid Gateway Adapter* gets the results from the *Cache* and sends them back to the mobile device.

- Mobile Agents [5]: Offloadable elements in the form of autonomous mobile agents are migrated from a mobile device to a surrogate for asynchronous execution. The mobile agent platform (JADE) handles the migration back to the client once execution is completed and the mobile device is available.

- 3DMA [39]: The middleware used in the 3DMA system uses the concept of spaces to enable asynchronous communication and message buffering. Offload requests from mobile devices are placed in a space, are processed on the surrogate, and results are placed in the same space. When a device becomes disconnected, it waits until a connection is restored, and then reads all available messages (results) from the space.

- CPA [93]: Offload requests are sent to the Cloud Personal Assistant component on the surrogate. The request is added as a user task, the task executes, and the status and result data are added as task information. If the mobile device is disconnected, the user can later log in to the system to check task status and results.

- Sonora [120]: Sonora uses a construct called a sync stream that buffers data during disconnections and resumes normal operation upon reconnection. Connectivity interruptions can either be handled transparently or a mobile app may decide to be notified when disconnections occur.

Dependencies. The Cached Results tactic requires a Surrogate Provisioning tactic (Section 3.2.3) to enable the surrogate for computation offload or data staging, and a Computation Offload tactic (Section 3.2.1) or Data Staging tactic (Section 3.2.2) to enable the computation offload or data staging process.
Variation: Client-Side Data Caching. The tactic as described caches results on the surrogate and sends them to mobile clients upon request or reconnection. A variation of this tactic that is useful for data staging systems that implement the Out-Bound-Pre-Processing (Section 3.2.2.3) is to cache collected data on the mobile device and send it to the surrogate upon reconnection, as shown in Figure 3.33. The Feel the World system [98] is an example

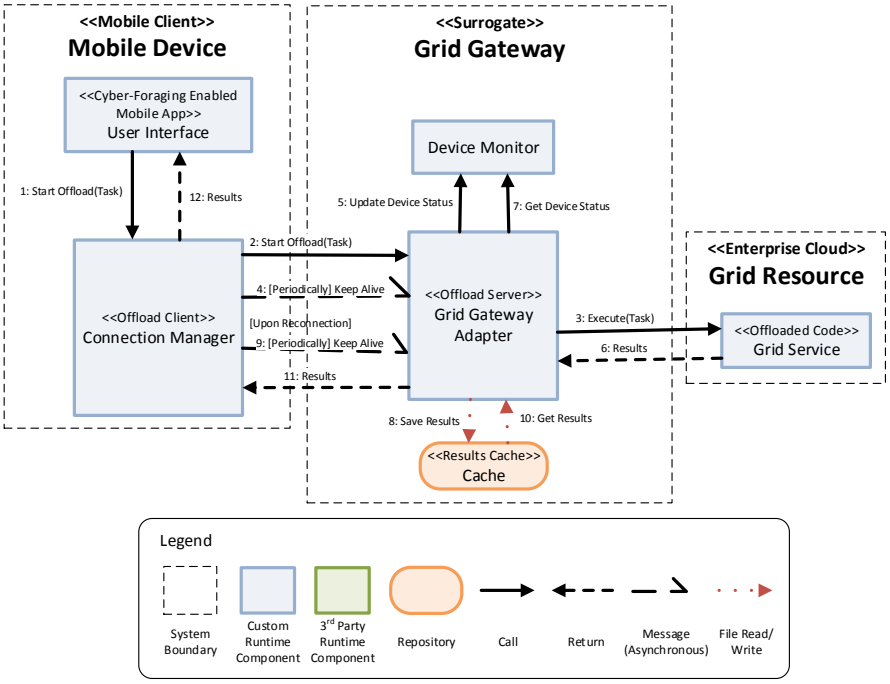

Figure 3.32: Grid-Enhanced Mobile Devices as an Example of the Cached Results Tactic

of this variation that collects sensor data that can be aggregated and/or transformed locally on the mobile client and uploaded to the surrogate in real-time if the connection is available, or at a later moment if it is unavailable.

3.3.2.4 Alternate Communications

The Alternate Communications tactic is present in the Edge Proxy system [6]. The system enables a user to be notified when web pages of interest change (Section 3.2.2.2 contains system details).

Motivation. Cyber-foraging systems typically leverage single-hop, higher bandwidth communication mechanisms such as WiFi or short-range radio instead of broadband wireless (e.g., 3G/4G) because of the potential for energy savings and faster response time (Section 2.5.2.1). However, these mechanisms require the mobile device to be in proximity of the surrogate. The Alternate

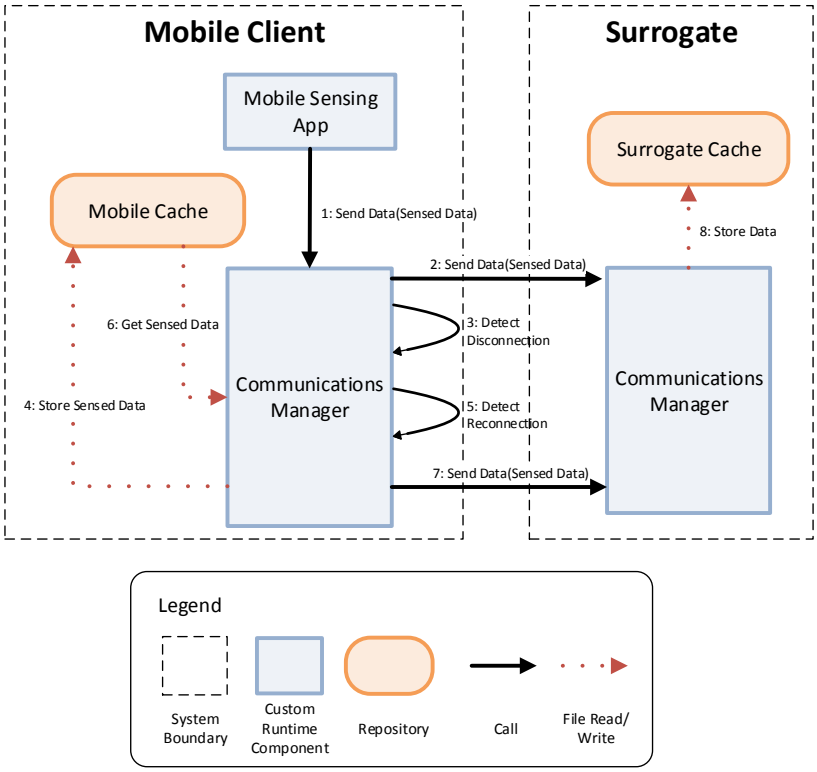

Figure 3.33: Client-Side Data Caching

Communications tactic enables the system to switch to an alternate, potentially less energy-efficient communications mechanism, to continue serving the mobile user in spite of disconnection (even if in a degraded mode due to less amount of information or less timely responses).

Description. Figure 3.34 shows the main elements of the Alternate Communications tactic with number to indicate the sequence of operations. Steps 1 to 11 correspond to the basic offload process using the *Default Communications Manager*. In this tactic the interaction between the *Cyber-Foraging Enabled Mobile App* and the *Offloaded Code* happens through the *Offload Client* and the *Offload Server*. When the *Offload Server* is ready to send the results back to the mobile device it detects that it is disconnected. Therefore, the

results are delivered to the mobile device using the *Alternate Communications Manager*.

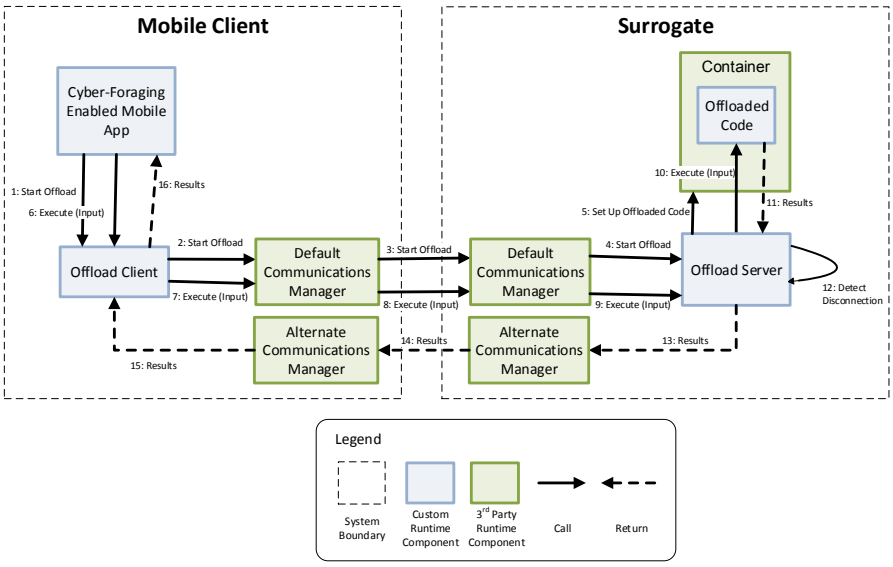

Figure 3.34: Alternate Communications

Constraints. The Alternate Communications tactic as described assumes that the mobile device is enabled to use the alternate communication mechanism. In addition, depending on the type of interaction between the surrogate and the mobile device (i.e., responding to a single offload request or sending data periodically to the mobile device), the surrogate would require a mechanism to determine when connectivity has been restored so it can go back to the default communications mechanism.

Example. Edge Proxy [6] is a data staging system that implements the Alternate Communications tactic. The system enables a user to be notified when web pages of interest change (Section 3.2.2.2 contains system details). Steps 1 to 4 in Figure 3.35 show the registration process using the *WiFi Manager*. When the *Edge Proxy* is ready to send web page changes to the *Mobile Device* and detects that it s disconnected, it leverages the existing Short Message Service (SMS) infrastructure that most wireless carriers provide. It creates a single SMS message with two parts and sends it using the *SMS Manager*. The first part contains control information which includes the number of updates

and the size of the download. The second part is an update summary that includes a list of the pages that have changed, and if particular values were being monitored, the changes that occurred. The *Mobile Proxy* intercepts the SMS message, extracts the control information, and passes the update summary back to the *SMS Manager* for delivery to the user via the *SMS Client*. The *Mobile Proxy* uses the control information to make a decision on how to acquire the updates. Because the user receives an update summary, it may be the case that the information of interest is already there and therefore there is no immediate need to reconnect.

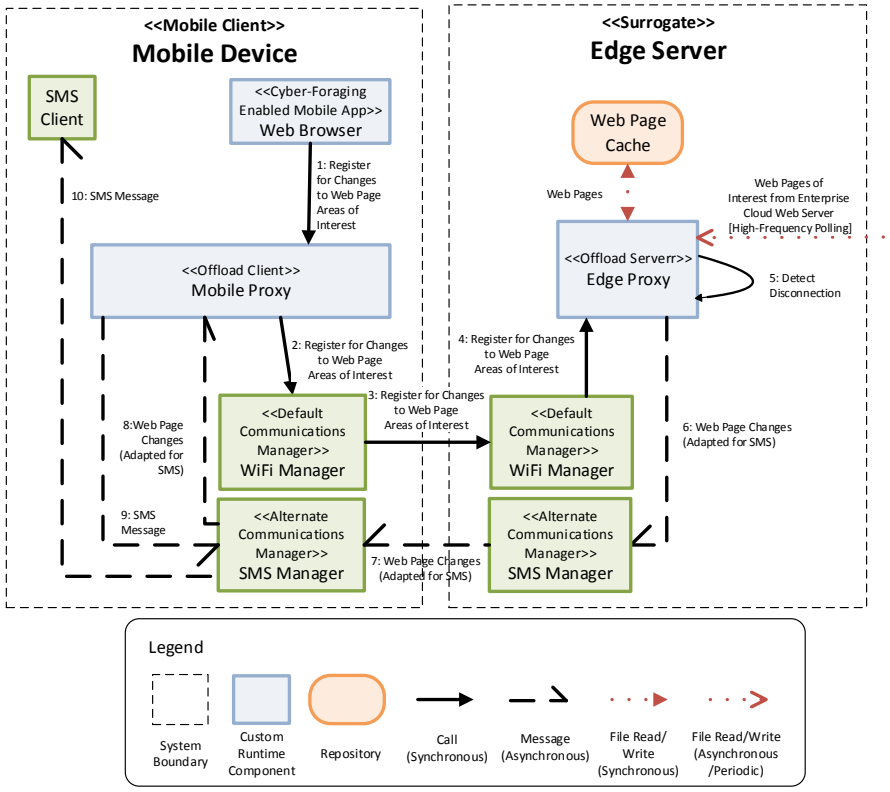

Figure 3.35: Edge Proxy as an Example of the Alternate Communications Tactic

Dependencies. The Alternate Communications tactic requires a Surrogate

Provisioning tactic (Section 3.2.3) to enable the surrogate for computation offload or data staging, and a Computation Offload tactic (Section 3.2.1) or Data Staging tactic (Section 3.2.2) to enable the computation offload or data staging process.

3.3.2.5 Eager Migration

The Eager Migration tactic is present in the Offloading Toolkit and Service system [121]. This system has elements that enable the surrogate to migrate the offloaded computation to another connected surrogate when it detects that it might not be able to continue serving the mobile device that generated the offload request.

Motivation. Due to mobile device mobility or decrease in the quality of the communications channel between the mobile device and the surrogate, the mobile device might lose connectivity to the surrogate. The Local Fallback (Section 3.3.2.1), Cached Results (Section 3.3.2.3), and Alternate Communications (Section 3.3.2.4) tactics for fault tolerance are reactive; that is, they perform a corrective action after the disconnection is detected. The Eager Migration tactic takes a more proactive approach and migrates the offloaded computation to a connected surrogate before it becomes disconnected from the mobile device so that it can continue supporting the offload or data staging operations.

Description. Figure 3.36 shows the main elements of the Eager Migration tactic with numbers to indicate the sequence of operations. Steps 1 to 4 are part of the basic offload process from the *Mobile Client* to the *Source Surrogate*. Periodically, the *Offload Client* sends connection information to the *Offload Server* that it uses to determine if there is a potential for disconnection. This information could be location, signal strength, or available bandwidth. An alternative is for the *Offload Server* to obtain this information periodically using a network monitor. Once the *Offload Server* determines that there is a potential for disconnection, it starts the migration process by contacting the *Offload Server* of the *Target Surrogate* to migrate the offloaded code. It may be the case that there is more the one *Target Surrogate* available, in which case the *Offload Server* would have to select one based on a defined optimization function such as connection bandwidth, load, or available resources on the target. Depending on the granularity of the offloaded code (Section 2.5.1.3) and whether state needs to be transfered or not, the migration process can range from changing the endpoint for communication to migrating just the offloaded code to migrating the full container. Once the migration is complete, the *Offload Server* informs the *Offload Client* to connect to the *Target Surrogate*.

Optionally, the *Offload Server* may need to clean up the offload process by for example stopping running instances, deleting state files, or terminating VMs. The *Target Surrogate* takes over the execution entirely. The interaction between the *Cyber-Foraging Mobile App* and the *Source Surrogate* finishes. The results from invoking the *Offloaded Code* will come from the *Target Surrogate* and any new interactions will be with the *Target Surrogate*.

Constraints. The tactic as described requires the source and target surrogates to be connected. The impact on the user experience will highly depend on the bandwidth between surrogates. In addition, the system has to be able to obtain any parameters for the algorithm that determine potential disconnection such as the distance and communications quality between the mobile device and both the source and target surrogate.

Example. The Offloading Toolkit and Service [121] system implements the Eager Migration technique as shown in Figure 3.37. If the communication between the *Surrogate (Source)* and the *Mobile Handheld* deteriorates based on reaching an established threshold for connection quality, the execution of the offloaded *Classes* is terminated on the *Source Surrogate* and migrated from the *Source Surrogate* to a *Connected Target Surrogate*. The migration consists of serializing and sending the *Classes* from the JVM on the *Source Surrogate* to the JVM on the *Connected Target Surrogate* where they are deserialized and loaded.

Dependencies. The Eager Migration tactic requires a Surrogate Provisioning tactic (Section 3.2.3) to enable the surrogate for computation offload or data staging, and a Computation Offload tactic (Section 3.2.1) or Data Staging tactic (Section 3.2.2) to enable the computation offload or data staging process.

Variation: Lazy Migration. In Eager Migration the offloaded computation fully moves from a *Source Surrogate* to a *Target Surrogate* and the *Mobile Client* continues its interaction with the *Target Surrogate*. In Lazy Migration, the execution of the offloaded computation remains on the *Source Surrogate* but the interaction with the *Mobile Client* is handed off to the *Target Surrogate*. This means that all interaction between the *Mobile Client* and the *Source Surrogate* goes through the *Target Surrogate* that acts as an intermediary. This tactic is not present in any of the systems but was considered as an alternative for the Offloading Toolkit and Service [121] system. It was not selected because of the high bandwidth between surrogates that enabled the system to perform a fast full migration.

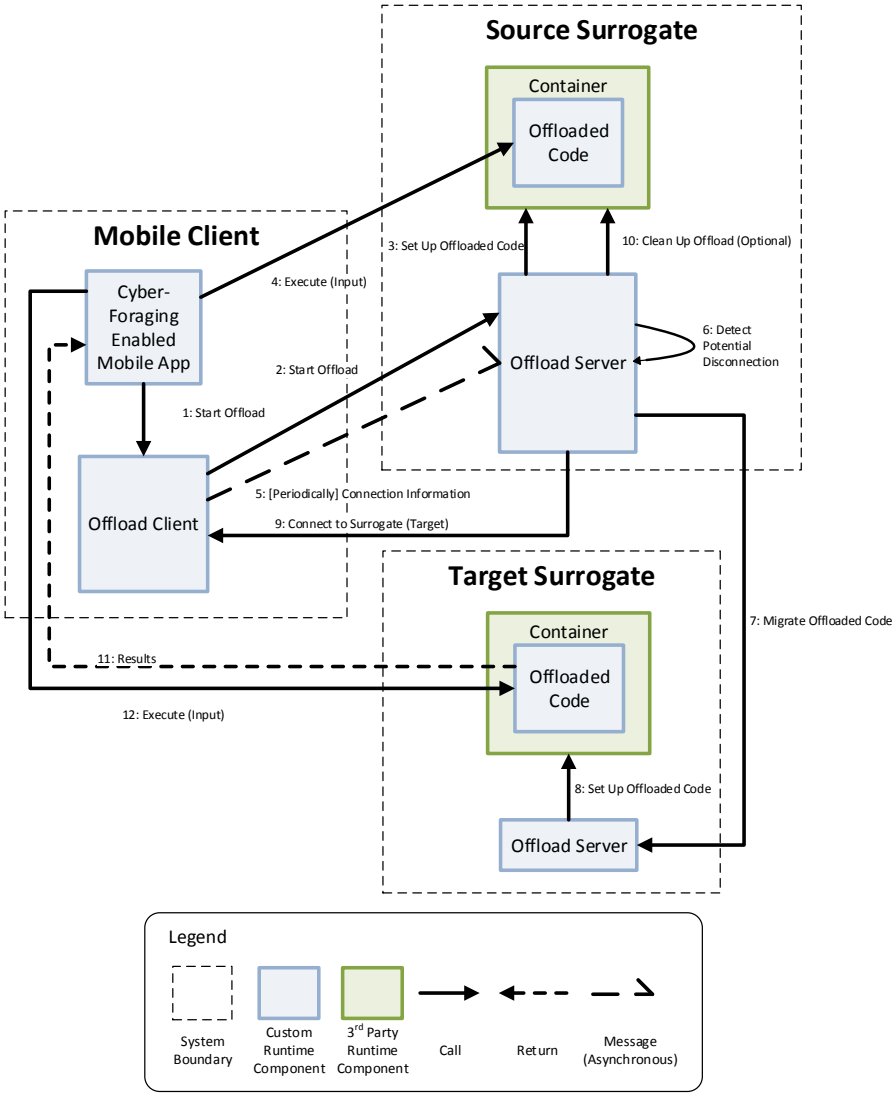

Figure 3.36: Eager Migration

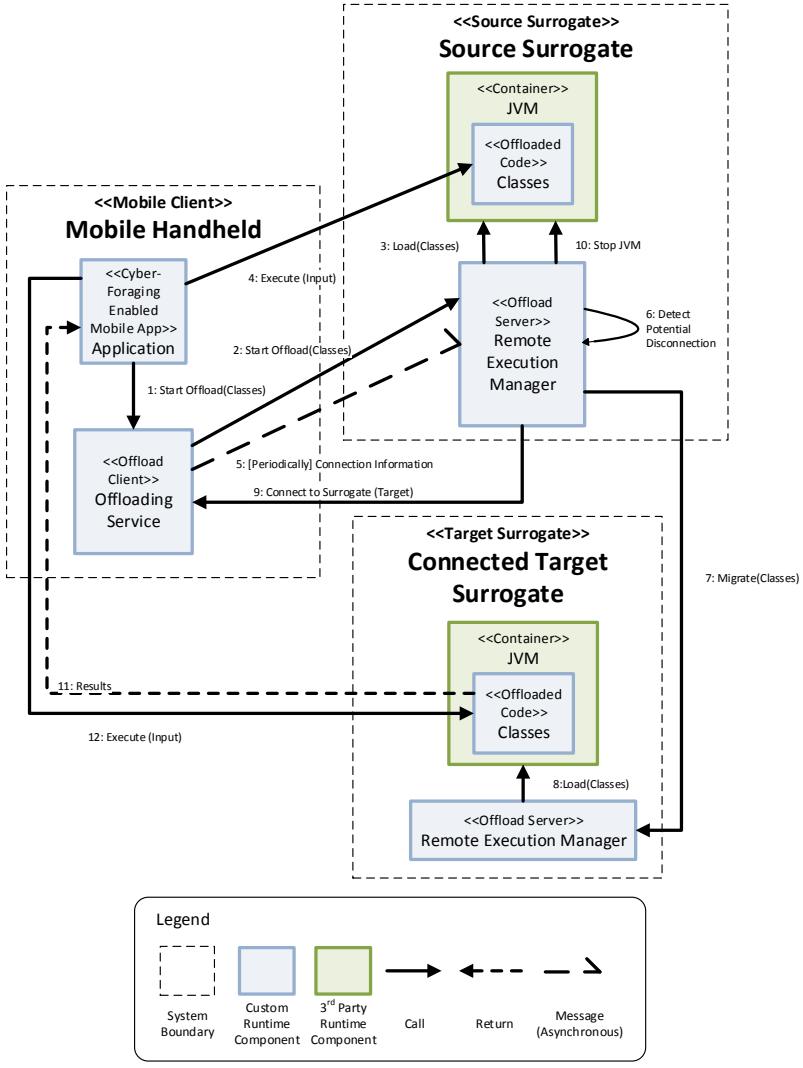

Figure 3.37: Offloading Toolkit and Service as an Example of the Eager Migration Tactic

3.3.3 Scalability/Elasticity

A scenario for Scalability/Elasticity is the following: A mobile app is enabled for cyber-foraging and is leveraging a surrogate for computation offload that is also being leveraged by other mobile apps on other mobile devices. The surrogate is able to optimize computing resources either locally or by leveraging other connected surrogates so that multiple mobile devices can be supported with the goal of minimal effect on user experience due to surrogate load.

3.3.3.1 Just-in-Time Containers

The Just-In-Time Containers tactic is present in the Grid-Enhanced Mobile Devices [51] and VM-Based Cloudlets [108] systems.

Motivation. In an operational cyber-foraging scenario a single surrogate may support multiple mobile users. To decrease the load on a surrogate, and therefore support a greater number of offload requests, 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 the offload request is completed.

Description. Figure 3.38 contains the main elements of the Just-In-Time Containers tactic with numbers to indicate the sequence of operations. The *Cyber-Foraging Enabled Mobile App* starts the offload process by invoking the *Offload Client*. When the *Offload Server* on the *Surrogate* receives the offload request, it creates and starts an instance of the *Offloaded Code* inside the *Container*. The *Cyber-Foraging Enabled Mobile App* interacts with the *Offloaded Code* until it finishes the offload request or closes. At this time the *Cyber-Foraging Enabled Mobile App* ends the offload process by invoking the *Offload Client*. When the *Offload Server* receives the request to end the offload process it destroys the instance of the *Offloaded Code*, thereby releasing the resources that were allocated to it.

Constraints. The tactic as described has a greater startup time than a tactic in which the offloaded code is already running because it has to set up the container, which is the execution environment for the offloaded code.

Examples. In the Grid-Enhanced Mobile Devices [51] system a *Deputy Object* is created for each offload request (task) from a mobile device in the *Grid Gateway*. When the task is completed and the mobile device terminates the connection to the Grid Gateway, resources on the surrogate are released and the Deputy Object is destroyed. The Grid Gateway has a *gateway capacity* that measures its load. Offload requests are granted by the Grid Gateway only if load values are below the gateway capacity. If not, offload requests

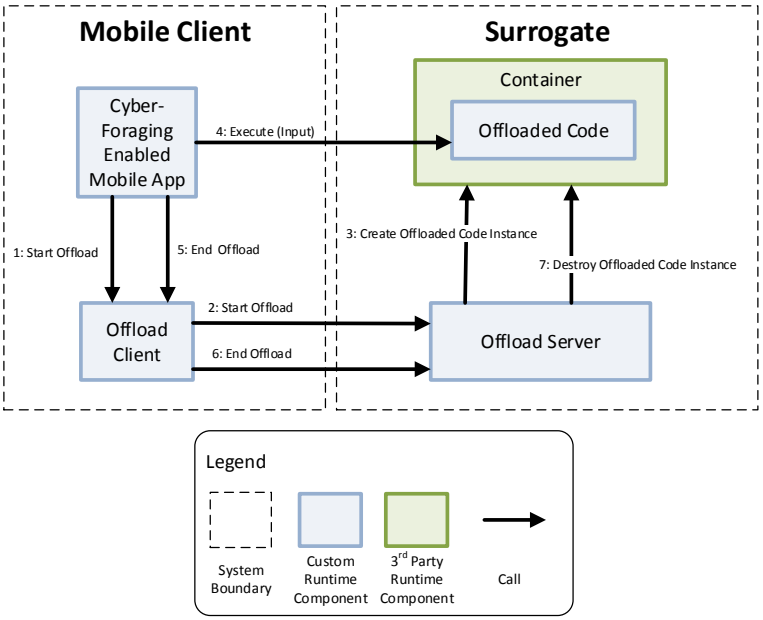

Figure 3.38: Just-In-Time Containers

have to wait until resources are released. In the VM-Based Cloudlets system [108] shown in Figure 3.39, offloaded computation is prepared for execution on a *Cloudlet* using a technique called *VM Synthesis* (details are provided in Section 3.2.3.2). The *KCM Client* starts the offload process. The *KCM Server* creates and installs the synthesized VM inside the *VM Manager* and informs the *KCM Client* that the VM is ready for execution. The *KCM Client* starts a VNC Client that is used to interact with the *Launch VM*. When the *VNC Client* closes, the *KCM Client* ends the offload process by invoking the *KCM Server*, which terminates the *Launch VM*. The term used by the authors to describe the approach is *transient customization of cloudlet infrastructure using hardware VM technology*.

Dependencies. The Just-In Time Containers tactic requires a Surrogate Provisioning tactic (Section 3.2.3) to enable the surrogate for computation offload or data staging, and a Computation Offload tactic (Section 3.2.1) or Data Staging tactic (Section 3.2.2) to enable the computation offload or data staging process.

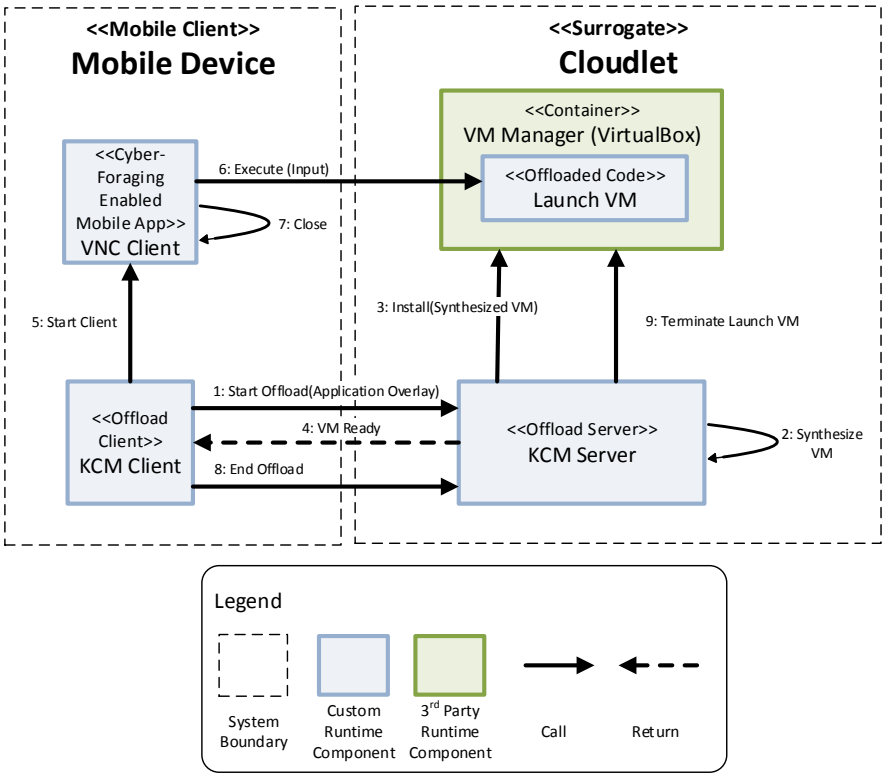

Figure 3.39: VM-Based Cloudlets as an Example of the Just-In-Time Containers Tactic

3.3.3.2 Right-Sized Containers

The Right-Sized Containers tactic is present in the ThinkAir system [64]. This system has elements that create execution containers that are of the appropriate size for the offloaded computation in order to optimize resource usage on the surrogate.

Motivation. In an operational cyber-foraging scenario, a single surrogate may support multiple mobile users. However, not all mobile users are offloading the same computation. Some users may be executing a small task that does not require a large quantity of surrogate resources while others may be executing very computation-intensive tasks that require much more resources.

To optimize resources on a surrogate, and therefore support a greater number of offload requests, the Right-Sized Containers tactic creates a container for the offloaded code that is of the smallest size possible in order to run the offloaded computation, based on computation requirements metadata related to the offloaded code.

Description. Figure 3.40 shows the main elements of the Right-Sized Containers tactic. The *Cyber-Foraging Enabled Mobile App* starts the offload process by invoking the *Offload Client* with *Offloaded Code Metadata* that indicates the computing requirements for the *Offloaded Code*. In the case of pre-provisioned surrogates (Section 3.2.3.1) the *Offloaded Code Metadata* could reside on the *Surrogate*. Based on the metadata received from the *Offload Client*, the *Offload Server* obtains a container from the *Container Repository* that best matches the metadata, meaning that the resources that are required from the *Surrogate* are sufficient to execute the *Offloaded Code*. The *Offload Server* then starts the container and sets up the *Offloaded Code* so that it is ready for execution from the *Cyber-Foraging Enabled Mobile App*.

Constraints. The tactic as described requires a surrogate to maintain different container configurations. In addition, similar to the Just-In-Time Containers tactic (Section 3.3.3.1), it has a greater startup time than a tactic in which the offloaded code is already running because it has to set up the right container as the execution environment for the offloaded code.

Example. The ThinkAir system [64] implements the Right-Sized Containers tactic, as shown in Figure 3.41. When a surrogate (*Application Server*) receives an offload request, the *ThinkAir Framework* on the *Application Server* determines the configuration of the VM (or VMs) to allocate for the task based on *App Requirements* in the offload request that indicate the need for extra computing power (the system has six VM configurations which differ in terms of CPU and memory). The *ThinkAir Framework* starts the selected VM configuration and sets up the offloaded code (*Code and Data*) in the VM.

Dependencies. The Right-Sized Containers tactic requires a Surrogate Provisioning tactic (Section 3.2.3) to enable the surrogate for computation offload or data staging, and a Computation Offload tactic (Section 3.2.1) or Data Staging tactic (Section 3.2.2) to enable the computation offload or data staging process.

Variation: Dynamically-Sized Containers. The ThinkAir system [64] also implements this tactic. If an error occurs at runtime that would indicate that the VM does not have the necessary computing power for the task, such as an *OutOfMemoryError* error, the Client Handler starts a more powerful VM and moves the offload request to the newly started VM.

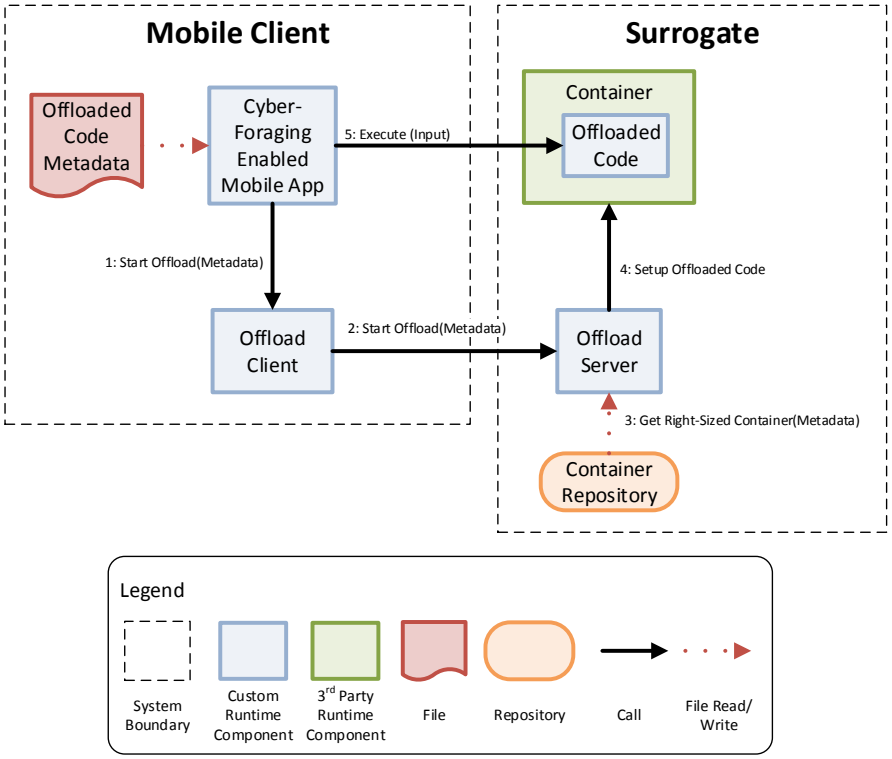

Figure 3.40: Right-Sized Containers

3.3.3.3 Surrogate Load Balancing

The Surrogate Load Balancing tactic is present in the The Cloud Operating System to Support Multi-Server Offloading [56].

Motivation. In an operational cyber-foraging scenario the relationship between mobile devices and surrogates may be many-to-many, meaning that multiple mobile devices may be leveraging multiple surrogates for computation offload and data staging. The Surrogate Load Balancing tactic enables surrogates to send offloaded computation or data to other less-loaded, connected surrogates in order to provide a better user experience to all mobile devices.

Description. The Surrogate Load Balancing tactic uses the same computa-

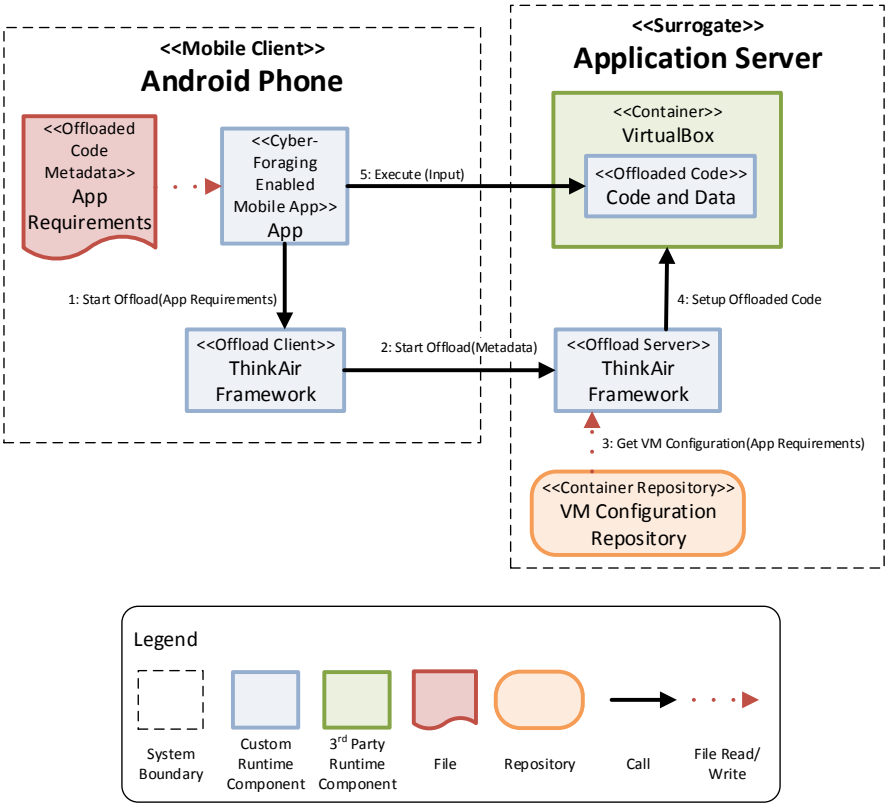

Figure 3.41: ThinkAir as an Example of the Right-Sized Containers Tactic

tion migration techniques as the Eager Migration tactic (Section 3.3.2.5) but for a different purpose (scalability/elasticity instead of fault tolerance). Figure 3.42 shows the main elements of the tactic with numbers that indicate the sequence of operations. Steps 1 to 4 are part of the basic offload process from the *Mobile Client* to the *Source Surrogate*. During the execution of the *Offloaded Code*, the *Load Monitor* informs the *Offload Server* that the *Surrogate* has reached its load threshold. The *Offload Server* then migrates one or more instances of *Offloaded Code* to a *Target Surrogate*. It may be the case that there is more than one connected *Target Surrogate* available, in which case the *Offload Server* would have to select one based on a defined optimization function which should balance the load among all connected surrogates, but may

also include connection bandwidth or available resources on the *Target Surrogate*. Depending on the granularity of the offloaded code (Section 2.5.1.3) and whether state needs to be transfered or not, the migration process can range from changing the endpoint for communication to migrating just the offloaded code (application-level migration) to migrating the full container (container-level migration). Once the migration is complete, the *Offload Server* informs the *Offload Client* to connect to the *Target Surrogate*. The *Offload Server* terminates the instance of the *Offloaded Code* by stopping running instances, deleting state files, or terminating VMs in order to reduce the load on the *Source Surrogate*. The *Target Surrogate* takes over the execution entirely. The interaction between the *Cyber-Foraging Mobile App* and the *Source Surrogate* finishes. The results from invoking the *Offloaded Code* will come from the *Target Surrogate* and any new interactions will be with the *Target Surrogate*.
Constraints. The tactic as described requires the source and target surrogates to be connected. The impact on the user experience will highly depend on the bandwidth between surrogates. The source surrogate requires a mechanism to access the load level of all connected surrogates in order to migrate computation to the less-loaded one and keep the load on all the surrogates balanced.
Example. The Cloud Operating System to Support Multi-Server Offloading (COS) system [56] implements this tactic. *Surrogates* in COS are not connected to the enterprise but to other surrogates to load balance. As shown in Figure 3.43, application modules are implemented as *SALSA Actors* that are self-contained and therefore can easily migrate between a *Source Node* and a *Target Node* (application-level migration). The *Target Node* is selected based on resource availability, communication cost with other actors, and the cost for migration. Because migrating actors is similar to performing a split (removing an actor from a VM on a *Source Node*) and merge (adding an actor to a VM on a *Target Node* operation, COS refers to this aspect of the system as *VM malleability*. The system also has a *COS Manager* that is connected to all Node Managers and is contacted during the *Identify Target Surrogate* operation (Step 6 in Figure 3.43). The *COS Manager* can run on any COS node or in a separate node. When the *Source Node* reaches a load threshold, the *Node Manager* informs the *COS Manager*, which determines the optimal *Target Node* and then prepares the *Target Node* for migration.
Dependencies. Even though the Surrogate Load Balancing tactic does not require any other tactic in order to be implemented, it only makes sense if combined with a Surrogate Provisioning tactic (Section 3.2.3) to enable the surrogate for computation offload or data staging, and a Computation Offload tactic (Section 3.2.1) or Data Staging tactic (Section 3.2.2) to enable the com-

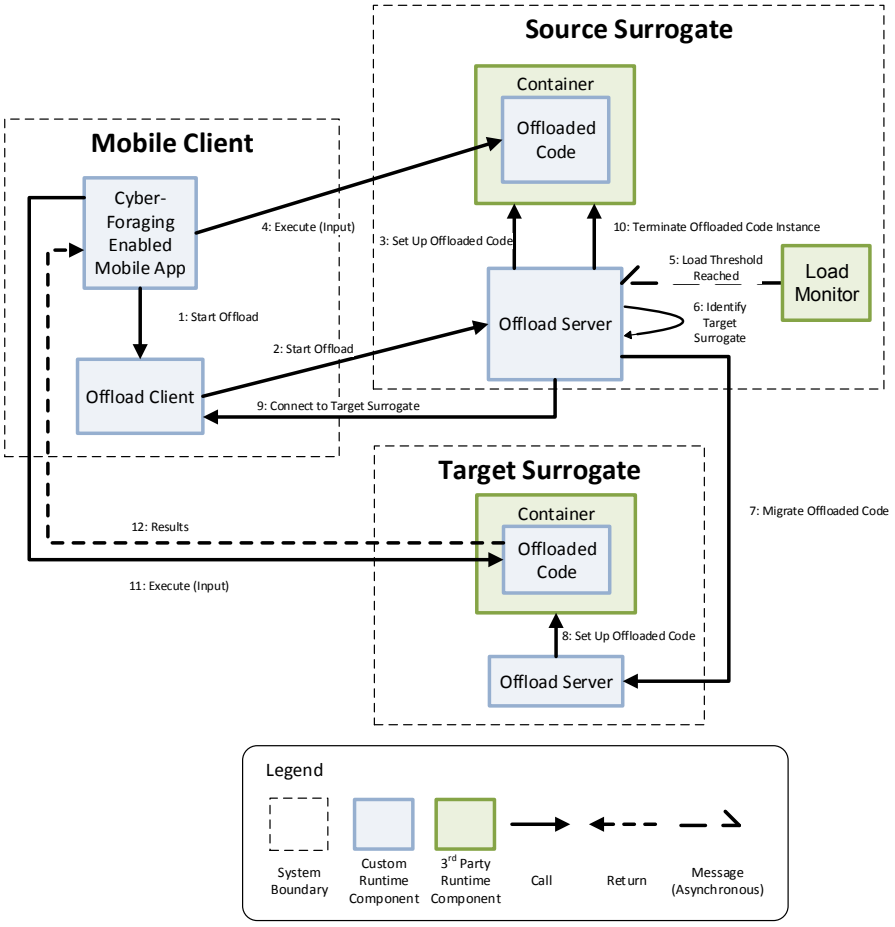

Figure 3.42: Surrogate Load Balancing

putation offload or data staging process. The Surrogate Load Balancing tactic then provides scalability to a computation offload or data staging system.

3.3.4 Security

As stated in the main findings from the primary studies (Section 2.6), there is very little discussion of system-level concerns that have to be addressed when

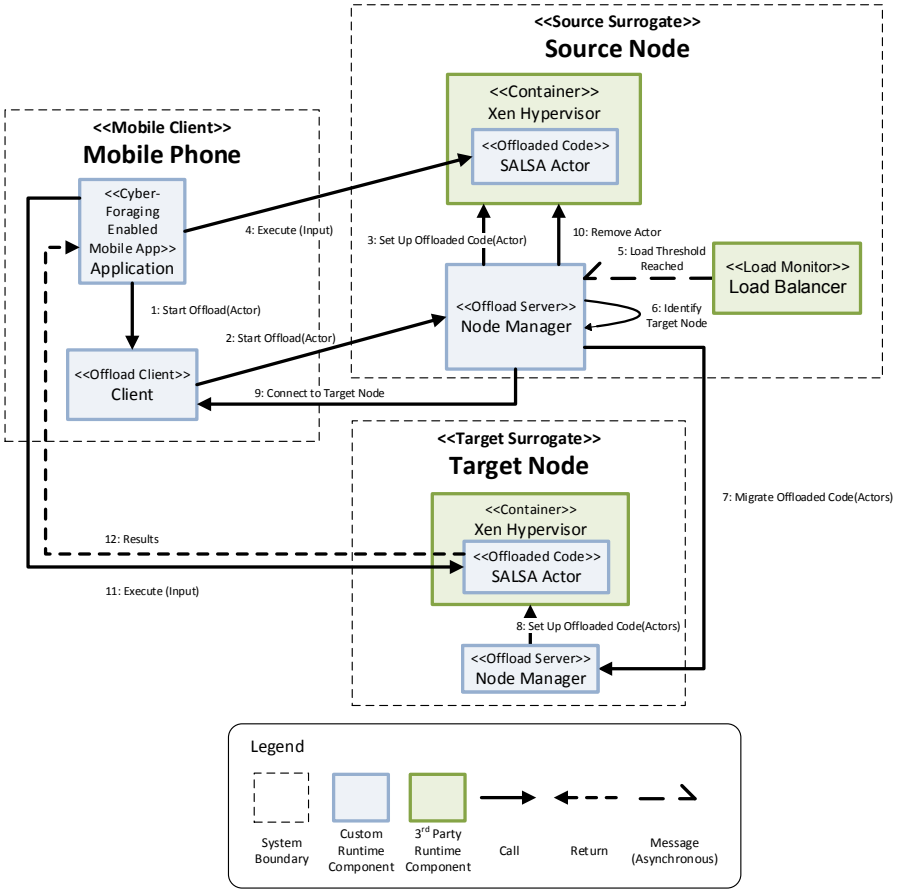

Figure 3.43: Cloud Operating System to Support Multi-Server Offloading as an Example of the Surrogate Load Balancing Tactic

moving from experimental prototypes to operational systems. One of these system-level concerns is security.

A scenario for Security is the following: A mobile app is enabled for cyber-foraging and is in the process of discovering a surrogate for computation offload. User and surrogate credentials are exchanged and validated before the offload process so that the mobile app and surrogate can interact according to agreed security policies.

3.3.4.1 Trusted Surrogates

Motivation. When a mobile device discovers a surrogate it expects a trustworthy surrogate execution environment, meaning that once an offload operation starts, code and data are not maliciously modified or stolen and that it provides trustful services. In the same way, a surrogate expects that a mobile device is a valid client and that it will not offload malicious code or use it as a vehicle to other code and data offloaded by other mobile devices. The Trusted Surrogate tactic adds this trust element to the interaction between a mobile device and a surrogate.

Description. As mentioned earlier, there is not much discussion about security or trust in the primary studies. An approach that is shown in some of the primary studies is to own the surrogate. Roam [20] assumes that a user would only offload applications among his/her personal devices such as cell phones, PDAs, and home PCs. Collaborative Applications [16] and SPADE [112] offload only to personal trusted servers such as a home server. The Grid-Enhanced Mobile Devices system [51] assumes a pre-existing trust relationship between mobile devices, the *Grid Gateway* that serves as an intermediary between the mobile device and the surrogates, and the surrogates (*Grid Service Providers*). In the proposed implementation, the mobile user uses his own desktop as the Grid Gateway and all Grid Service Providers are owned by the user's organization.

Another hardware-based approach that is suggested for establishing trust, but not implemented in any of the primary studies, is to use an on-board secure hardware component such as Trusted Platform Module (TPM). TPM is a device/chip that has a unique and secret RSA key that is burned into it when it is produced.[8] Collaborative Applications [16], Virtual Phone [55] and VM-Based Cloudlets [108] suggest the use of TPM for stronger levels of trust.

In the Collective Surrogates system [48] only the trusted *Collective Manager* that serves as the broker between mobile devices and surrogates has direct access to the VM running on a surrogate (*Participating Node*). This system exploits the isolation provided by VM technology for safely running arbitrary code provided by mobile devices. However, the system assumes a trust relationship between mobile device and the Collective Manager.

While a password- or hardware-based approach is useful for some scenarios, it is not appropriate in more dynamic scenarios in which mobile devices discover nearby surrogates that are not owned by the owner of the mobile device (Section 3.2.4.3). These scenarios require more dynamic ways of establish-

[8]The ISO/IEC 11889 specification for TPM is available at `http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html`

ing trust between mobile devices and surrogates, such as a third-party, online trusted authority that validates credentials or a certificate authority that provides certificates and keys for authentication, to determine if data or code has been tampered with, or even encryption (as an example, the Virtual Phone system [55] has a fully-encrypted filesystem on the surrogate to ensure that data is not accessible by surrogate owners or other virtual machines running on the surrogate).

Constraints. Each of the approaches listed above has constraints related to how the trust relationship is established. Password-based approaches such as those employed by systems in which surrogates are owned by the mobile device user require users to be registered on the surrogate. Hardware-based approaches such as TPM require surrogates to have TPM chips on them. Systems that rely on third parties have to be connected to online authorities or require certificates and keys to be obtained from a central certificate authority.

Example. The only system that implements a trust solution that uses a third-party trusted authority is the Trusted and Unmanaged Data Staging Surrogates system [42]. This system was used as an example for the Pre-Fetching tactic in Figure 3.6. A subset of this figure with detail related to the trust components is presented in Figure 3.44 with numbers to indicate the sequence of operations. The user's idle *Desktop* serves as the trusted third party that sits in between the *Server* and the *Surrogate*. When the *File Client* requests a file, the *Client Proxy* communicates with the *Data Pump* that runs on the *Desktop* to obtain the key and hash for the requested data file. The *Data Pump* retrieves the data file from the *File Server* and encrypts it before sending it to the *Surrogate* for staging it in the *Cache*. It then sends the *Client Proxy* the key and hash for the file so it can be compared it to the hash of the file that is retrieved from the *Surrogate* to determine if the file has been tampered with.

Dependencies. Even though the Trusted Surrogate tactic does not require any other tactic in order to be implemented, it only makes sense if combined with a Surrogate Provisioning tactic (Section 3.2.3) to enable the surrogate for computation offload or data staging, and a Computation Offload tactic (Section 3.2.1) or Data Staging tactic (Section 3.2.2) to enable the computation offload or data staging process. The Trusted Surrogate tactic then provides a trusted environment for computation offload or data staging.

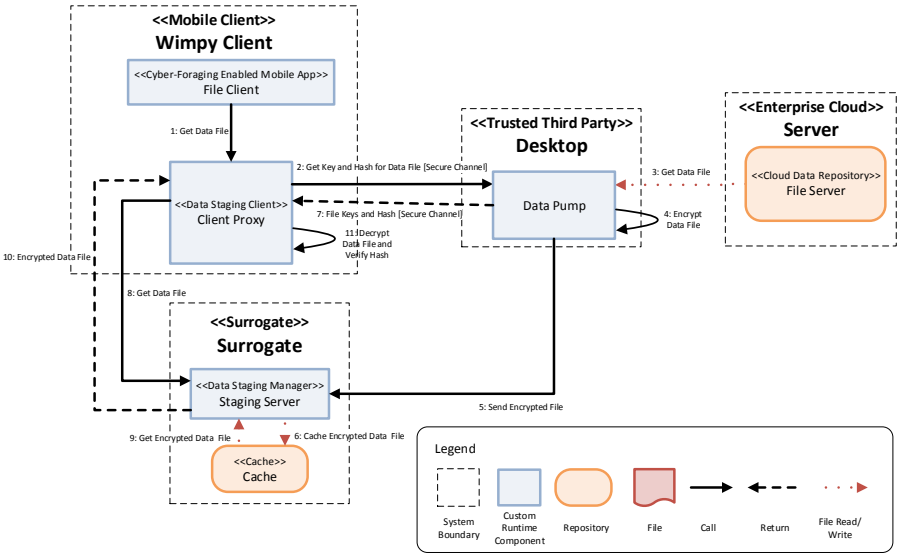

Figure 3.44: Trusted and Unmanaged Data Staging Surrogates as an Example of the Trusted Surrogates Tactic

3.4 Summary and Conclusions

This chapter presented a set of architectural tactics for cyber-foraging, derived from the architectural design decisions in the primary studies identified in the SLR described in Chapter 2. Common design decisions present in the cyber-foraging systems were codified into architectural tactics for cyber-foraging, and then grouped into functional and non-functional tactics.

Functional tactics provide the basic cyber-foraging operations. The primary studies show that at a minimum a cyber-foraging system implements (1) a tactic for computation offload and/or data staging, (2) a tactic for surrogate provisioning, and (3) a tactic for surrogate discovery.

Non-functional tactics are combined with the functional tactics to support required system qualities. We identified tactics for resource optimization, fault tolerance, scalability/elasticity, and security. Even though the latter is a key system quality to guarantee in a cyber-foraging system, especially in situations in which mobile devices discover available surrogates in the environment, there was minimal reference to security and trust in the primary studies. There are

also other system qualities, such as ease of deployment and reliability, that are not considered by the studied cyber-foraging systems, yet are key for the deployment of operational cyber-foraging systems. We see these gaps as an opportunity for research and development.

The goal of the tactics is to serve as a reference for architects designing cyber-foraging systems. A software architect would first select the functional tactics that implement the essence of a cyber-foraging system:

- A computation offload and/or data staging tactic that fulfills cyber-foraging functionality
- A surrogate provisioning tactic so that the offloaded code or data staging/processing code is available on the surrogate
- A surrogate discovery tactic so that mobile devices can locate and connect to available surrogates

Then, based on additional non-functional requirements, a software architect would navigate the catalog identifying tactics that can fulfill the requirements. However, there are always tradeoffs when making architectural decisions, which make decision models such as the ones that will be presented in Chapter 8 a valuable tool for architects.

In the meantime, the next three chapters present case studies that validate the architectural tactics described in this chapter.

4

Case Study 1: Tactical Cloudlets — Cyber-Foraging for Computation Offload

This chapter addresses research question RQ2 and is the first of three case studies to validate the architectural tactics presented in Chapter 3. The goal of this case study is to discover the architectural design decisions in the existing implementation of the Tactical Cloudlets system developed by the Carnegie Mellon Software Engineering Institute to support computation offload in tactical environments [32], and then verify the mapping of the architectural design decisions to architectural tactics for cyber-foraging.

4.1 Introduction

A set of architectural tactics for cyber-foraging was presented in Chapter 3. However, these tactics need to be validated in real cyber-foraging systems to fully address research question RQ2: *What architectural tactics can be derived from the identified architectural design decisions?*.

The goal of this first case study is to discover the architectural design decisions in the existing implementation of the Tactical Cloudlets system developed by the Carnegie Mellon Software Engineering Institute to support computation offload [32], and then verify the mapping of the architectural design decisions to the architectural tactics for cyber-foraging.

We followed the guidelines for conducting case studies from [15] and [118]. Accordingly, the structure of the chapter follows the steps proposed in these guidelines. Section 4.2 presents the case study design, including research questions and procedures for data collection and analysis. Section 4.3 presents the results of the case study and threats to validity. Section 4.4 concludes the

chapter with a summary of the findings as well as their implications and limitations.

4.2 Case Study Design

4.2.1 Research Questions

Given the goal to discover architectural design decisions in the existing implementation of the Tactical Cloudlets system, we defined the following research questions to be answered in the execution of the case study.

- Which of the architectural tactics for cyber-foraging can be identified in the Tactical Cloudlets system?
- How do the implemented tactics support their intended functional and non-functional requirements?

4.2.2 Data Collection Procedure

Data collection involves identifying the data to be collected, defining a data collection plan, and defining how the data will be stored [15]. Given that the goal of this case study is to discover the architectural design decisions in an existing system implementation, and both the system artifacts and system developers are available, the data collection is executed with an independent analysis of work artifacts (third degree data collection method) combined with developer interviews for validation (first degree data collection method) [118].

We therefore define the following steps to collect data about the design and implementation of the Tactical Cloudlets system that will enable us to answer the case study research questions:

1. Understand system requirements: System requirements are gathered from the project Wiki, system documentation, and publications. The identified requirements are documented and confirmed by members of the development team.
2. Recover software architecture: The software architecture is recovered from the project Wiki, system documentation, and publications. The as-designed architecture is compared to the as-is architecture through code inspection of the code available at `https://github.com/SEI-AMS/pycloud` and verification with the development team.

3. Map architectural design decisions to system requirements: Architectural design decisions are mapped to system requirements in order to fully understand how each requirement was met.

4.2.3 Analysis Procedure

Once the system requirements and architectural design decisions are fully understood we perform two activities as part of the analysis.

1. Map architectural design decisions to architectural tactics: The identified architectural design decisions are mapped to elements of the tactics presented in Chapter 3. We do this by (1) selecting tactics that could meet systems requirements based on the description of the tactic, and (2) mapping components of the tactics to component(s) in the architecture that perform each component role. Both matches and gaps are identified in order to determine completeness of the tactics, as well as variations of the tactics implemented in the system to fulfill specific requirements.

2. Qualitatively and quantitatively (if possible) determine if the implementation of the tactics meets the corresponding system requirements: Through system testing, data collected (and published) by system developers, as well as discussions with the system developers, we determine if the implementations of the tactics meet their intended requirements.

4.3 Results

4.3.1 System Context

Tactical environments, such as those in which first responders and military personnel operate, are characterized by dynamic context, limited computing resources, disconnected-intermittent-limited (DIL) network connectivity, and high levels of stress. Forward-deployed, discoverable, virtual-machine-based tactical cloudlets can be hosted on vehicles or other platforms to

- provide infrastructure to offload computation,
- provide forward data staging for a mission,
- perform data filtering to remove unnecessary data from streams intended for mobile users, and
- serve as collection points for data heading for enterprise repositories.

The forward-deployed, single-hop proximity to mobile devices promotes energy efficiency as well as lower latency (faster response times).

Given the uncertainty and dynamicity of tactical environments, one of the main drivers for the Tactical Cloudlets system is survivability, defined as the capability of a system to continue functioning in spite of adversity [32].

4.3.2 System Requirements

The requirements of the Tactical Cloudlets system can be divided into functional and non-functional requirements.

4.3.2.1 Functional Requirements

Tactical Cloudlets need to satisfy the following functional requirements.

- **FR1: Offload of Computation-Intensive Operations:** Applications that are useful to first responders and military personnel include speech and image recognition, natural language processing, and situational awareness. These are all computation-intensive tasks that take a heavy toll on the device's battery power and computing resources and should therefore be offloaded to proximate, more powerful cloudlets.
- **FR2: Cloudlet Discovery:** Due to the dynamic nature and potential mobility of cloudlets in tactical environments (e.g., vehicle-hosted cloudlets), mobile devices need to be able to discover nearby cloudlets.
- **FR3: Disconnected Operations:** In tactical environments it is not possible to guarantee connectivity between cloudlets in the field and the cloud. Therefore, offloaded capabilities should be self-contained and pre-loaded so they do not require connectivity to the cloud in order to operate.
- **FR4: Support for Separate Deployment of Mobile Devices and Cloudlets:** Cloudlets should be able to be used by mobile devices already deployed or available in the field. Therefore the cloudlet should enable mobile devices to be provisioned with the required apps to use its capabilities.
- **FR5: Optimal Cloudlet Selection:** If more than one cloudlet is available, the mobile device should offload computation to the cloudlet that is likely to return a response in the shortest amount of time, before the mobile device loses connectivity to the cloudlet.

- **FR6: Cloudlet Management:** In addition to being able to provision the cloudlet with capabilities for use by mobile devices, the cloudlet administrator should be able to see what capabilities have been started from mobile devices as well as start capabilities and stop capabilities as needed.
- **FR7: Cloudlet Migration:** Due to the potential mobility of cloudlets in tactical environments, offloaded capabilities should be able to migrate between cloudlets when requested.

4.3.2.2 Non-Functional Requirements

Tactical Cloudlets need to satisfy the following non-functional requirements.

- **NFR1: Energy Efficiency**: Energy consumption on the mobile device when offloading computation-intensive operations (request, execution, and response) should be less than energy consumed by local execution.
- **NFR2: Scalability and Elasticity**: Tactical cloudlets cannot be servers with huge computing power due to power availability and size limitations of what can be carried into a tactical environment to support a mission. Tactical Cloudlets therefore should only run capabilities when they are actively being used by mobile devices.
- **NFR3: Ease of Deployment and Re-Deployment**: First responders and military personnel executing a mission cannot rely on the availability of IT personnel in the field to help with cloudlet setup. Therefore, tactical cloudlets should be easy to set up by non-IT personnel.

4.3.3 System Architecture and Design

The as-is architecture for the Tactical Cloudlets system is shown in Figure 4.1. The main elements of the architecture are:

- Client: Mobile device running Android 4.x that hosts three main components:
  - Cloudlet-Ready App(s): Mobile apps that are set up to offload computation to a cloudlet.
  - Cloudlet Client GUI: Mobile app that is used to access the app store capability.

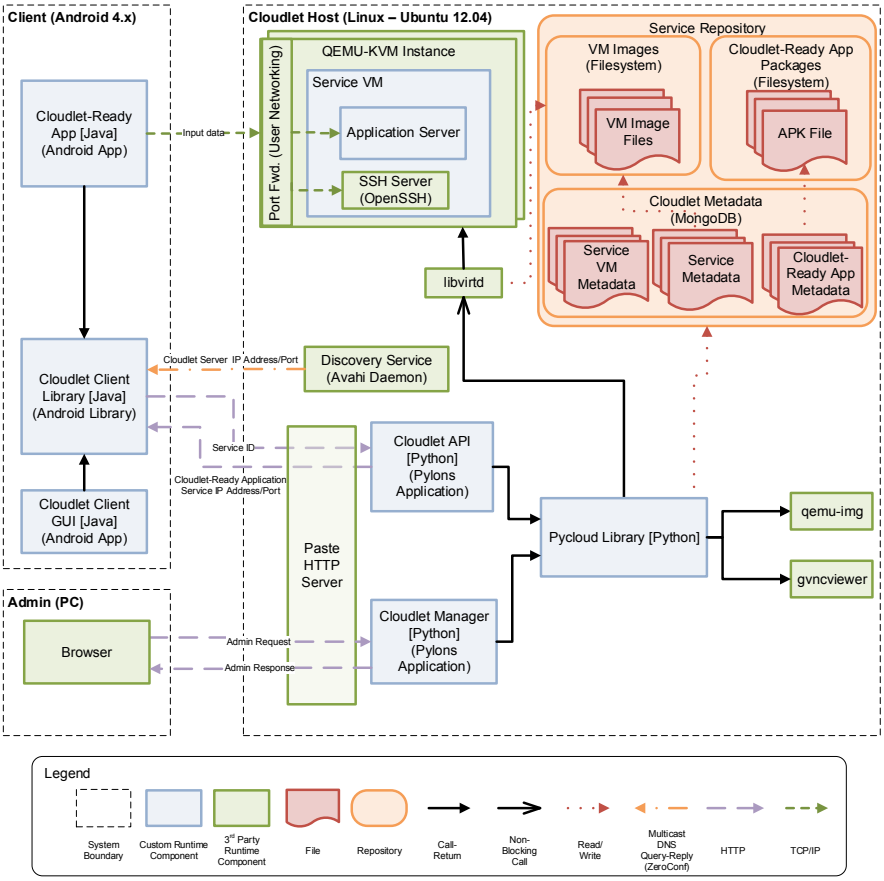

Figure 4.1: High-Level Architecture of the Tactical Cloudlets System

  - Cloudlet Client Library: Library that is used by the two components above to discover cloudlets, retrieve cloudlet metadata, select cloudlets, and offload computation. It interacts with the Cloudlet Host using HTTP.

- Cloudlet Host: Linux server that runs a tactical cloudlet. The main components are:

  - PyCloud Library: Python component that implements the core

cloudlet functionality.

  - Cloudlet API: Python component that is used by the Cloudlet Client Library to start Services as Service VMs.
  - Cloudlet Manager: Python web application that is used by an administrator to manage Services (along with their VM Images) and Cloudlet-Ready Apps.
  - Service Repository: Each capability that is made available to mobile apps is considered a service. A running service is called a Service VM. Each service has associated metadata (Service Metadata), the actual capabilities packaged as VM disk and memory images (VM Images), and one or more Cloudlet-Ready Apps that can use the capability (Cloudlet-Ready App Packages). In addition, the repository stores metadata related to running services (Service VM Metadata) and the available Cloudlet-Ready Apps (Cloudlet-Ready App Metadata)
  - QEMU-KVM Instance: Each Service VM runs inside a QEMU-KVM virtual machine instance.

- Admin (PC): Browser that is used to access the Cloudlet Manager web application.

4.3.4 Mapping of Architectural Design Decisions to Architectural Tactics

The mapping of functional and non-functional requirements to components of the architecture is shown in Table 4.1.

Architectural design decisions implemented in the Tactical Cloudlets system were mapped to functional and non-functional architectural tactics described in Chapter 3. The mapping was performed in the following way:

1. For each system requirement, we selected a tactic that could meet system requirements based on its description.

2. For each component in the selected tactic, we identified component(s) in the architecture that performed the functionality described in the tactic.

3. We created an architecture diagram for the Tactical Cloudlets system in which component names in the tactic are used as stereotypes in the architecture components to indicate the mapping between components.

Table 4.1: Mapping of Functional and Non-Functional Requirements to the Architecture of the Tactical Cloudlets System

Component	FR1: Offload	FR2: Discovery	FR3: Disconnected Ops.	FR4: Separate Deployment	FR5: Selection	FR6: Management	FR7: Migration	NFR1: Energy Efficiency	NFR2: Scalability	NFR3: Ease of Deployment
Cloudlet Client Library	X	X		X	X			X		
Discovery Service		X								
Pycloud Library	X		X	X	X	X	X	X	X	X
Service Repository	X		X	X		X	X	X	X	X
Cloudlet Metadata				X	X					
Cloudlet Client GUI				X						
Cloudlet Manager						X	X			X
QEMU-KVM Instance	X							X	X	

4. We then added components found in the Tactical Cloudlets system that were not present in the tactic, but were important for the implementation of the tactic.

The follwing subsections describe how each selected tactic was identified and implemented in the Tactical Cloudlets system.

4.3.4.1 Computation Offload

The Computation Offload tactic (Section 3.2.1) enables mobile clients to offload expensive computation to surrogates, as shown in Figure 4.2(a). In summary, a component on the mobile client (Offload Client) coordinates the offload process with its counterpart on the surrogate (Offload Server). The Offload Server sets up the offloaded code in an execution container for the Cyber-Foraging-Enabled Mobile App to use.

The Computation Offload tactic can be identified in the Tactical Cloudlets architecture as shown in Figure 4.2(b), with numbers to indicate the sequence

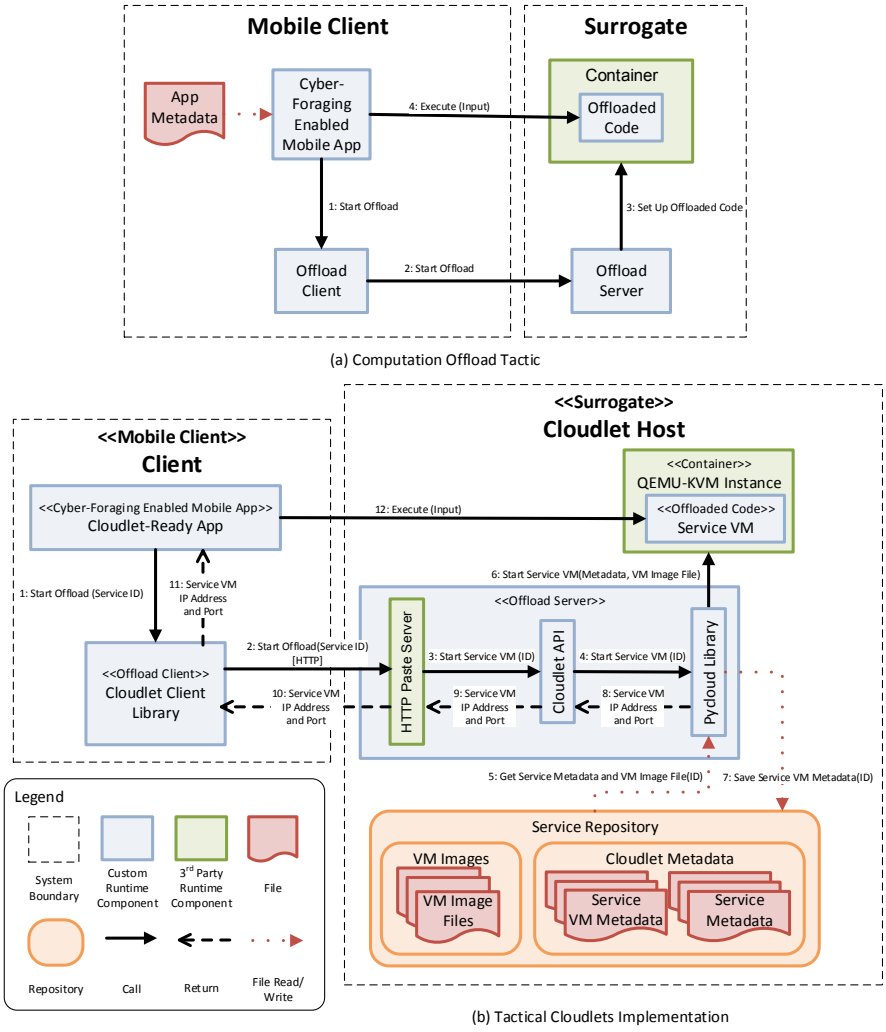

Figure 4.2: Tactical Cloudlets Implementation of the Computation Offload Tactic

of operations. The component names in Figure 4.2(a) are used as stereotypes for the components in Figure 4.2(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included. The computation offload operation takes place as follows:

1-4. The Cloudlet-Ready App requests to offload service *Service ID*.

5. The Pycloud Library retrieves Service Metadata and VM Image Files for *Service ID*.

6. The Pycloud Library starts the Service VM as a QEMU-KVM Instance.

7. The Pycloud Library saves Service VM Metadata in the Service Repository.

8-11. The Pycloud Library returns the IP address and port on which the Service VM is listening.

12. The Cloudlet-Ready App opens a socket to the given IP address and port and starts interacting with the Service VM.

4.3.4.2 Pre-Provisioned Surrogate

In the Pre-Provisioned Surrogate tactic (Section 3.2.3.1) surrogates are provisioned before their deployment with the capabilities that are offloaded by mobile clients, as shown in Figure 4.3(a). In summary, an Admin Client enables cloudlet administrators to add capabilities to a surrogate via a Surrogate Manager.

The Pre-Provisioned Surrogate tactic can be identified in the Tactical Cloudlets architecture as shown in Figure 4.3(b), with numbers to indicate the sequence of operations. The component names in Figure 4.3(a) are used as stereotypes for the components in Figure 4.3(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included. Provisioning a cloudlet with a service capability takes place as follows:

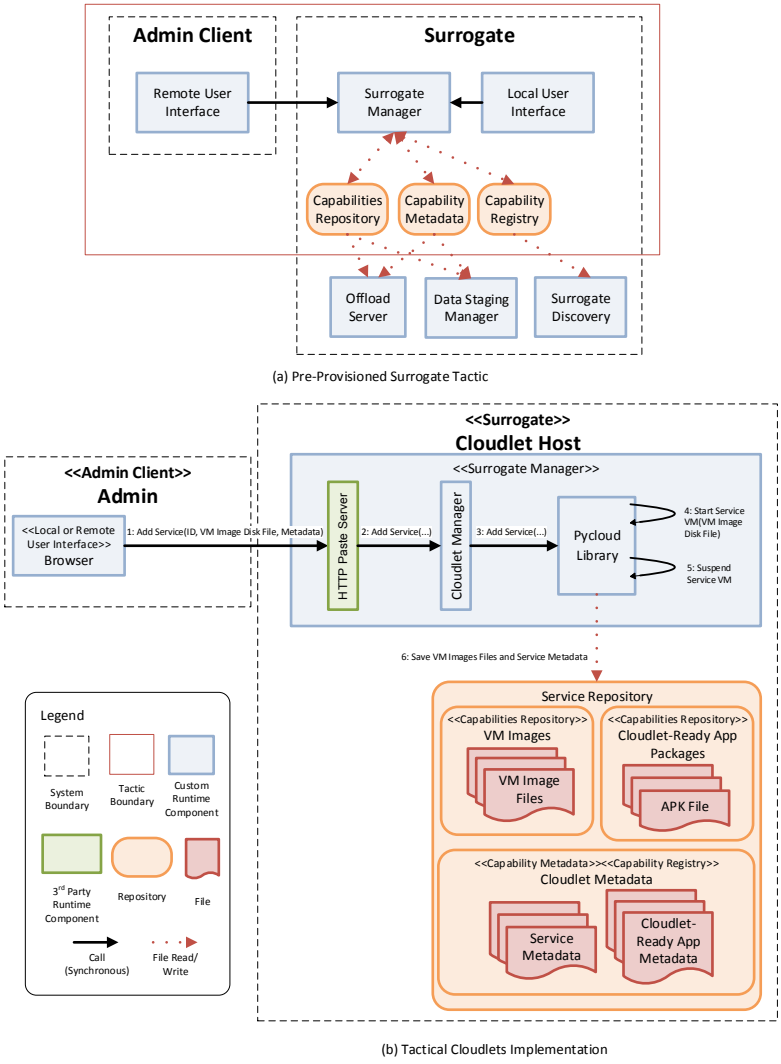

Figure 4.3: Tactical Cloudlets Implementation of the Pre-Provisioned Surrogate Tactic

1-3. The Admin client requests to add a new service *Service ID* to a cloudlet.

4. In order to provide a faster startup time for when service capabilities are requested, the Pycloud Library first starts the Service VM from the given VM Image Disk File.

5. The Pycloud Library then suspends the Service VM, which generates a VM Image Memory File. The faster startup time is because the Service VM will be started from a suspended state instead of a cold state.

6. Both the VM image disk and memory file are saved as VM Image Files in the Service Repository along with Service Metadata.

This same general process is followed when adding a Cloudlet-Ready App to the Service Repository. Cloudlet-Ready Apps are linked to services by *Service ID*. At runtime, the Cloudlet-Ready App uses the *Service ID* provided in steps 1-3 to start the computation offload process.

4.3.4.3 Surrogate Broadcast

In the Surrogate Broadcast tactic (Section 3.2.4.3) surrogates advertise their availability and selected metadata to mobile devices for discovery, as shown in Figure 4.4(a). In summary, the Broadcast Component on the surrogate broadcasts surrogate metadata for the Offload Client on the mobile client to discover.

The Surrogate Broadcast tactic can be identified in the Tactical Cloudlets architecture as shown in Figure 4.4(b), with numbers to indicate the sequence of operations. The component names in Figure 4.4(a) are used as stereotypes for the components in Figure 4.4(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included.

Cloudlet discovery in the Tactical Cloudlets system is based on the Avahi daemon[1] that implements Zeroconf (Zero Configuration Networking).[2] Avahi uses DNS Service Discovery (DNS-SD) along with Multicast DNS to enable a client to request a service without knowing the IP address of the server that provide the service. Cloudlet discovery by cloudlet-ready apps takes place as follows:

[1] http://avahi.org
[2] http://zeroconf.org

0. When the cloudlet starts, its Discovery Service joins a specific Cloudlet Multicast IP Address as a listener.
1. The Cloudlet-Ready App requests to offload service *Service ID*.
2. The Cloudlet Client Library sends a DNS-SD Query for cloudlet services (defined as a *_cloudlet._tcp* service) through Multicast DNS to the Cloudlet Multicast IP Address. The query reaches the Discovery Service of any cloudlets in the network through Multicast DNS. The Discovery Service replies with a DNS-SD Response indicating the IP address and port of the cloudlet server.

3-9. The Cloudlet Client Library sends a request for cloudlet metadata and the list of available services to each cloudlet that replied.

10. The Cloudlet Client Library selects the cloudlet that contains the service *Service ID* and has the lowest load, based on the assumption that it will have the fastest processing and response time. The architecture enables other algorithms to be plugged in.
11. The Cloudlet Client Library starts the computation offload process (Section 4.3.4.1) with the selected cloudlet.

4.3.4.4 Just-in-Time Containers

The Just-in-Time Containers tactic (Section 3.3.3.1) 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 the offload request is completed, as shown in Figure 4.5(a).

In the Tactical Cloudlets system, as shown in Figure 4.2(b), the computation offload process presented in Section 4.3.4.1, a QEMU-KVM Instance for a Service VM is only created upon an offload request.

In addition, to promote greater scalability and elasticity, when adding a service to a cloudlet (Section 4.3.4.2), one of the elements of the Service Metadata is whether the service will be shared or non-shared. A non-shared service will start a separate instance with every request. However, a shared service will only start an instance for its first request. All other requests will share the same instance. A counter of active users for the service is maintained as Service Metadata. This means that Step 6 in Figure 4.2 only takes place if the service is non-shared, or if it is the first request for a shared service.

The final step in the computation offload process presented in Section 4.3.4.1 is that the Cloudlet-Ready App starts the interaction with the Service VM. To implement the Just-in-Time Containers tactic, when the Cloudlet-Ready App is closed, the operations shown in Figure 4.5(b) take place. The

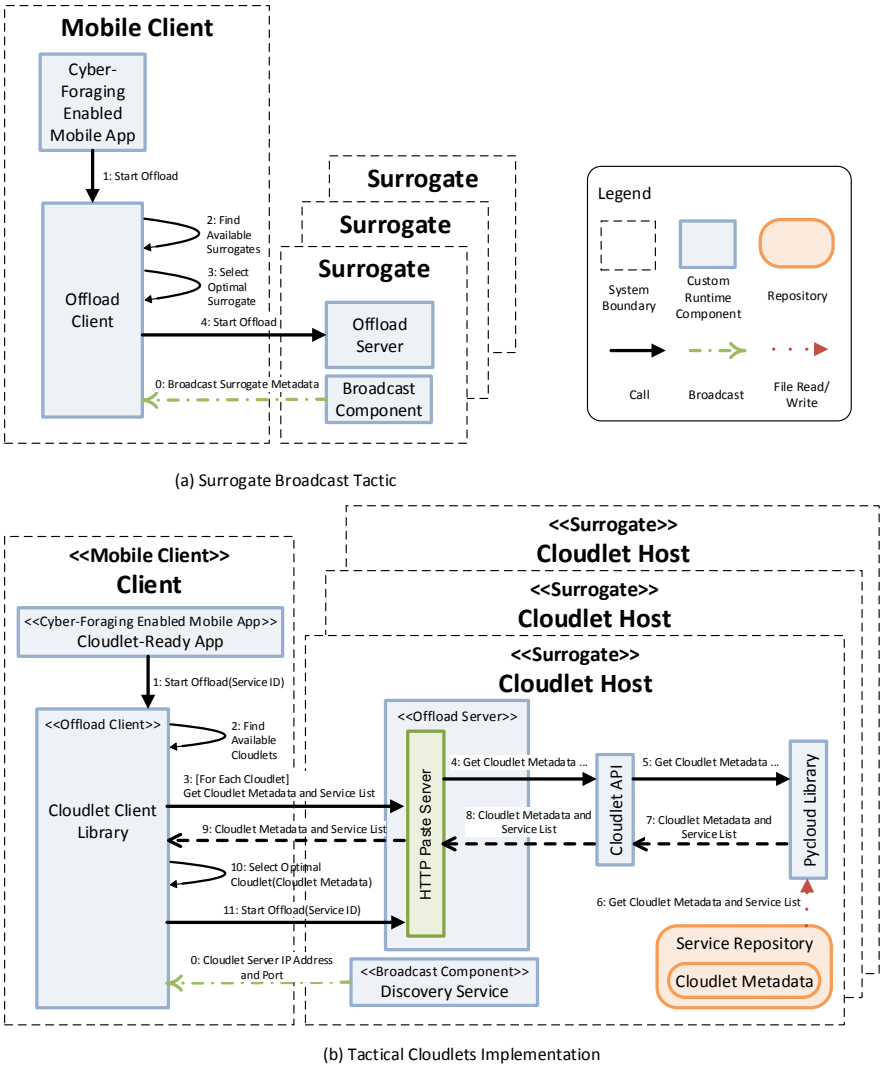

Figure 4.4: Tactical Cloudlets Implementation of the Surrogate Broadcast Tactic

component names in Figure 4.5(a) are used as stereotypes for the components in Figure 4.5(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included.

1-4. The Cloudlet-Ready App requests to stop service *Service ID*.

5. If the service is non-shared or the number of active users for the service is one (i.e., last active user), the Pycloud Library stops the instance of the service *Service ID*.

6. Service Metadata and Service VM Metadata are updated to indicate that the service has stopped and/or the number of active users for the shared service is one less.

4.3.5 Analysis

4.3.5.1 Mapping between Tactics and Requirements

The review of the Tactical Cloudlets architecture resulted in the identification of four architectural tactics for cyber-foraging. The mapping between the identified tactics and the Tactical Cloudlets functional and non-functional requirements is shown in Table 4.2.

Table 4.2: Mapping of Architectural Tactics to Functional and Non-Functional Requirements

Tactic	**FR1: Offload**	**FR2: Discovery**	**FR3: Disconnected Ops.**	**FR4: Separate Deployment**	**FR5: Selection**	**FR6: Management**	**FR7: Migration**	**NFR1: Energy Efficiency**	**NFR2: Scalability**	**NFR3: Ease of Deployment**
Computation Offload	X							X		
Pre-Provisioned Surrogate			X	X						
Surrogate Broadcast		X			X					
Just-in-Time Containers								X	X	

The **Computation Offload** tactic supports the requirement to offload

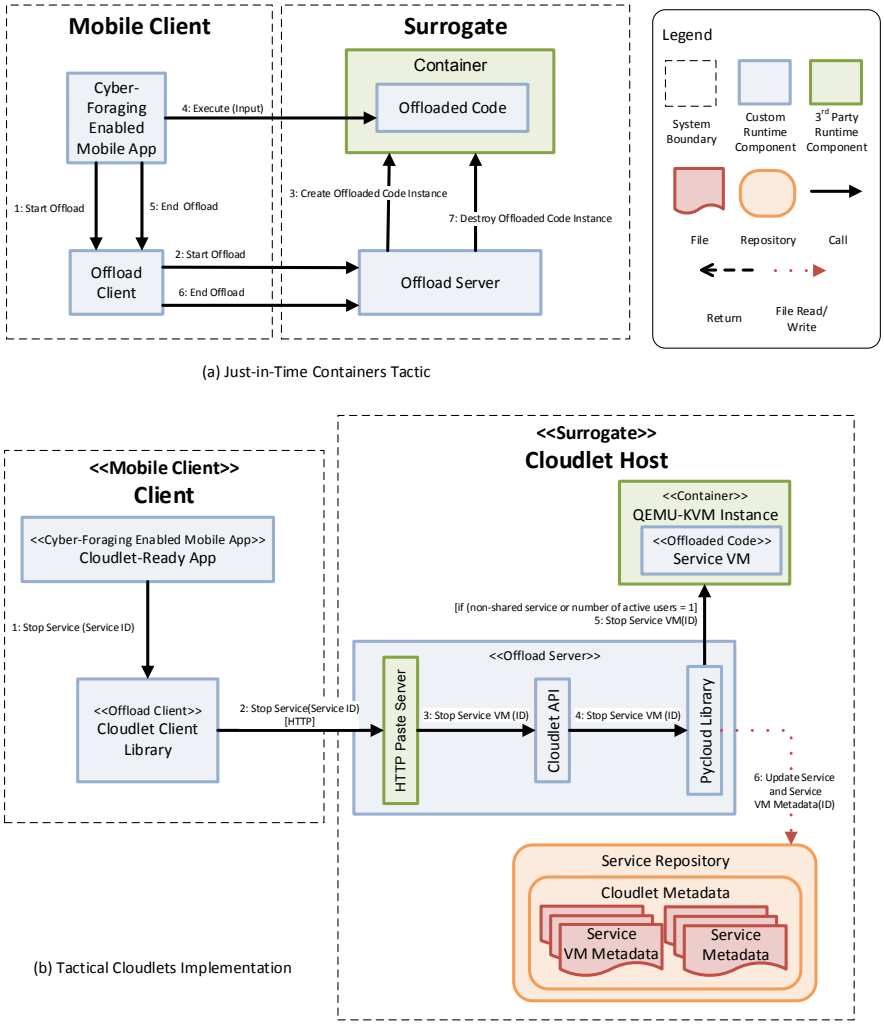

Figure 4.5: Tactical Cloudlets Implementation of the Just-in-Time Containers Tactic

expensive computation to nearby surrogates (FR1) as well as the energy efficiency requirement (NFR1). The developers of the tactical cloudlets sys-

tem split applications into a very thin client (Cloudlet-Ready App) and a very computation-intensive server (Service VM) such that energy efficiency is reached on the mobile device. The mapping between the tactic and the Tactical Cloudlet implementation in Figure 4.2 shows two differences:

1. The Tactical Cloudlets system does not use an external *App Metadata* file in the offload process. This is because the only metadata that is required is the *Service ID* which is hard-coded in the *Cloudlet-Ready App*. An improvement for a future version of the tactic is to mark the *App Metadata* component as optional.

2. The Tactical Cloudlets system has an additional *Service Repository* component from which offloaded code is fetched and then started as a *Service VM*. This additional step would be required of any system that implements the Computation Offload tactic together with the Pre-Provisioned Surrogate tactic, as is the case of the Tactical Cloudlets system (Section 4.3.4.2). An additional improvement for the catalog would be to include variations of the Computation Offload tactic when used with the different surrogate provisioning tactics.

The **Pre-Provisioned Surrogate** tactic supports the requirement for disconnected operations (FR3) because cloudlets are pre-provisioned with capabilities that are needed for a mission. In addition, because cloudlets are also pre-provisioned with the apps to use the capabilities, the tactic also supports the requirement to enable mobile devices to be provisioned in the field (FR4). The mapping between the tactic and the Tactical Cloudlet implementation is complete, as shown in Figure 4.3.

The **Surrogate Broadcast** tactic supports the requirement for cloudlet discovery (FR2) as well as the requirement for optimal cloudlet selection when more than one cloudlet is available (FR5). The mapping between the tactic and the Tactical Cloudlet implementation in Figure 4.4. shows two differences:

1. The cloudlet selection process is a two-step process in which the *Cloudlet Server IP Address and Port* broadcast by the *Broadcast Component* (Step 0) is used to query each cloudlet for capabilities (Step 3). The reason for this is that the Zeroconf protocol used by the Tactical Cloudlets implementation has a size limitation for broadcast information. While not a gap in the tactic itself, what this shows is that technology selection can introduce variations in the implementation of a tactic. An improvement for the catalog would be to include variations of the Surrogate Broadcast tactic when used with different technologies (or known limitations of technologies).

2. For this same reason, the *Surrogate Repository* is added to the implementation of the tactic. The cloudlet metadata and service list is obtained from the repository when the cloudlet is queried for its capabilities. This component would be part of the variation introduced by the broadcast protocol size limitation.

The **Just-in-Time Containers** tactic supports the requirement for capabilities to only be running when they are being used in order to promote scalability and elasticity (NFR2). The mapping between the tactic and the Tactical Cloudlet implementation in Figure 4.5 shows two differences:

1. The Tactical Cloudlets system introduces the concept of shared and non-shared capabilities, which is not specified in the original tactic. This is why the container is destroyed only if is it is a non-shared capability or the number of active users is one (i.e., only active user of the capability). An improvement for the catalog would be to include a variation of the Just-in-Time Containers tactic to support shared and non-shared capabilities.

2. For the same reason, the *Surrogate Repository* is added to the implementation of the tactic. *Service Metadata* and *Service VM Metadata* needs to be updated based on the results of the request to end the offload request. This component would be part of the variation introduced by the support for shared/non-shared capabilities.

Although not stated as a benefit of the tactic in Section 3.3.3.1, and not stated as a requirement for the system in Section 4.3.2, the Just-in-Time Containers tactic also supports energy efficiency on the cloudlet, which is critical in tactical environments where access to power might not always be available.

The requirement to provide a form of management console for a cloudlet admin to use (FR6) does not map to any of the tactics, as shown in Table 4.2. This fact is expected as it relates to one of the findings from the SLR that states a lack of focus on system-level concerns that is required when moving from experimental prototypes to operational systems (Section 2.6). One of these concerns is management of deployed capabilities. Related to this fact, there is not a tactic in the catalog that maps to ease of deployment and re-deployment (NFR3). However, in the Tactical Cloudlets system, the Admin component that implements the Admin Client in the Pre-Provisioned Surrogates tactic (Figure 4.3) is a lightweight, web-based interface that enables cloudlet management and easy deployment and redeployment of capabilities (FR6 and NFR3). It provides the following functionality:

- Service VM creation, edit and deletion
- Service VM import and export
- Service VM Instance start, stop and migration
- Cloudlet-Ready App repository (i.e., app store)

The extension of the catalog with tactics for ease of deployment and management would be useful for moving from experimental prototypes to operational cyber-foraging systems.

The requirement to be able to migrate capabilities between cloudlets when requested (FR7) does not map directly to any of the tactics either. However, the functionality in the Pycloud Library that enables this migration is very similar to that explained in the Eager Migration tactic once the monitoring component detects that the connection between the mobile device and the cloudlet is deteriorating (Section 3.3.2.5). The Admin component of the tactical cloudlets system that implements the Admin Client in the Pre-Provisioned Surrogates tactic (Figure 4.3) also contains functionality to manually migrate a Service VM Instance to another connected cloudlet. An improvement for the catalog would be to include a variation of the Eager Migration tactic to support manual migration.

To determine if the tactics meet their intended functional and non-functional requirements, the developers conducted extensive system testing and collected data to support their design and implementation decisions. In addition to successful test results, data collected included cloudlet provisioning time, energy consumption on the mobile device, payload size and response time. All implementation details and supporting data are available in several publications [32][76][77][78].

4.3.5.2 Discussion of Tactics for System Enhancements

The development team was presented with the complete list of architectural tactics for cyber-foraging so they could confirm the identified tactics. In addition, the team was asked what tactics not implemented in the system would be useful to address some of the characteristics of tactical environments. The team identified the following architectural tactics:

Eager Migration (Section 3.3.2.5) and Surrogate Load Balancing (Section 3.3.3.3): As stated in Section 4.3.5.1, the migration of capabilities in the tactical cloudlets system is manual. Adding capabilities for automated migration could enable load balancing, similar to what is done in cloud data

centers for resource optimization, or to enable migration to a more powerful or nearby cloudlet to improve response time and provide continued operations.

Trusted Surrogates (Section 3.3.4.1): The tactical cloudlets system relies on the underlying network to provide the secure communication between the mobile device and the cloudlet. While this may be enough in some scenarios, it is not enough for many military scenarios. A common solution for establishing trust between two nodes is to use a third-party online trusted authority that validates the credentials of the requester, similar to what happens in the Trusted Surrogate tactic. However, the characteristics of tactical environments do not consistently provide access to that third-party authority because tactical cloudlets operate in what is known as DIL environments (disconnected, interrupted, low bandwidth). The team is currently developing a trusted identity mechanism for use in disconnected environments that does not rely on a third party for validation of credentials. Instead, it generates server and device credentials in the field and relies on out-of-band channels for transfer of credentials to the device and therefore generates trust between the mobile device and the cloudlet.

Cached Results (Section 3.3.2.3): The system has been tested and demonstrated with applications that take between 8 and 12 seconds to receive a response, such as speech recognition, face recognition, and object recognition [76]. However, there are many other applications, such as sample analysis or data analysis, that could take a much longer time to respond. Given that tactical cloudlets operate in DIL environments, it is likely that a mobile device could lose connectivity to the cloudlet while the cloudlet is executing the offloaded operation. If this happens in the current tactical cloudlets system the mobile device is simply informed that connectivity has been lost. Implementing the Cached Results tactic would enable the cloudlet to cache the results of the offloaded operation until the mobile device regains connectivity.

Pre Fetching (Section 3.2.2.1) and Out-Bound Pre-Processing (Section 3.2.2.3): Even though the tactical cloudlets system was built with computation offload in mind, the reality is that many applications useful in tactical environments rely on data to provide better capabilities and would benefit from data staging tactics. For example, face recognition can work with a pre-loaded face database, but a better capability would be for that database to synchronize with a central database as windows of connectivity become available. The Pre-Fetching tactic would enable the system to download faces from a central database for people in the area of the cloudlet location to update the local database as connectivity becomes available. The Out-Bound Pre-Processing tactic would enable faces collected in the field to be checked for quality and duplication, and pre-processed before sending to the central

database during moments of connectivity.

4.3.5.3 Findings

The goal of the case study was to discover (1) which of the architectural tactics for cyber-foraging can be identified in the Tactical Cloudlets system, and (2) how do the implemented tactics support their intended functional and non-functional requirements. The context for the case study is the validation of the tactics in real systems.

The analysis of the Tactical Cloudlets system identified four architectural tactics for cyber-foraging. As a starting point, tactics were identified to satisfy the main functional requirements for a cyber-foraging system presented in Section 3.2:

- Pre-Provisioned Surrogate was used for surrogate provisioning
- Surrogate Broadcast was used for surrogate discovery
- Computation Offload was used to implement computation offload capabilities

To meet the scalability non-functional requirement, the Just-in-Time Containers tactic was used to create containers for computation when they are needed, and destroyed when they are no longer needed.

However, there were some gaps in the identified tactics (Section 4.3.5.1) that create opportunities for improvement of the tactics catalog:

1. Tactics should differentiate between core and optional components and interactions. Each optional component/interaction should contain rationale for when it is necessary to include it in the implementation of the tactic.
2. As tactics are implemented in operational cyber-foraging systems it is likely that variations will arise. The Tactical Cloudlets system introduced several potential tactic variations:
    - Variations of the Computation Offload tactic based on the surrogate provisioning tactic selected for the system
    - A variation of the Just-in-Time Containers tactic to support shared and non-shared capabilities
    - A variation of the Eager Migration tactic to support manual migration

3. Technology selection can also lead to tactic variations. As tactics are implemented and evaluated in cyber-foraging systems, technology limitations and constraints may require the implementation of additional components or interactions between components. The Tactical Cloudlets system introduced a variation of the Surrogate Broadcast tactic due to limitations in broadcast message size.

4. There is great potential for extending the catalog with tactics to support system qualities necessary for moving from experimental prototypes to operational systems. The Tactical Cloudlets system showed the need for tactics to support Ease of Deployment and Manageability.

5. Even if tactics are targeted at promoting a particular system quality, the tactics may have an effect on other system qualities. As an example, the Just-in-Time Containers tactic is a tactic for scalability/elasticity but also promotes energy efficiency on the surrogate. Even though the secondary effect of the tactic is positive, it could also have been a negative effect. This observation makes decision models such as the ones that will be presented in Chapter 8 a valuable tool for software architects to understand the potential effects of their decisions.

6. Related to the previous point, energy efficiency in cyber-foraging systems is mainly targeted at energy savings on mobile devices because of battery limitations. However, the Tactical Cloudlets system showed the need for tactics to support energy efficiency on surrogates, especially if deployed in areas with power limitations.

The utility of the tactics was supported by the main developer for the Tactical Cloudlets system in the following statement: "Having a set of architectural tactics for cyber-foraging systems would help considerably when starting the design of a new system. Cyber-foraging software has very particular requirements, and it is not easy to know how to create the architecture for the overall system to properly satisfy the appropriate quality attributes. A set of tactics would be an invaluable guide to make decisions at this stage."

The identification and analysis of these tactics in the Tactical Cloudlets system therefore answers the research questions for the case study, and in combination with the utility statement from the main system developer, serves as a validation of the tactics in cyber-foraging systems for computation offload.

4.3.6 Threats to Validity

There are two main threats to the validity of the results of this case study. The first is related to *internal validity* because the data collection and analysis was conducted by a single researcher and therefore subjective interpretations might exist. To mitigate this threat, collected data was reviewed by system developers that confirmed that the data collected was an accurate representation of the system. The developers also confirmed that the identified tactics were indeed present in the system. The second threat is related to *external validity*, specifically whether the findings are generalizable given that the results are drawn from the analysis of a single system. To mitigate this threat we conducted two additional case studies that are reported in Chapters 5 and 6. In addition, the system developers were provided the full set of tactics and asked to identify tactics that could be used to enhance the current system. The developers identified several tactics and recognized the potential for the tactics to build a better system.

4.4 Conclusions

This chapter presented the results of the first of three case studies to validate the architectural tactics for cyber-foraging presented in Chapter 3, in the context of RQ2, which is to identify the architectural tactics that can be derived from the architectural design decisions identified by the SLR.

For this case study two research sub-questions were defined for an existing computation offload system.

1. Which of the architectural tactics for cyber-foraging can be identified in the Tactical Cloudlets system?

The analysis of the Tactical Cloudlets system resulted in the identification of four architectural tactics for computation offload, cloudlet provisioning, cloudlet discovery and scalability/elasticity. In addition, elements of the Pre-Provisioned Surrogate tactic were also used to meet cloudlet management and ease of deployment and re-deployment requirements.

Although based on the analysis of a single system, the results show that a subset of the architectural tactics was found in an existing cyber-foraging system for computation offload. In addition, several gaps were identified that show that there is great potential to further extend the tactics catalog as more operational cyber-foraging systems are developed and evaluated.

2. How do the implemented tactics support their intended functional and non-functional requirements?

System testing and data collection show that the implemented tactics meet their intended functional and non-functional requirements.

More importantly, the results of this case study show that there is potential for taking a tactics-driven approach to fulfill functional and non-functional requirements for cyber-foraging systems. As indicated by the developers of the Tactical Cloudlets system, a catalog of architectural tactics would have been useful not only to discover ways to implement system requirements, but also to identify aspects of the system that had not been considered. The next case study explores the same case study research questions in the context of a data staging system.

4.5 Acknowledgments

Very special thanks to Sebastián Echeverría and the rest of the development team at the Carnegie Mellon Software Engineering Institute for their invaluable support during the execution of this case study.

5

Case Study 2: GigaSight — Cyber-Foraging for Data Staging

This chapter addresses research question RQ2 and is the second of three case studies to validate the architectural tactics presented in Chapter 3. The goal of this case study is to discover the architectural design decisions in the existing implementation of the GigaSight data staging system for crowd-sourced video [113], and then verify the mapping of the design decisions to the identified architectural tactics for cyber-foraging.

5.1 Introduction

This chapter continues the validation of the architectural tactics for cyber-foraging presented in Chapter 3, in the context of research question RQ2: *What architectural tactics can be derived from the identified architectural design decisions?*

The goal of this second case study is to discover the architectural design decisions in the existing implementation of the GigaSight system developed by Ghent University, Aalto University, Intel Labs, and Carnegie Mellon University to support data staging of crowd-sourced video [113], and then verify the mapping of the architectural design decisions to the architectural tactics for cyber-foraging.

As in the case study in the previous chapter, we followed the guidelines for conducting case studies from [15] and [118]. Accordingly, the structure of the chapter follows the steps proposed in these guidelines. Section 5.2 presents the case study design, including research questions and procedures for data collection and analysis. Section 5.3 presents the results of the case study and

threats to validity. Section 5.4 concludes the chapter with a summary of the findings as well as their implications and limitations.

5.2 Case Study Design

5.2.1 Research Questions

Given the goal to discover architectural design decisions in the existing implementation of the GigaSight system, we defined the following research questions to be answered in the execution of the case study.

- Which of the architectural tactics for cyber-foraging can be identified in the GigaSight system?
- How do the implemented tactics support their intended functional and non-functional requirements?

5.2.2 Data Collection Procedure

Given that the research sub-questions identified for this case study are the same as for the Tactical Cloudlets system, the data collection procedure is the same (Section 4.2.2). The main difference between systems is that Tactical Cloudlets is targeted at computation offload while the GigaSight system is targeted at data staging.

The code for the GigaSight system that is analyzed as part of the data collection procedure is available at `https://github.com/cmusatyalab/GigaSight`.

5.2.3 Analysis Procedure

The analysis procedure is also equivalent to procedure defined for the Tactical Cloudlets system (Section 4.2.3).

5.3 Results

5.3.1 System Context

GigaSight is a cyber-foraging system targeted at continuous collection of crowd-sourced video from mobile devices and wearables [113]. Given the potentially-sensitive nature of video, GigaSight collects video on surrogates called cloudlets where privacy-sensitive information is automatically removed from the video

based on user-defined privacy settings related to time, location, and content — a process called denaturing. Denatured video is then indexed and resulting tags and metadata are uploaded to a cloud catalog where users can perform content-based searches on the total catalog of denatured videos.

Use cases for crowd-sourced video systems such as GigaSight include marketing and advertising; location of missing people, pets and things; creation of family vacation albums; public safety; and fraud detection [110].

5.3.2 System Requirements

The requirements of the GigaSight system can be divided into functional and non-functional requirements.

5.3.2.1 Functional Requirements

GigaSight needs to satisfy the following functional requirements.

- **FR1: Video capture**: The mobile device captures and stores video.
- **FR2: User-specified privacy settings**: Users are able to specify privacy settings based on location, time, and image content. These settings are used by the denaturing process to automatically remove privacy-sensitve content from videos.
- **FR3: Video upload to cloudlets**: When a cloudlet becomes available, the mobile device uploads captured video and privacy settings.
- **FR4: Offload of video denaturing and indexing processes**: The highly computation-intensive denaturing and indexing operations are executed on the cloudlet according to user-specified privacy settings.
- **FR5: Index upload to cloud catalog**: Video metadata and tags generated by the indexing process are uploaded from the cloudlet to a cloud catalog that can be queried by users.
- **FR6: User requests for denatured videos**: A user of the cloud catalog can request denatured videos from cloudlets.

5.3.2.2 Non-Functional Requirements

GigaSight needs to satisfy the following non-functional requirements.

- **NFR1: Energy Efficiency**: Energy consumption on the mobile device when offloading the computation-intensive denaturing and indexing operations should be less than energy consumed by executing them locally.
- **NFR2: Scalability**: One cloudlet should be able to process and store video from multiple users.
- **NFR3: Fault Tolerance**: If a cloudlet is not available for upload, the mobile device should be able to cache video until a cloudlet becomes available.
- **NFR4: Privacy**: Privacy-sensitive information should not be made available to users of the cloud catalog.

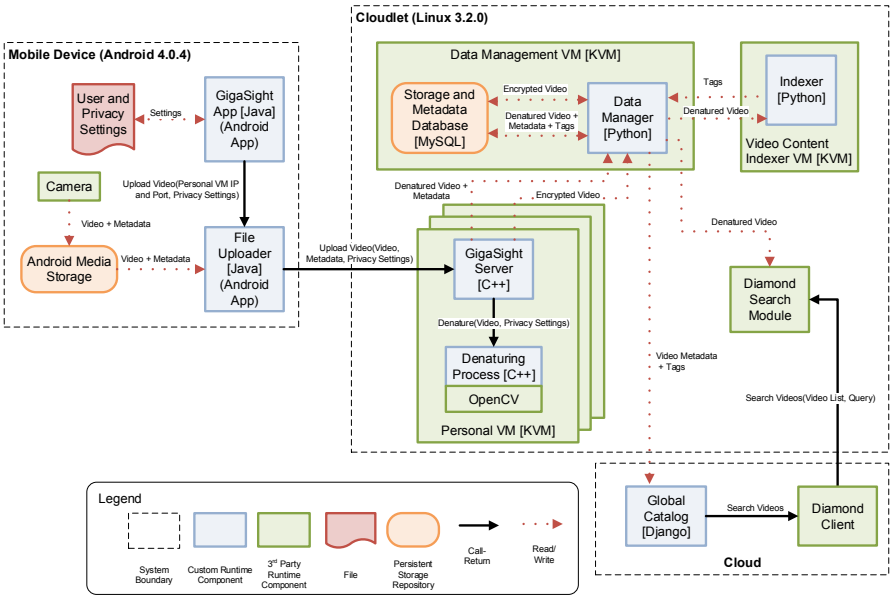

Figure 5.1: High-Level Architecture of the GigaSight System

5.3.3 System Architecture and Design

The as-is architecture for the GigaSight system is shown in Figure 5.1. The main elements of the architecture are:

- Mobile Device: The mobile device is an Android 4.0.4 device. It leverages the device's built-in camera for video capture.
  - GigaSight App: Performs all the user and privacy setting management. User settings include IP address and port of its Personal VM. Privacy settings include time filters, location filters and object-based filters. The object-based filters are currently limited to the faces present in the training set of the face recognition algorithms.
  - File Uploader: Connects to the user's Personal VM and uploads video files and metadata. Once files are successfully uploaded, these are removed from the mobile device to make space for more video content.
- Cloudlet: Cloudlets are data staging points for denatured video data en route to the cloud. Cloudlets in GigaSight are implemented as servers running Linux 3.2.0.
  - Personal VM: Each mobile device user is associated to a Personal VM that performs the customized denaturing for that user according to the user-defined privacy settings. The Denaturing Process that executes inside this VM is implemented using C++ and OpenCV 2.4.2[1] as a multi-step pipeline: video decoding, early-discard of frames based on metadata and sampling rate, content-based blurring, and video encoding. The output of the denaturing process is a low-frame-rate denatured video file. For additional privacy, an encrypted version of the original video is also created during the upload process. Both files are stored in the Data Management VM so that they are accessible to other VMs on the cloudlet.
  - Data Management VM: The Data Manager inside this VM handles all video and metadata storage and retrieval in the Storage and Metadata Database. It notifies the Indexer when new denatured video is available for indexing. In addition, each time the Indexer adds tags to the database, these are automatically synchronized with the Global Catalog running in the Cloud.
  - Video Content Indexer VM: The Indexer inside this VM is a background activity that extracts metadata about denatured videos (e.g., owner (anonymized), location of capture, start and end time

[1] `http://www.opencv.org`

of capture, cloudlet address where stored, and tags) and sends it to the Data Manager which in turn pushes this information to the Global Catalog in the Cloud. The metadata is also stored locally for use by search algorithms that could be implemented inside the Personal VM for personal use.

  - Diamond Search Module: The Diamond Search Module is a third-party component for interactive search of non-indexed data.[2]

- Cloud: Cloud-based data center that aggregates video metadata from a set of associated cloudlets.

  - Global Catalog: The Global Catalog is a web application implemented using Django[3] that stores and manages the metadata from denatured videos available on cloudlets. The front end to the application enables users to browse through the metadata and select videos of interest for viewing.

  - Diamond Client: Once a user selects videos of interest, the Diamond Client contacts the Diamond Server of each cloudlet that contains a video of interest to initiate content-based search.

5.3.4 Mapping of Architectural Design Decisions to Architectural Tactics

The mapping of functional and non-functional requirements to components of the architecture is shown in Table 5.1.

Architectural design decisions implemented in the GigaSight system were mapped to functional and non-functional architectural tactics described in Chapter 3. The process that was used to perform the mapping was the same as described in Section 4.3.4 for the Tactical Cloudlets system.

The following subsections describe how each selected tactic was identified and implemented in the GigaSight system.

5.3.4.1 Out-Bound Pre-Processing

In the Out-Bound Pre-Processing tactic (Section 3.2.2.3) surrogates collect data from mobile devices and pre-process the data – clean, filter, summarize, or merge — such that the data that is sent on to the enterprise cloud is ready for consumption and serves an immediate need, as shown in Figure 5.2(a).

[2] `http://diamond.cs.cmu.edu`
[3] `http://www.djangoproject.com`

Table 5.1: Mapping of Functional and Non-Functional Requirements to the Architecture of the GigaSight System

Component	FR1: Video Capture	FR2: Privacy Settings	FR3: Video Upload to Cloudlet	FR4: Computation Offload to Cloudlet	FR5: Video Index Upload to Cloud	FR6: User Requests	NFR1: Energy Efficiency	NFR2: Scalability	NFR3: Fault Tolerance	NFR4: Privacy
Camera	X									
Android Media Storage	X								X	
GigaSight App		X	X						X	X
File Uploader			X				X		X	
Personal VM								X		X
GigaSight Server			X							
Denaturing Process				X			X			X
Data Manager (+ Storage)					X	X				X
Video Context Indexer VM								X		
Indexer				X			X			
Diamond Search Module						X				
Global Catalog					X	X				
Diamond Client						X				

In summary, data collected from mobile devices is stored in a Cache on the Surrogate and then pre-processed by a Data Processing Application(s) before it is sent to a Cloud Data Repository.

The Out-Bound Pre-Processing tactic can be identified in the GigaSight architecture as shown in Figure 5.2(b), with numbers to indicate the sequence of operations. The component names in Figure 5.2(a) are used as stereotypes for the components in Figure 5.2(b) to indicate the mapping between compo-

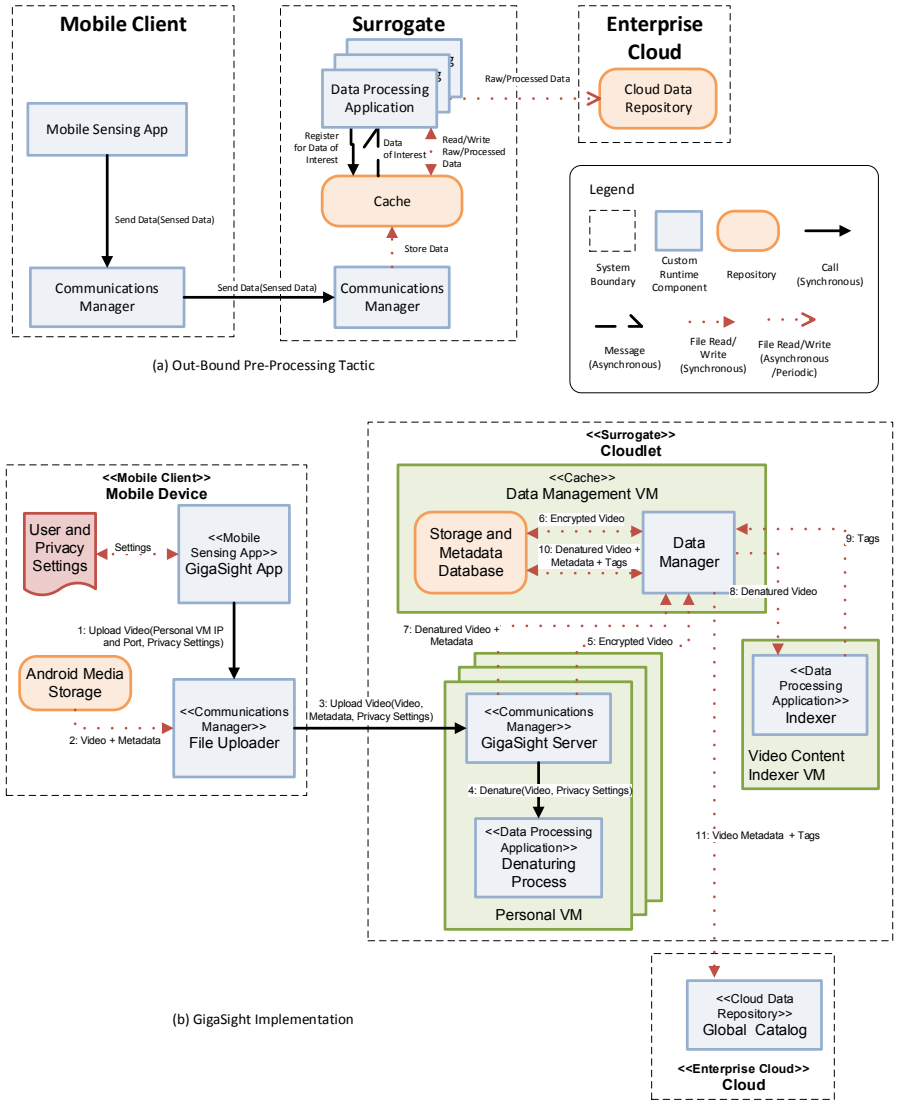

Figure 5.2: GigaSight Implementation of the Out-Bound Pre-Processing Tactic

nents. Only the components that are relevant to the tactic are included. The out-bound pre-processing takes place as follows:

1-3 *GigaSight App* uploads stored video and metadata to the *Personal VM* identified by Personal VM IP Address and Port using the *File Uploader*.

4 The *GigaSight Server* receives the video, metadata and privacy settings for the user and sends these to the *Denaturing Process* for denaturing according to the user's privacy settings.

5-6 The *GigaSight Server* encrypts the original video and sends it to the *Data Manager* for storage.

7 The *GigaSight Server* sends the denatured video and metadata to the *Data Manager* for storage and indexing.

8-9 The *Data Manager* sends the denatured video to the *Indexer* for indexing, which returns the set of tags for elements identified in the video.

10 The *Data Manager* stores the denatured video, metadata and tags.

11 The *Data Manager* sends the video metadata and tags to the *Global Catalog* in the *Cloud*.

5.3.4.2 Pre-Provisioned Surrogate

In the Pre-Provisioned Surrogate tactic (Section 3.2.3.1) surrogates are provisioned before their deployment with the capabilities that are offloaded by mobile clients, as shown in Figure 5.3(a). In summary, an Admin Client enables surrogate administrators to add capabilities to a Surrogate via a Surrogate Manager.

Some elements of the Pre-Provisioned Surrogate tactic can be identified in the GigaSight architecture as shown in Figure 5.3(b). The component names in Figure 5.3(a) are used as stereotypes for the components in Figure 5.3(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included.

In the GigaSight system all data processing capabilities are provisioned on the cloudlet before deployment. However, this is a manual process. There is not the equivalent of a Surrogate Manager component to help with the provisioning process as shown in the tactic. In addition, because capabilities are not advertised, but rather each mobile device stores the IP Address and Port of its Personal VM as part of the User Settings, there is not the equivalent of

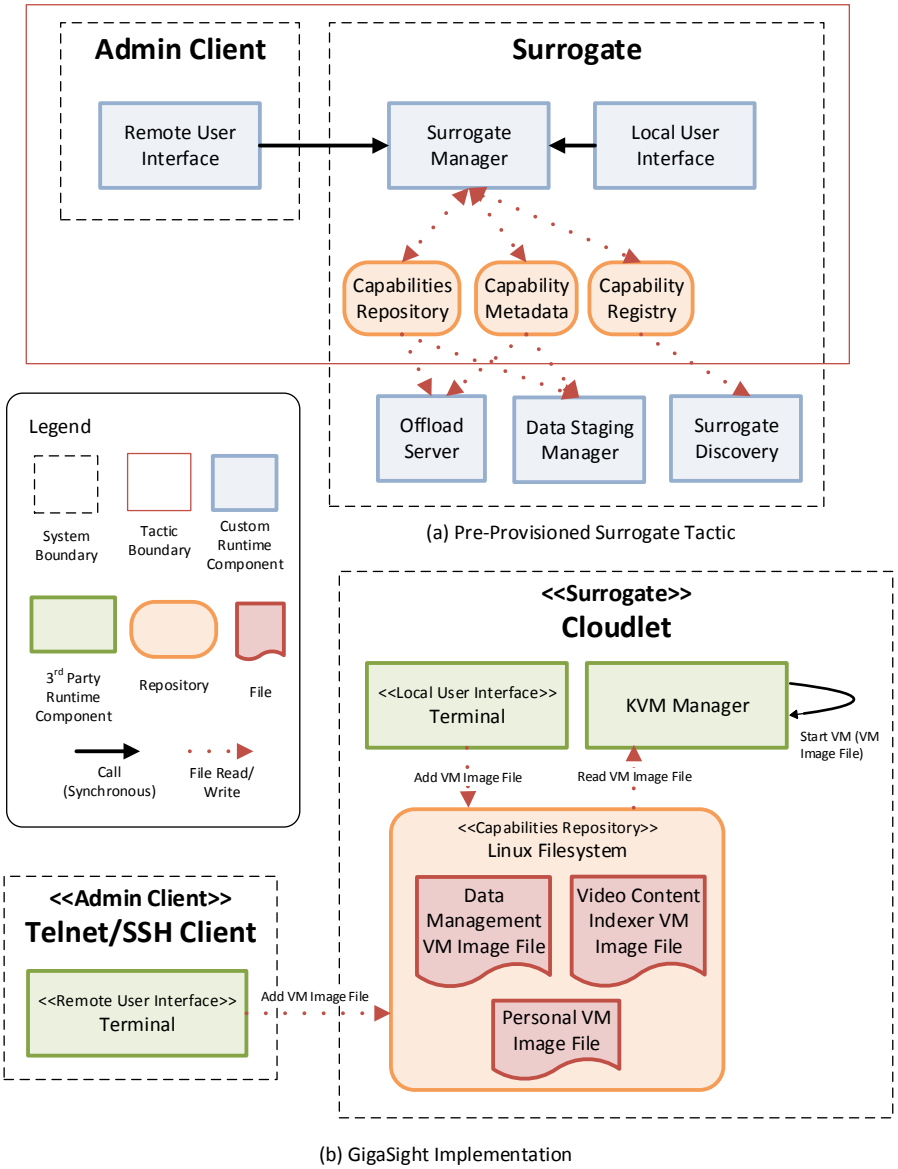

Figure 5.3: GigaSight Implementation of the Pre-Provisioned Surrogate Tactic

a Capability Metadata component nor the equivalent of a Capability Registry component.

As depicted in Figure 5.3(b), prior to deployment the Terminal program that comes with the Linux distribution is executed locally or remotely to copy the Data Management VM Image File, the Video Content Indexer VM Image File, and the Personal VM Image File that contains the denaturing capabilities into the Linux Filesystem. The KVM Manager program that also comes with the Linux distribution is initially used to start one instance of the Data Management VM and one or more instances of the Video Content Indexer VM.[4] A Personal VM is then started for each mobile device that wants to use the GigaSight system for video offload. After starting the Personal VM the mobile device user is provided with its IP Address and Port, which needs to be added to the User Settings using the GigaSight app shown in Figure 5.1.

5.3.4.3 Local Surrogate Directory

In the Local Surrogate Directory tactic (Section 3.2.4.1), mobile devices maintain a list of surrogates with their network addresses or URLs, in addition to any information that can help the mobile device to select the best offload target in case more than one is available, as shown in Figure 5.4(a). In summary, the list of surrogates is retrieved from the Surrogate Directory before starting the offload process and an optimal surrogate is selected for offload.

The process is much simpler in the GigaSight system, as shown in Figure 5.4(b). There is not a cloudlet selection process because every mobile device is assigned a Personal VM on a single Cloudlet. The location of a cloudlet for data upload takes place as follows:

0 As indicated in Section 5.3.4.2, when a *Personal VM* is started for a *Mobile Device*, the *GigaSight App* in its role as *Surrogate Directory UI* is used to save the Personal VM IP Address and Port to the *User and Privacy Settings* file.

1 When the video upload process is started by the *GigaSight App* in its role of *Offload Client*, it reads the Personal VM IP Address and Port from the *User and Privacy Settings* file.

2-3 Video and Metadata are uploaded to the *Personal VM* at the provided IP Address and Port.

[4]Deployment of more than one Video Context Indexer VM is an architectural design decision for scalability, as discussed in Section 5.3.5

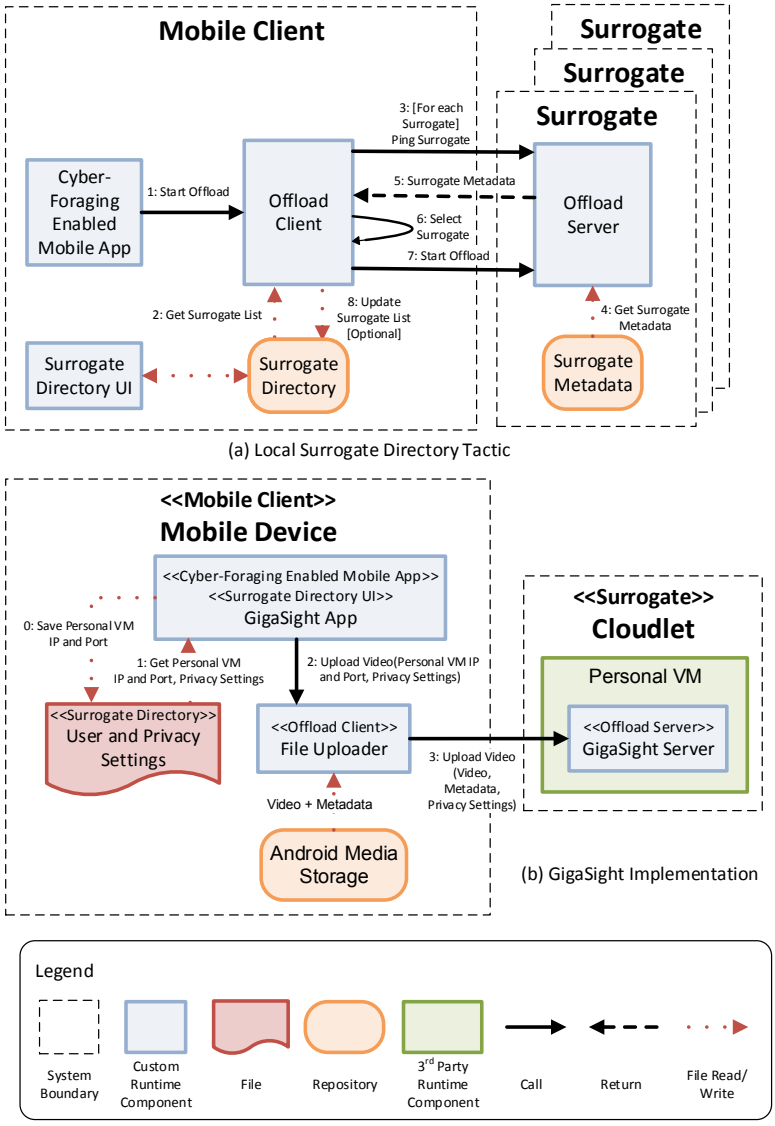

Figure 5.4: GigaSight Implementation of the Local Surrogate Directory Tactic

5.3.4.4 Client-Side Data Caching

The Client-Side Data Caching tactic is a variation of the Cached Results tactic (Section 3.3.2.3). Data collected by a mobile client is cached on the mobile device and sent to the surrogate upon connection or re-connection, as shown in Figure 5.5(a).

The Client-Side Data Caching tactic can be identified in the GigaSight architecture as shown in Figure 5.5(b), with numbers to indicate the sequence of operations. The only difference between the implementation and the tactic is that the sensed data (video + metadata) is saved in the cache upon capture, instead of upon disconnection. The component names in Figure 5.5(a) are used as stereotypes for the components in Figure 5.5(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included. The client-side data caching takes place as follows:

0 Video captured using the *Camera* on the *Mobile Device* is stored in the *Android Media Storage* along with metadata such as location.

1-4 The *GigaSight App* tries to upload video to its *Personal VM* to be encrypted, denatured, and stored in the *Data Management VM* on the *Cloudlet*.

5-7 Only if the operation is successful, the just uploaded video is deleted from the *Android Media Storage* to make room for new video. If it is not successful the user gets an error message and is asked to try the upload at a later time.

5.3.5 Analysis

5.3.5.1 Mapping between Tactics and Requirements

The review of the GigaSight architecture resulted in the identification of four architectural tactics for cyber-foraging. The mapping between the identified tactics and the GigaSight functional and non-functional requirements is shown in Table 5.2.

The **Out-Bound Pre-Processing** tactic supports all of the functional requirements because it maps well to sensing applications such as GigaSight. Because denaturing and indexing are extremely computation-intensive activities that are executed on the cloudlet and not on the mobile device (as demonstrated via experimentation in [113]), the tactic also supports energy efficiency (NFR1). Finally, the pre-processing that occurs on the cloudlet in the Denaturing Process, supports the privacy requirement (NFR4). Because the Personal VM is assigned to one and only one mobile device, there is a guarantee

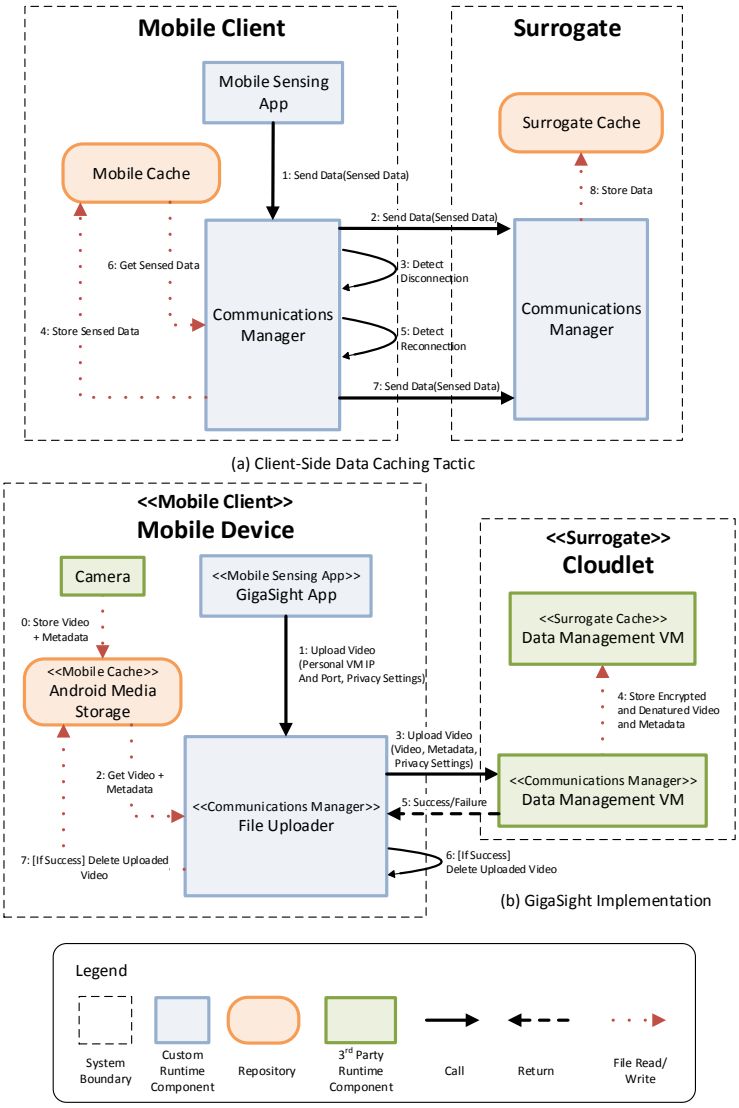

Figure 5.5: GigaSight Implementation of the Client-Side Data Caching Tactic

Table 5.2: Mapping of Functional and Non-Functional Requirements to the Architecture of the GigaSight System

Component	FR1: Video Capture	FR2: Privacy Settings	FR3: Video Upload to Cloudlet	FR4: Computation Offload to Cloudlet	FR5: Video Index Upload to Cloud	FR6: User Requests	NFR1: Energy Efficiency	NFR2: Scalability	NFR3: Fault Tolerance	NFR4: Privacy
Out-Bound Pre-Processing	X	X	X	X	X	X	X			X
Pre-Provisioned Surrogate				X	X		X	X		X
Local Surrogate Directory								X		X
Client Side Data Caching	X		X				X		X	

that the raw video is only processed by the Personal VM. Because the video is encrypted before it is stored in the Data Management VM, access to the raw video would only be possible via the Personal VM which is the only system component that knows the encryption key. The mapping between the tactic and the GigaSight implementation in Figure 5.2 shows three main differences:

1. The GigaSight system has an additional *User and Privacy Settings* file that is read by the *GigaSight App* to obtain settings for uploading video to the cloudlet. This is reasonable and equivalent to the *App Metadata* component in the Computation Offload tactic (Section 3.2.1), which is where the setttings for the offload process are stored. An improvement for a future version of the Out-Bound Pre-Processing tactic is to include a more general *Settings* component that performs this role and mark it as optional.

2. The GigaSight system has an additional *Android Media Storage* component because video and metadata sent to the cloudlet are read from internal storage. This component makes sense for a system that stores

data before sending it to the surrogate, as opposed to sending data as it is received. An improvement for a future version of the tactic is to include a more general *Data Storage* component that performs this role and mark it as optional.

3. The GigaSight system has an instance of the *GigaSight Server (Communications Manager)* for each user, as opposed to a single instance. This is done to support the privacy requirement and will be discussed shortly when the mapping to the Pre-Provisioned Surrogate tactic is analyzed.

The **Pre-Provisioned Surrogate** tactic supports computation offload to the cloudlet (FR4) and video index upload to the cloud (FR5). These are capabilities that are pre-provisioned on the cloudlet in the form of VMs. Because the capabilities already exist on the cloudlet, there is no need to transfer any extra computation from the mobile device or the cloud, leading to energy efficiency (NFR1). The mapping between the tactic and the GigaSight implementation in Figure 5.3 shows two main differences:

1. The GigaSight system does not have the equivalent of the *Surrogate Manager*, as discussed in Section 5.3.4.2. Adding this component to the system would promote ease of deployment and manageability as GigaSight moves from a prototype to an operational system.

2. As also discussed in Section 5.3.4.2, the GigaSight system does not have the equivalent of the *Capabilities Metadata* and *Capability Registry* components because (1) capabilities are not advertised and (2) capabilities on all surrogates are the same. Therefore, an improvement for a future version of the tactic would be to mark these two components as optional.

Even though having a pre-provisioned surrogate by itself does not support scalability (NFR2), in this particular system it does. Mobile devices are assigned a specific Personal VM on a cloudlet (by IP address and port) and therefore the number of mobile devices supported by a cloudlet can be controlled. Once a defined disk and memory threshold on a cloudlet has been reached, new mobile devices would need to be assigned to a different cloudlet. In essence, a pre-provisioned surrogate has more control over its load.

The **Local Surrogate Directory** tactic supports scalability (NFR2) because a cloudlet can support multiple users by instantiating multiple instances of a Personal VM. It also supports privacy (NFR4) because the Personal VM is the cloud-based counterpart of the mobile device: an entity that the user trusts to store personal content, but with much more computational and storage resources [113]. The mapping between the tactic and the GigaSight implementation in Figure 5.4 shows two main differences:

1. The GigaSight system has an additional *Android Media Storage* component because video and metadata sent to the cloudlet are read from internal storage. This is not a gap in this particular tactic but an area for improvement for the Out-Bound Pre-Processing tactic, as discussed earlier.

2. The GigaSight system does not have a *Surrogate Metadata* component because there is not a cloudlet selection process. An improvement for a future version of the tactic would be to mark this component as optional, as well as the surrogate selection process (Steps 3-6 in Figure 5.4(a)).

The **Client-Side Data Caching** tactic supports video capture and upload to a cloudlet (FR1 and FR2). It supports energy efficiency (NFR1) because uploading longer segments (instead of uploading as video is captured) requires the device to wake up less frequently from the sleep state, while the total number of bytes transmitted remains constant [113]. Finally, it supports fault tolerance (NFR3) because video is not uploaded until a cloudlet is available, and is not deleted from the device until the cloudlet confirms the upload. The mapping between the tactic and the GigaSight implementation in Figure 5.5 shows two differences:

1. The GigaSight system contains a *Camera* component as the data source. An improvement for a future version of the tactic would be to include a more general *Data Source* core component to indicate the source of the data that is stored in the *Mobile Cache*.

2. As indicated in Section 5.3.4.4, sensed data is saved in the cache upon capture, instead of upon disconnection. This difference could be added as a variation of the Client-Side Data Caching tactic.

It is important to note that some non-functional requirements in GigaSight are supported by specific technology selection as opposed to the use of tactics. The use of VMs as containers for data and computation on the cloudlet promotes scalability and elasticity because of the ease for container creation, migration, and destruction provided by VM management tools. For example, additional instances of the content indexer can be instantiated on one or more cloudlets to handle increasing loads. A Personal VM can also be easily moved to another cloudlet as long as the device is informed of its new address. The use of VMs as containers also promotes privacy in the system because Personal VMs are single-user and VM isolation is a well-known property of VMs. Although there are potential vulnerabilities and attacks, for the most part this property can be guaranteed [60].

To determine if the tactics meet their intended functional and non-functional requirements, the developers conducted extensive system testing and collected data to support their design and implementation decisions. In addition to successful test results, data collected included system throughput, cloudlet performance, algorithm accuracy, and energy consumption on the mobile device. All implementation details and supporting data are available in several publications [110][113].

5.3.5.2 Discussion of Tactics for System Enhancements

The two main GigaSight developers were presented with the complete list of architectural tactics for cyber-foraging so they could confirm the identified tactics. In addition, they were asked what tactics not implemented in the system would be useful to address some of the GigaSight requirements or desired features. The developers identified the following architectural tactics:

Surrogate Provisioning from the Mobile Device (Section 3.2.3.2) and Surrogate Provisioning from the Cloud (Section 3.2.3.3): While having cloudlets pre-provisioned with a user's Personal VM as presented in Section 5.3.4.2 is acceptable for a prototype implementation, a more flexible form of provisioning would be necessary to make the system production-ready. A hybrid strategy that combines provisioning from the cloud and the mobile device would work well for the GigaSight system. Generic Personal VMs could be provisioned from the cloud. For privacy reasons, the filters used in the denaturing process could be provisioned from the mobile device. For the GigaSight system, the latency drawback of provisioning from the cloud does not apply: only in very few cases, such as a lost child scenario, would it be important to have the indexed tags of the denatured videos rapidly available.

Resource-Adapted Computation (Section 3.3.1.3): This tactic could be explored in GigaSight to address content that is highly sensitive. For example, coarse-grained denaturing of entire frames with sensitive content could take place on the device before sending to the cloudlet. The denaturing process on the cloudlet is more fine-grained, where only parts of individual frames are denatured. The fine-grained denaturing process that takes place on the cloudlet is much more computation intensive than the coarse-grained one that would execute on the mobile device.

Just-in-Time Containers (Section 3.3.3.1): In the current GigaSight system, the Video Content Indexer VM is started at cloudlet startup and is always running. An alternative approach is to run the indexer only when needed (e.g., when there is a batch of N new videos available).

Surrogate Load Balancing (Section 3.3.3.3): Additional cloudlets

could be selected and started for denaturing and indexing video based on network connectivity and existing load on the cloudlet.

Trusted Surrogates (Section 3.3.4.1): A mechanism in which the mobile device could confirm the identity of the cloudlet would be crucial to implement because it would increase the user's trust in the GigaSight system for denaturing raw videos with potentially sensitive information.

As additional information, the GigaSight system is the basis for a new large-scale event participation platform for the wireless transmission of user-generated videos. The goal of the system is to boost audience involvement and immersion by, for example, integrating user-generated content into the event experience [27]. The system is in the field test phase (as of December 2015). Two tactics will be considered for resource optimization and therefore scalability of the system to deal with large amounts of users:

- **Resource-Adapted Computation (Section 3.3.1.3):** Some of the analytics that run on the cloudlets could run on the device itself at a lower scale to detect shakiness, blurred, or dark/bright video. This should help in only uploading higher-quality videos.
- **Right-Sized Containers (Section 3.3.3.2):** The moments when the load on the system will be high could be predicted based on the script of the event, i.e., some moments of the show are more interesting to capture than others. At these times, larger containers could be created to process larger amounts of captured video.

5.3.5.3 Findings

The goal of the case study was to discover (1) which of the architectural tactics for cyber-foraging can be identified in the GigaSight system, and (2) how do the implemented tactics support their intended functional and non-functional requirements. The context for the case study is the validation of the tactics in real systems.

The analysis of the GigaSight system identified four architectural tactics for cyber-foraging. Similar to the Tactical Cloudlets analysis, tactics were identified to satisfy the main functional requirements for a cyber-foraging system presented in Section 3.2:

- Pre-Provisioned Surrogate was used for surrogate provisioning
- Local Surrogate Directory was used for surrogate discovery

- Out-Bound Pre-Processing was used to implement data staging capabilities

In addition, the Client-Side Data Caching tactic was used to implement a fault tolerance non-functional requirement to store captured video until a surrogate is available so that it can continue functioning despite being disconnected from a surrogate.

However, there were some gaps in the identified tactics (Section 5.3.5.1) that create opportunities for improvement of the tactics catalog:

1. Consistent with the Tactical Cloudlets system (Section 4.3.5.3), tactics should differentiate between core and optional components and interactions. Each optional component/interaction should contain rationale for when it is necessary to include it in the implementation of the tactic.

2. Consistent with the Tactical Cloudlets system, even if tactics are targeted at promoting a particular system quality, the tactics may have an effect on other system qualities, further supporting the value of decision models such as the ones that will be presented in Chapter 8.

3. Consistent with the Tactical Cloudlets system, as tactics are implemented in operational cyber-foraging systems it is likely that variations will arise. The GigaSight system introduced a potential tactic variation of the Client-Side Data Caching tactic that always caches data, as opposed to only caching data when a surrogate is not found.

4. Functional and non-functional requirements in cyber-foraging systems can also be met by technology selection, rather than by the use of a particular tactic. In the GigaSight system, the use of VMs as containers had a positive effect on scalability/elasticity as well as privacy. Insights that are gained from the implementation and evaluation of cyber-foraging systems could be added as notes to the tactics to provide even greater value to software architects.

The utility of the tactics was supported by the main developer for the GigaSight system in the following statement: "It is helpful for developers to have some 'best practices' in software architecture for cyber foraging. Today, we already have many patterns (e.g., Gang of Four [43]), but these are very focused on object-orientation, rather than on taking into account the actual deployment. Having a reference list of tactics, plus possibly coding elements in the future, would, in my view, be very helpful in designing production-grade cyber-foraging applications. So far, cyber-foraging has not truly left the lab

prototype phase and typically good software design practices are second hand during this phase of the research. But with cloudlets, micro data centers, and edge clouds appearing everywhere, there will emerge a need from industry on this."

The identification and analysis of these tactics in the GigaSight system therefore answers the research questions for the case study, and, in combination with the utility statement from one of the system developers, serves as a validation of the tactics in cyber-foraging systems for data staging.

5.3.6 Threats to Validity

There are two main threats to validity of the results of this case study. The first is related to *internal validity* because the data collection and analysis was conducted by a single researcher and therefore subjective interpretations might exist. To mitigate this threat, collected data was reviewed by system developers that confirmed that the data collected was an accurate representation of the system. The developers also confirmed that the identified tactics were indeed present in the system. The second threat is related to *external validity*, specifically whether the findings are generalizable given that the results are drawn from the analysis of a single system. To mitigate this threat we conducted two additional case studies that are reported in Chapters 4 and 6. In addition, the system developers were provided with the full set of tactics and asked to identify tactics that could be used to enhance the current system. The developers identified several tactics and recognized the potential for the tactics to build a better system.

5.4 Conclusions

This chapter presented the results of the second of three case studies to validate the architectural tactics for cyber-foraging presented in Chapter 3, in the context of RQ2, which is to identify the architectural tactics that can be derived from the architectural design decisions identified by the SLR.

For this case study, as in the Tactical Cloudlets system, two research questions were defined for an existing data staging system:

1. Which of the architectural tactics for cyber-foraging can be identified in the GigaSight system?

The analysis of the GigaSight system resulted in the identification of four architectural tactics for data staging, cloudlet provisioning, cloudlet discovery and fault tolerance. In addition, elements of these tactics were also used to

meet energy efficiency requirements as well as privacy requirements. Scalability requirements were met by a combination of tactics plus the selection of virtual machines as containers for data processing applications.

Although based on the analysis of a single system, the results show that a subset of the architectural tactics was found in an existing cyber-foraging system for data staging. In addition, several gaps were identified that further show that there is great potential to further extend the tactics catalog as more operational cyber-foraging systems are developed and evaluated.

2. How do the implemented tactics support their intended functional and non-functional requirements?

System testing and data collection show that the implemented tactics meet their intended functional and non-functional requirements.

More importantly, the results of this case study further show that there is potential for taking a tactics-driven approach to fulfill functional and non-functional requirements for cyber-foraging systems. As indicated by the developers of the GigaSight system, a catalog of architectural tactics would definitely be an asset for the development of real cyber-foraging systems. The next case study explores the use of the architectural tactics in the development of a new computation offload and data staging cyber-foraging system.

5.5 Acknowledgments

Very special thanks to Pieter Siemons from Ghent University and Zhou Chen from Carnegie Mellon University for their invaluable support during the execution of this case study.

6

Case Study 3: AgroTempus — Using Architectural Tactics for Cyber-Foraging Systems Development

This chapter addresses research question RQ2 and is the final of three case studies to validate the architectural tactics presented in Chapter 3. The goal of this case study is to demonstrate the use of the architectural tactics in the development of a cyber-foraging system for both computation offload and data staging targeted at agricultural knowledge exchange in resource-challenged regions, to then determine how the selected tactics support their intended functional and non-functional requirements.

6.1 Introduction

This chapter continues the validation of the architectural tactics for cyber-foraging presented in Chapter 3, in the context of research question RQ2: *What architectural tactics can be derived from the identified architectural design decisions?*

The goal of this third case study is to identify tactics that could be used in the development of the AgroTempus system, targeted at agricultural knowledge exchange in resource-challenged regions, and then to validate if the implementation of each of the tactics met its intended functional and non-functional requirements.

As in the case studies in the previous two chapters, we followed the guidelines for conducting case studies from [15] and [118]. Accordingly, the structure of the chapter follows the steps proposed in these guidelines. Section 6.2

presents the case study design, including research questions and procedures for data collection and analysis. Section 6.3 presents the results of the case study and threats to validity. Section 6.4 concludes the chapter with a summary of the findings as well as their implications and limitations.

6.2 Case Study Design

6.2.1 Research Questions

Given the goal to determine if the architectural tactics for cyber-foraging identified in Chapter 3 can be used in the development of the AgroTempus system, we defined the following research questions to be answered in the execution of the case study. These questions are slightly different from the previous two case studies as the context is the use of the tactics in new development, as opposed to the analysis of an existing system to identify tactics.

- Which of the architectural tactics for cyber-foraging can be used in the development of the AgroTempus system to fulfill its functional and non-functional requirements?
- How do the selected tactics support their intended functional and non-functional requirements?

6.2.2 Data Collection Procedure

Data collection involves identifying the data to be collected, defining a data collection plan, and defining how the data will be stored [15]. Given that the goal of this case study is to determine if the architectural tactics for cyber-foraging can be used in the development of a new system, the data collection is executed with direct observation of the development process (first degree data collection method) combined with developer and project stakeholder interviews for validation (first degree data collection method) [118].

We therefore define the following steps to collect data about the use of architectural tactics for cyber-foraging in the development of the AgroTempus system that will enable us to answer the case study research questions:

1. Gather system requirements: System requirements are gathered by the system developer from the main project stakeholder. The identified requirements are documented and confirmed by the main stakeholder.

2. Map system requirements to architectural tactics for cyber-foraging: The system developer maps system requirements to functional and non-functional tactics for cyber-foraging that could be used in the realization of the requirements.

3. Develop system architecture: The system developer designs the software architecture for the AgroTempus system using components derived from the identified architectural tactics, combined with components to address requirements that are outside the scope of the architectural tactics for cyber-foraging. The system architecture is documented as a component-and-connector diagram.

4. Map architecture components to system requirements: The system developer maps architecture components to system requirements to ensure that all system requirements are assigned to components of the architecture.

5. Map architecture components to identified architectural tactics: The system developer maps architecture components and design decisions to elements of the identified tactics.

6. Implement system based on system architecture: The system developer implements the system according to specifications provided by the system architecture.

6.2.3 Analysis Procedure

Once the system is implemented, we perform two activities as part of the analysis:

1. Qualitatively and quantitatively (if possible) determine if the implementation of the tactics meets the corresponding system requirements: The system is validated against the system functional and non-functional requirements to determine if requirements are met as intended by the architectural tactics..

2. The developer is observed and interviewed throughout the design and implementation of the system to understand how the architectural tactics were used and how they influenced the development process.

6.3 Results

6.3.1 System Context

As many developing areas have to deal with the lack of proper access to resources such as Internet and electricity, cyber-foraging offers potential solutions to these resource challenges by leveraging proximate surrogates that can provide services that involve heavy computation such as image processing, store large sets of data collected in the field, or store information retrieved from data centers during scarce moments of Internet connectivity.

The goal of the AgroTempus system is to enable people involved in agriculture (e.g., farmers and non-governmental organization (NGO) employees helping farmers), who work in environments with little to no access to the Internet or electricity, to collect and retrieve data about the weather in their area. AgroTempus performs different types of computation on the collected data as examples of valuable services for its users.

End users interact with the system with smartphones, the proliferation of which is predicted to rise significantly in the coming years in developing regions [34][125]. The capabilities of the mobile applications running on the smartphone are extended by surrogates in the form of single-board computers running on solar power. To be able to eventually store all collected data in a cloud-based back-end, a mobile hub carrying a computer system with increased storage capabilities will connect to each surrogate periodically, and eventually connect to the Internet. This also makes it possible to propagate data from the Internet to the surrogates and mobile devices. This setup was inspired by the DakNet project in India [97].

Figure 6.1 shows an overview of the system. The mobile hub (3) is a computer system with networking and storage capabilities (e.g., laptop) carried by a vehicle. It can move from villages that lack access to the Internet to a larger city that does have access (1), such that it can store as well as retrieve data that can then be used for services in the villages. Surrogates (4) are single-board computers (e.g., Raspberry Pi[1]), extended with network adapters and solar batteries. Mobile devices (2) are smartphones, most of which will be in the low-end range, generally with computing capability and storage capacity lower than the surrogates.

[1] `http://www.raspberrypi.org/`

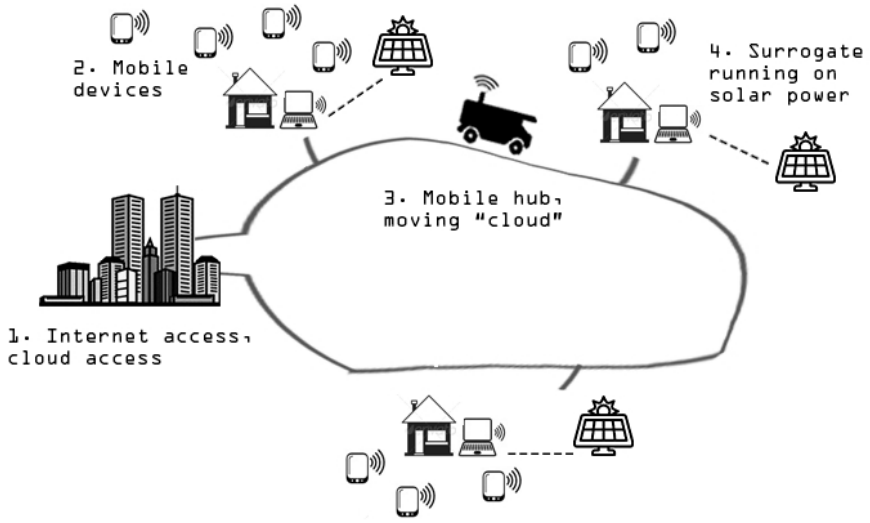

Figure 6.1: AgroTempus System Context

6.3.2 System Requirements

6.3.2.1 Functional Requirements

The AgroTempus system needs to satisfy the following functional requirements.

- **FR1: Store weather data**: NGO employees and farmers can store weather data related to a certain area via a mobile app.
- **FR2: Retrieve weather data**: NGO employees and farmers can retrieve weather data related to a certain area using a mobile app. This data is derived from earlier reports (FR1), as well as from a third-party weather API accessible via the Internet.
- **FR3: Perform regressions on weather data**: NGO employees can select a weather information dataset and perform a regression on it using the mobile app. A visualization of the results will be available when the operation completes.
- **FR4: Predict future weather data values**: NGO employees and farmers can obtain predictions of future values of variables related to

the weather, based on data collected in the field, up to a week in the future.

- **FR5: Surrogate setup**: Surrogates are assigned to serve a certain region and as such need a setup procedure that enables NGO employees to enter the correct settings before it can be used.
- **FR6: Forecast delivery**: Weather forecasts for the region that the user is in can be retrieved using a mobile app.
- **FR7: Integration with cloud-based storage systems**: The system eventually stores all data collected from mobile devices in a cloud-based system such as ERS [17].
- **FR8: Voice interface**: The user interface for the farmers can support voice instructions to help users navigate the app.
- **FR9: Synchronize weather data**: Periodically, the latest weather forecasts and data for relevant regions are retrieved from a third-party weather API on the Internet. This data is eventually stored on the surrogates.
- **FR10: Surrogate registration on mobile hub**: When new surrogates are added to the system and are operational, their identification and location information (as provided in FR5) is stored on the mobile hub so that it can collect relevant data for this surrogate (FR9).

6.3.2.2 Non-Functional Requirements

The AgroTempus system needs to satisfy the following non-functional requirements.

- **NFR1: Fault tolerance and reliability**: The system should be able to recover from failures such as crashes and loss of connection between mobile devices and surrogates.
  - Because it is expected that there will be few people proficient in IT in the regions where the system will be used, surrogates should be able to detect failures in the services that they offer and restart them accordingly.
  - Losing connection during the interaction between surrogates and mobile hubs, as well as between surrogates and mobile devices,

should not cause the services running on the surrogates to stop functioning.

  - Because it is expected that mobile app users will regularly be moving in and out of range of surrogates during use of the system, this should not cause users to lose results of completed computations or lose data that they have stored on the mobile app.

- **NFR2: Ease of deployment**: The system should be easy to deploy.

  - The mobile app can be installed through an app store and does not have to be configured. It should detect and connect to surrogates automatically.
  - Surrogates have to be configured locally (FR5), and this process should be able to be performed by NGO personnel with only basic IT knowledge. It should be a simple process, comparable to entering data in a form and confirming.
  - Active surrogates should register with the mobile hub automatically on first connection.

- **NFR3: Usability**: Literacy among users of mobile devices will vary. Most end users will have low technical knowledge as well. The interfaces to the functionality that they use should be understandable to them.

  - Text in English, including voice explanations.
  - Text in French, including voice explanations (one of the target languages, but will not be implemented in the AgroTempus system).

- **NFR4: Extensibility**: Developing new functionality and adding it to the system should be supported and made easy. A standard format for services that perform either computation offload or data staging should be available to future developers, including documentation and an example.

- **NFR5: Energy efficiency**: The mobile device and surrogate systems will run in an energy-challenged environment. Access to electrical power is limited and not always available.

  - Energy use on mobile devices should be minimized.
  - Energy use on surrogates should be minimized, but energy efficiency for mobile devices has higher priority.

- **NFR6: Capacity**: Low-end smartphones have low storage capacity and therefore storage should, for the most part, be the responsibility of the surrogates and mobile hubs.
  - The surrogate should be able to provide computation offload and data staging capabilities to multiple users at the same time.
  - Storage used on smartphones should be kept under 100 MB, not counting results for calculations that the user has saved.
  - Surrogates should be able to run 10 instances of services at the same time.
- **NFR7: Availability**: Capabilities provided by surrogates should, in principle, be available 24 hours a day. However, because surrogates will run on solar energy, it is expected that they can run out of energy during heavy use, especially during periods with no or little sunshine.
  - Every 24-hour period, the surrogate should be able to deliver services amounting to 4 hours of surrogate activity. This does not provide guarantees about unavailability due to crashes (which is discussed in NFR1 and NFR9).
  - When remaining battery life drops below 10% of the battery's capacity, computations that will take longer than 5 minutes should be queued until the battery is recharged to above 15%.
- **NFR8: Performance**: There are no hard performance requirements, except for the transfer of data between the mobile hub and the surrogate. This is because the window during which there is opportunity to interchange data is short and infrequent.
  - The transfer of data between the mobile hub and the surrogate should be prioritized over other offloaded computation or data staging operations that the surrogate is performing.
  - The only operation with higher priority is the registration of a new surrogate.
  - The mobile hub should check for a surrogate broadcast signal at least 10 times per second, as long as it is not interacting with one already.
  - The surrogate should broadcast its presence at least 10 times per second.

- **NFR9: Recovery**: When a surrogate has crashed, restarting the hardware should have it operational again within 10 minutes. Similarly, when a mobile hub has crashed, resetting the hardware should have it operational again within 10 minutes.
- **NFR10: Data integrity**: When weather data is entered on the mobile app, it should be checked for valid values, e.g., temperature values between certain valid limits. The same applies to setup data during the setup process.

6.3.2.3 Constraints and Assumptions

The following constraints for the development of the AgroTempus system were identified:

- **C1: Low cost infrastructure and hardware**: End-users will mostly use low-end mobile devices, while the rest of the system will be deployed on hardware locally, for which the cost should be as low as possible.
- **C2: Use of FirefoxOS**: Agrotempus has to be developed for the FirefoxOS mobile operating system [89]. FirefoxOS is open source, based on standard Web APIs, and targeted at low-end smartphones and developing markets.
- **C3: Use of open standards**: There is a preference for open source components and the use of open standards where possible.
- **C4: Use of Java**: Because the implementation platform for surrogates is still evolving, the preference is to use Java due to its portability.

Only one assumption for the system was identified:

- **A1: Concurrent access to multiple surrogates**: Surrogate signals do not overlap because there is only one surrogate per village. This means the mobile devices and mobile hub can connect to different surrogates, but never at the same time.

6.3.3 Mapping of System Requirements to Architectural Tactics

Based on the functional and non-functional requirements for the AgroTempus system, several tactics were identified by the developer as feasible for their

fulfillment. The mapping of system requirements to architectural tactics from the catalog presented in Chapter 3 is shown in Table 6.1. The rationale for the selection of each tactic, as indicated by the developer, is provided in the following sub-sections.

Table 6.1: Mapping of System Requirements to Architectural Tactics

Tactic	FR1: Store Weather Data	FR2: Retrieve Weather Data	FR3: Data Regression	FR4: Future Weather Values	FR5: Surrogate Setup	FR6: Forecast Delivery	FR7: Integration Cloud Storage	FR8: Voice Interface	FR9: Synchronize Weather Data	FR10: Surrogate Registration	NFR1: Fault Tolerance	NFR2: Ease of Deployment	NFR3: Usability	NFR4: Extensibility	NFR5: Energy Efficiency	NFR6: Capacity	NFR7: Availability	NFR8: Performance	NFR9: Recovery	NFR10: Data Integrity
Computation Offload			X	X											X	X				
Out-Bound Pre-Processing	X						X									X				
Pre-Fetching		X				X			X							X				
Pre-Provisioned Surrogate	X	X	X	X	X	X	X		X	X		X							X	
Surrogate Broadcast										X		X						X		
Cached Results			X	X							X						X			
Client-Side Data Caching	X										X						X			
Just-in-Time Containers			X	X											X	X				

6.3.3.1 Computation Offload

The Computation Offload tactic (Section 3.2.1) enables mobile clients to offload expensive computation to surrogates. Regression and weather value prediction using extrapolation (FR3 and FR4) are computation-intensive operations that are initiated by the user on the mobile device, but the computation is offloaded to the surrogate. Data sets on which these operations are performed are located at the surrogates and can be reasonably large, while the input for the operations is a small set of variables of simple data types. Offloading small input/output, energy-intensive computations to the surrogate is the main method to minimize energy consumption on the mobile device (NFR5).

Offloading from from low-end mobile devices to surrogates with more computational power and data storage facilities increases the capacity of the system (NFR6).

6.3.3.2 Out-Bound Pre-Processing

In the Out-Bound Pre-Processing tactic (Section 3.2.2.3) surrogates collect data from mobile devices and pre-process the data – clean, filter, summarize, or merge — such that the data that is sent on to the enterprise cloud is ready for consumption and serves an immediate need. Weather data collected on mobile devices (FR1) is stored locally until it has been successfully transferred to a surrogate. The surrogates will store this data indefinitely, both to make it accessible to mobile users in the future, but also to make it available to the mobile hub, which will collect all data eventually. This data will not be saved on the mobile device after it has been successfully transferred to the surrogate because storage is limited on the low-end mobile devices (NFR6). The mobile hub will eventually be able to store new data that was entered on the mobile device in the cloud when it connects to the Internet (FR7). In the AgroTempus system there are therefore two levels of data staging: first at the surrogate and then at the mobile hub.

6.3.3.3 Pre-Fetching

The Pre-Fetching tactic (Section 3.2.2.1) anticipates data needs in order to minimize communication to the cloud and reduce latency. The mobile hub, according to a defined pre-fetch algorithm, retrieves weather data using a third-party weather API (FR9) based on the registered location of all the surrogates that it serves. Data retrieved from the mobile hub is stored on the surrogates and not the mobile devices to address storage limitations of low-end mobile devices (NFR6). Mobile devices that request weather data (FR2) will always obtain it from a surrogate where this data is staged, unless it has been explicitly saved on the mobile device by the user. The same is true for forecasts (FR6), which are calculated based on data downloaded from the mobile hub.

6.3.3.4 Pre-Provisioned Surrogate

In the Pre-Provisioned Surrogate tactic (Section 3.2.3.1) surrogates are provisioned before their deployment with the capabilities that are offloaded by mobile clients. All required functionality will be available on the surrogate from the start (FR1, FR2, FR3, FR4, FR6, FR7, FR9, FR10). Because all

surrogates serving all regions have the same capabilities, it is easier to provision them using the same OS image (e.g, Raspberry Pi with cloned SD card) (NFR2). The only difference between surrogates is the location and identification settings provided during the setup procedure (FR5). Restarting a pre-provisioned surrogate is easier to do if started from a common OS image (NFR9).

6.3.3.5 Surrogate Broadcast

In the Surrogate Broadcast tactic (Section 3.2.4.3) surrogates advertise their availability and selected metadata to mobile devices for discovery. Mobile device users should be able to make use of system functionality as soon as they install the app and come in range of a surrogate. To increase the ease of deployment (NFR2), surrogates broadcast their presence and mobile devices in need of surrogate services can pick up on these broadcasts. Surrogate broadcast is also key for the automatic registration of newly deployed surrogates with the mobile hub as soon as they are in communication range (FR10). Lastly, because the opportunities for interaction between surrogates and the mobile hub are scarce, both the surrogate broadcasting its presence continuously and the mobile hub continuously trying to discover surrogates are key to the system's performance (NFR8).

6.3.3.6 Cached Results

The Cached Results tactic (Section 3.3.2.3) enables a system to cache results and state on a surrogate until the mobile device is able to reconnect. The interaction between mobile devices and surrogates is susceptible to loss of connection in the AgroTempus system. When computation offload (FR3, FR4) has been correctly initiated, but the mobile user moves out of range of the surrogate during the computation, results should be cached (NFR1) so they can be sent to the user as soon as the mobile device connects to the surrogate again to promote availability (NFR7).

6.3.3.7 Client-Side Data Caching

The Client-Side Data Caching tactic is a variation of the Cached Results tactic (Section 3.3.2.3). Data collected by a mobile client is cached on the mobile device and sent to the surrogate upon connection or re-connection. Because mobile devices are not always in proximity of a surrogate, when entering weather data (FR1) without an available connection, data is cached on the mobile

device (NFR1), which will periodically try to resend the data. In this case, caching is used to enable users to keep working with the app, saving new readings, and not having to worry about the data being saved immediately on the surrogate, therefore promoting availability (NFR7).

6.3.3.8 Just-in-Time Containers

The Just-in-Time Containers tactic (Section 3.3.3.1) 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 the offload request is completed. Data regression (FR3) and weather value prediction (FR4) are heavy computations that will be used infrequently. Therefore, as opposed to the other services offered by surrogates, these services are better suited to run in their own containers, such that small operations will not get queued behind these large computations. To be able to handle multiple computation offload requests at the same time, as well as to not let these large computations cause small data transfers to have to wait for them (NFR6), each time a request for a computation offload is received at the surrogate, a container with the necessary functionality is created. Because requests for computation offload will be infrequent, often with long periods of time between requests, only creating containers for these capabilities when they are needed is a tactic that will save energy on the surrogate (NFR5).

6.3.4 System Architecture and Design

Based on the identified tactics, the developer created the high-level architecture for the AgroTempus system shown in Figure 6.2 as a UML component diagram. Some components of the architecture were derived from the architectural tactics and others were added to fulfill requirements not addressed by the tactics. The detailed components and mapping to the tactics are presented in Section 6.3.6. The main elements of the architecture are:

- Mobile Device Components
  - CD1: Voice Support Manager: Manages the voice snippets that map to the user interface elements.
  - CD2: Cyber-Foraging Enabled App User Interface: User interface component of the mobile app.
  - CD3: Mobile App Storage Manager: Manages storage of all permanent data and user settings on the mobile app, except for data

that is being staged before moving to the surrogate. Storing and retrieving data is done through its interfaces: *Store app data* and *Retrieve app data*.

- CD4: Offload Client: Handles computation offload from the mobile app to the surrogate, initiated through component CD2.
- CD5: Mobile App Data Exchange Client: Handles staging data and transferring it from the mobile app to the surrogate after it has been entered via component CD2. It also handles requesting and receiving data from the surrogate.
- CD22: Surrogate Discovery Manager: Finds available surrogate services.

- Surrogate Components
  - CD6: Offload Server: Handles requests for computation offload from mobile devices.
  - CD7: Setup Manager: Implements the setup process for newly deployed surrogates. Provides the interface *Setup surrogate*, which is used by component CD10 when the setup process is started.
  - CD8: Data Request Server: Handles requests for data stored on the surrogate from mobile devices, as well as from the mobile hub.
  - CD9: Offloaded Computation Manager: Creates containers that run offloaded computation and ensures that results are eventually stored in component CD13.
  - CD10: Surrogate User Interface: User interface component for the surrogate, available when a screen and mouse/keyboard are connected to the surrogate (e.g., during the setup process or to check console output).
  - CD11: Broadcast Manager: Broadcasts the presence of the surrogate and its capabilities through the interface *Broadcast services*. It is key for all requirements in which interaction between the surrogate and other system nodes is involved.
  - CD12: Data Storage Server: Handles requests from mobile devices and the mobile hub for storing data on the surrogate.
  - CD13: Surrogate Storage Manager: Manages storage of all permanent data, computation results, and settings on the surrogate. The interfaces for data retrieval include the possibility to delete data after a successful transmission.

- Mobile Hub Components
  - CD14: Surrogate Registration Manager: Handles the registration of surrogates that are new to the system by picking up broadcasts from component CD11 and storing new surrogate data in component CD19.
  - CD15: Mobile Hub Synchronization Client: Manages synchronization of data between the mobile hub and the surrogate.
  - CD16: Mobile Hub User Interface: User interface component for the mobile hub.
  - CD17: Cloud Synchronization Client: Ensures that data stored in the system is backed up to a cloud repository by interacting with component CD20.
  - CD18: API Data Fetcher: Retrieves weather data from a third party API and stores it on the mobile hub via component CD19. It also periodically checks whether the surrogate list stored by this component has new entries.
  - CD19: Mobile Hub Storage Manager: Handles storage and retrieval of data on the mobile hub, including settings, staged data, permanent weather data, and the list of known surrogates.
- Cloud Repository Component (External)
  - CD20: Cloud Repository Storage Manager: Third-party component that interacts with component CD17 to ensure that data stored in the system is backed up to a cloud repository.
- Internet Weather Service (External)
  - CD21: Weather API: Third-party component that provides weather data and forecasts through a REST interface.

6.3.5 Mapping of Architectural Components to System Requirements

The mapping of functional and non-functional requirements to components of the architecture is shown in Table 6.2. All requirements are implemented by one or more components, with the exception of *NFR4: Extensibility* because this requirement is related to the creation of artifacts to support developers, such as templates and documentation, and not to specific runtime components.

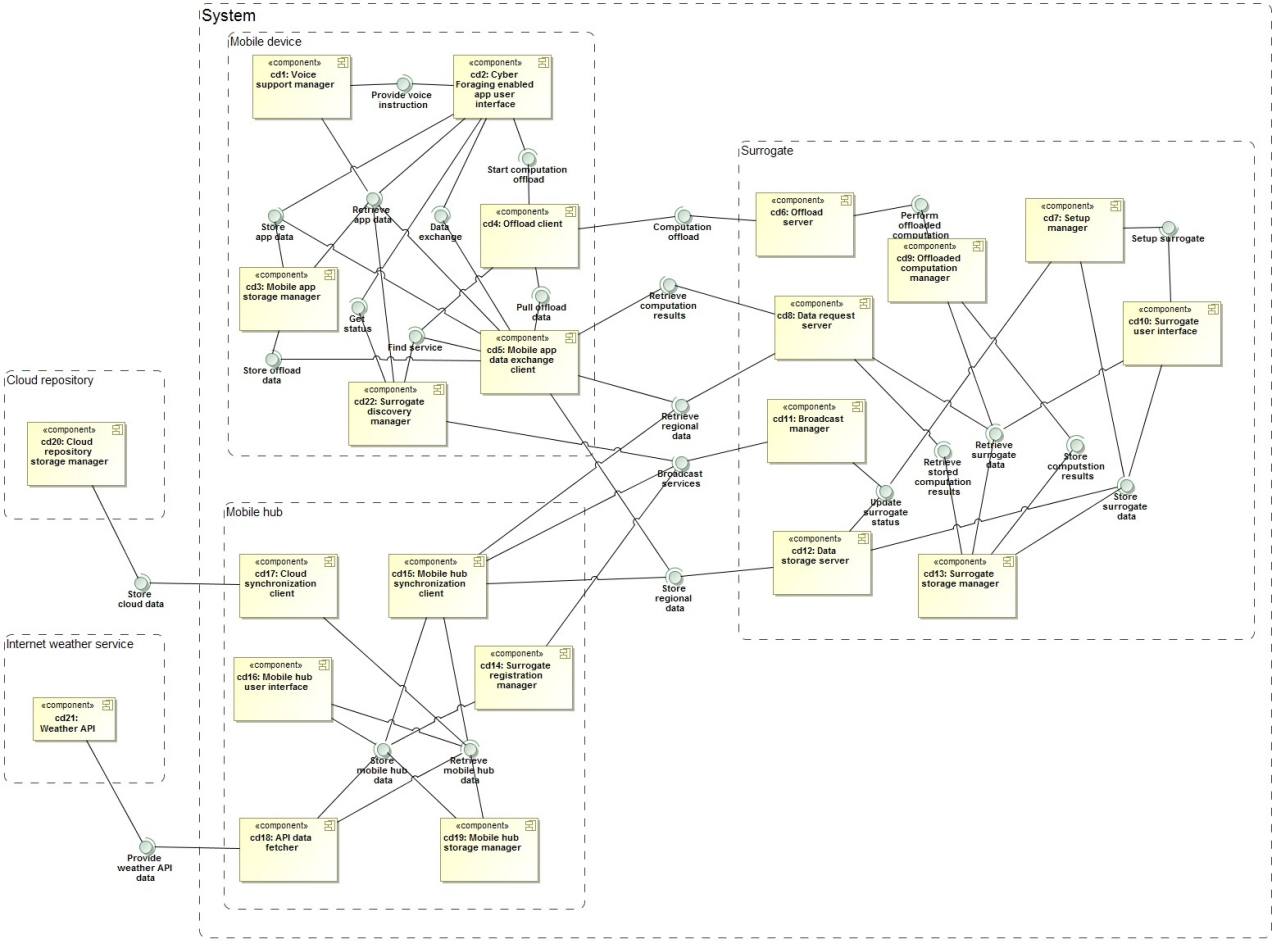

Figure 6.2: High-Level Architecture of the AgroTempus System

Table 6.2: Mapping of System Requirements to Architecture Componentss

Component	FR1: Store Weather Data	FR2: Retrieve Weather Data	FR3: Data Regression	FR4: Future Weather Values	FR5: Surrogate Setup	FR6: Forecast Delivery	FR7: Integration Cloud Storage	FR8: Voice Interface	FR9: Synchronize Weather Data	FR10: Surrogate Registration	NFR1: Fault Tolerance	NFR2: Ease of Deployment	NFR3: Usability	NFR4: Extensibility	NFR5: Energy Efficiency	NFR6: Capacity	NFR7: Availability	NFR8: Performance	NFR9: Recovery	NFR10: Data Integrity
CD1: Voice Support Manager								X					X		X					
CD2: App User Interface	X	X	X	X		X		X					X		X					X
CD3: App Storage Manager		X	X	X		X		X			X		X		X	X				
CD4: Offload Client			X	X							X		X		X					
CD5: App Data Exch Client	X										X		X		X					
CD6: Offload Server			X	X							X		X		X	X			X	
CD7: Setup Manager					X						X	X	X		X				X	X
CD8: Data Request Server	X	X	X	X		X					X		X		X	X			X	
CD9: Offloaded Comp Manager			X	X							X		X		X	X	X		X	
CD10: Surrogate UI					X						X	X	X		X				X	
CD11: Broadcast Manager	X	X	X	X		X	X		X	X	X	X	X		X			X	X	
CD12: Data Storage Server	X					X			X		X		X		X	X			X	
CD13: Surrogate Storage Mgr	X	X	X	X	X	X			X		X		X		X				X	
CD14: Surrogate Reg Mgr										X	X	X	X					X	X	
CD15: Mobile Hub Sync Client	X	X				X	X		X		X		X					X	X	
CD16: Mobile Hub UI													X						X	
CD17: Cloud Sync Client							X						X						X	
CD18: API Data Fetcher		X				X			X				X						X	
CD19: Mobile Hub Storage Mgr	X					X	X		X	X			X						X	
CD20: Cloud Repo Storage Mgr							X													
CD21: Weather API		X				X			X											
CD22: Surrogate Discovery Mgr	X	X	X	X		X					X	X	X		X					

6.3.6 Mapping of Architectural Components to Identified Architectural Tactics

The mapping between architecture components and the architectural tactics identified in Section 6.3.3 is provided in the following subsections to show component details, as well as the mapping to specific architectural tactic elements. All design decisions described at this point correspond to the as-initially-designed system. The final implementation decisions are described in Section 6.3.7.

6.3.6.1 Computation Offload

The Computation Offload tactic is designed in the AgroTempus architecture as shown in Figure 6.3(b), with numbers to indicate the sequence of operations. The component names in Figure 6.3(a) are used as stereotypes for the components in Figure 6.3(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included. The offload operation takes place as follows:

1-2. The Cyber-Foraging Enabled App User Interface requests to start an offloaded computation with input *Input*.

3. The Offload Server receives the request and invokes the Offloaded Computation Manager.

4. The Offloaded Computation Manager starts the offloaded computation in a separate Java Thread inside the JVM.

The main difference between the Computation Offload tactic and the AgroTempus architecture is how the offloaded computation is executed. In the tactic shown in Figure 6.3(a), after the offloaded computation is set up, the control returns to the *Cyber-Foraging Enabled Mobile App*, which then executes the offloaded computation via the operation `4:Execute(Input)`. This is because the assumption is that the app interacts with the offloaded code in a request/response manner until the app closes. In the AgroTempus system, offloaded computation corresponds to a lengthy computation that is executed only once in an offload request. Therefore, the *Input* to the offloaded computation is sent in the initial request to offload.

6.3.6.2 Out-Bound Pre-Processing

The Out-Bound Pre-Processing tactic is designed in the AgroTempus architecture as shown in Figure 6.4(b), with numbers to indicate the sequence of operations. The component names in Figure 6.4(a) are used as stereotypes for

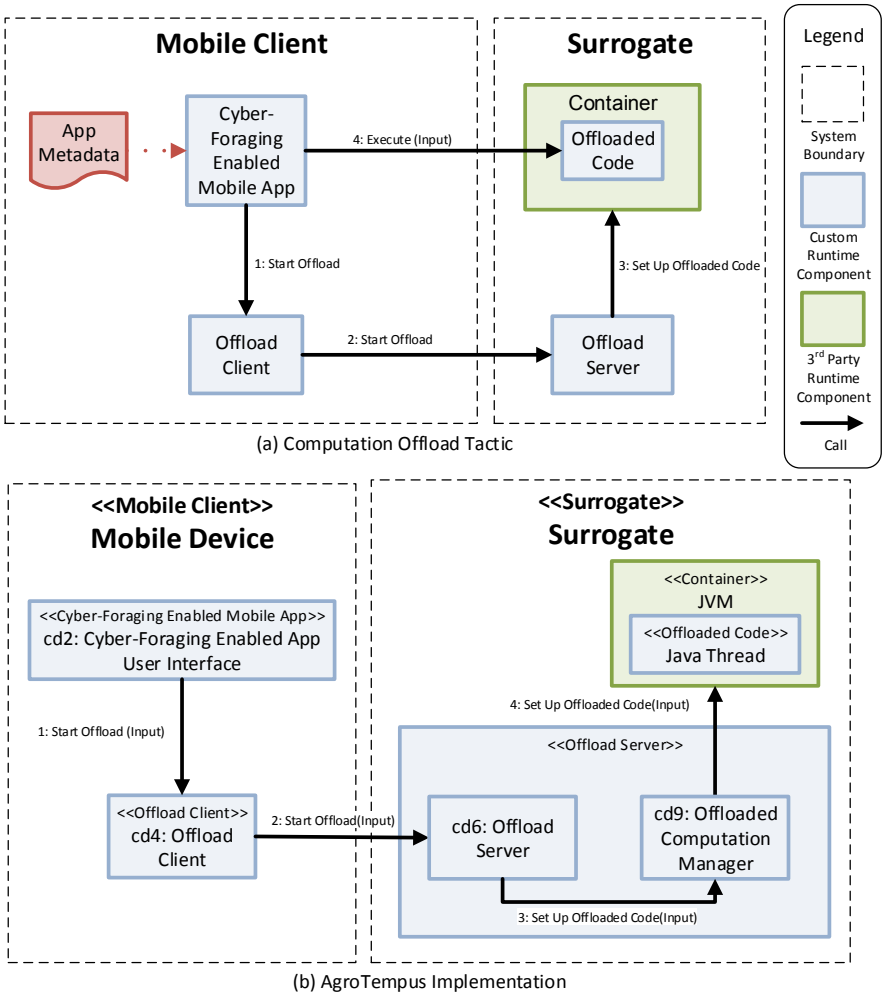

Figure 6.3: Mapping of the AgroTempus Architecture to the Computation Offload Tactic

the components in Figure 6.4(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included. The data staging from the mobile device to the operation takes place as follows:

1. The Cyber-Foraging Enabled App User Interface captures weather data and sends it to the Mobile App Data Exchange Client.

2-3. The Mobile App Data Exchange Client queues the weather data until a surrogate is in range and then sends it to the Data Storage Server for storage on the surrogate via the Surrogate Storage Manager.

4. The Data Request Server on the surrogate waits for a weather data request from the Mobile Hub Synchronization Manager (this happens when the mobile hub is in range of the surrogate).

5-6. The Data Request Server retrieves the weather data and sends it to the mobile hub for storage on the mobile hub via the Mobile Hub Storage Manager.

7-9. Once the Cloud Synchronization Client on the mobile hub has connectivity to the cloud repository, it retrieves the weather data from the Mobile Hub Storage Manager and sends it to the Cloud Repository Storage Manager for storage in the Cloud Repository.

The difference between the AgroTempus architecture and the Out-Bound Pre-Processing tactic is that the AgroTempus system performs data staging at two levels to get data from the mobile devices to the cloud: first at the surrogate and then at the mobile hub. Therefore, the Data Request Server on the surrogate and the Cloud Synchronization Client on the mobile hub perform two roles: data processing application for the cached data and communication manager for passing the information to the next level en route to the enterprise cloud.

6.3.6.3 Pre-Fetching

The Pre-Fetching tactic is designed in the AgroTempus architecture as shown in Figure 6.5(b), with numbers to indicate the sequence of operations. The component names in Figure 6.5(a) are used as stereotypes for the components in Figure 6.5(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included. The pre-fetching of data from the enterprise cloud to the surrogates serving mobile devices takes place as follows:

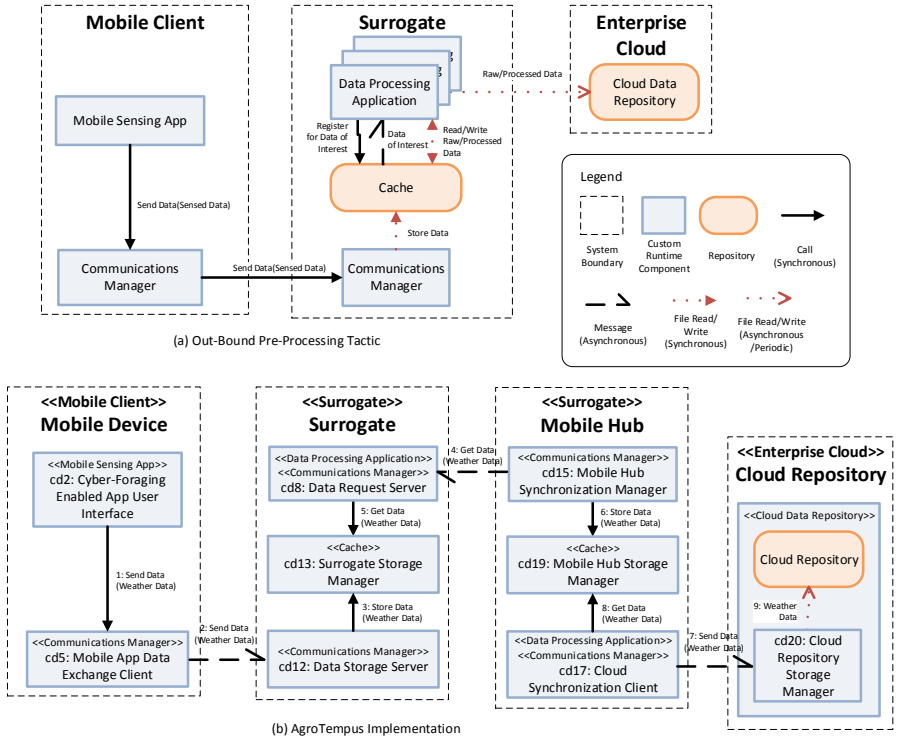

Figure 6.4: Mapping of the AgroTempus Architecture to the Out-Bound Pre-Processing Tactic

1. When the Mobile Hub has access to the Internet Weather Service, the Cloud Synchronization Client retrieves all weather data for the surrogates that it serves from the Weather API, based on the Surrogate Location List. It then caches the retrieved weather data.

2-3. When the Mobile Hub is in proximity of a Surrogate that it serves, the Mobile Hub Synchronization Manager reads the data for the surrogate location and pushes it to the Data Request Server on the Surrogate.

4. The Data Request Server caches the data on the Surrogate via the Surrogate Storage Manager.

5-7. When the mobile app has a request for weather data, the data is obtained from the Surrogate.

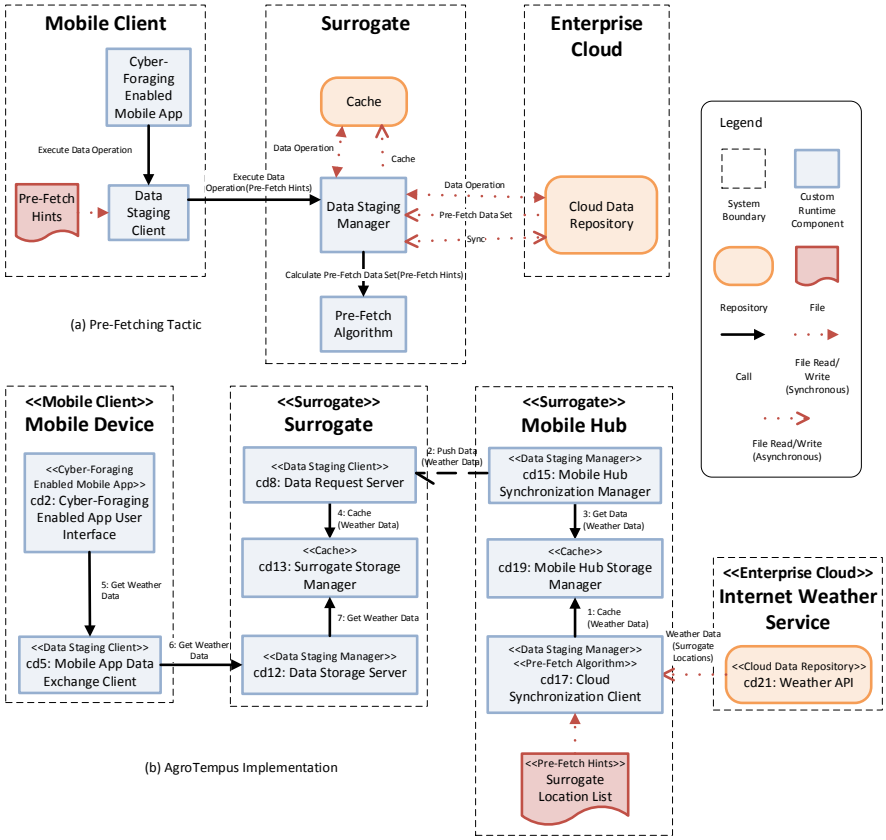

Figure 6.5: Mapping of the AgroTempus Architecture to the Pre-Fetching Tactic

There are two differences between the AgroTempus architecture and the Pre-Fetching tactic:

1. The AgroTempus system performs data staging at two levels to pre-fetch data from the cloud and host it on the surrogates: first from the cloud to the mobile hub, and then from the mobile hub to the surrogate.

2. The Pre-Fetch Algorithm and Pre-Fetch hints reside on the mobile hub and not on the mobile client. This is because the mobile hub needs to

fetch data from the cloud at a point in time when it is not likely that it will be near a surrogate or a mobile device. The Surrogate Location List is populated during Surrogate Registration (FR10).

6.3.6.4 Pre-Provisioned Surrogate

The Pre-Provisioned Surrogate tactic is designed in the AgroTempus architecture as shown in Figure 6.6(b), with numbers to indicate the sequence of operations. The component names in Figure 6.6(a) are used as stereotypes for the components in Figure 6.6(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included. The provisioning of capabilities on the surrogate takes place as follows:

1. A Terminal on the Surrogate is used to load the Surrogate Component Code Files on the Surrogate.

2-3. The Terminal is used to start the Surrogate User Interface to obtain setup parameters for the surrogate, such as location.

4-6. The Surrogate User Interface invokes the Setup Manager to start the remaining surrogate components.

Step 1 of the provisioning process is only executed once prior to surrogate deployment. Steps 2 and 3 are executed only once during surrogate deployment. Steps 4-6 are executed manually during deployment, and then automatically on start/restart of the surrogate. There is not the equivalent of the Capability Metadata component nor a Capability Registry component because the capabilities provided to all mobile devices are the same and are not advertised. In addition, there is not the equivalent of a Remote User Interface because surrogates are envisioned to be low cost, low-end servers that are set up on site.

6.3.6.5 Surrogate Broadcast

The Surrogate Broadcast tactic is designed in the AgroTempus architecture as shown in Figure 6.7(b), with numbers to indicate the sequence of operations. The component names in Figure 6.7(a) are used as stereotypes for the components in Figure 6.7(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included. The discovery of a surrogate takes place as follows:

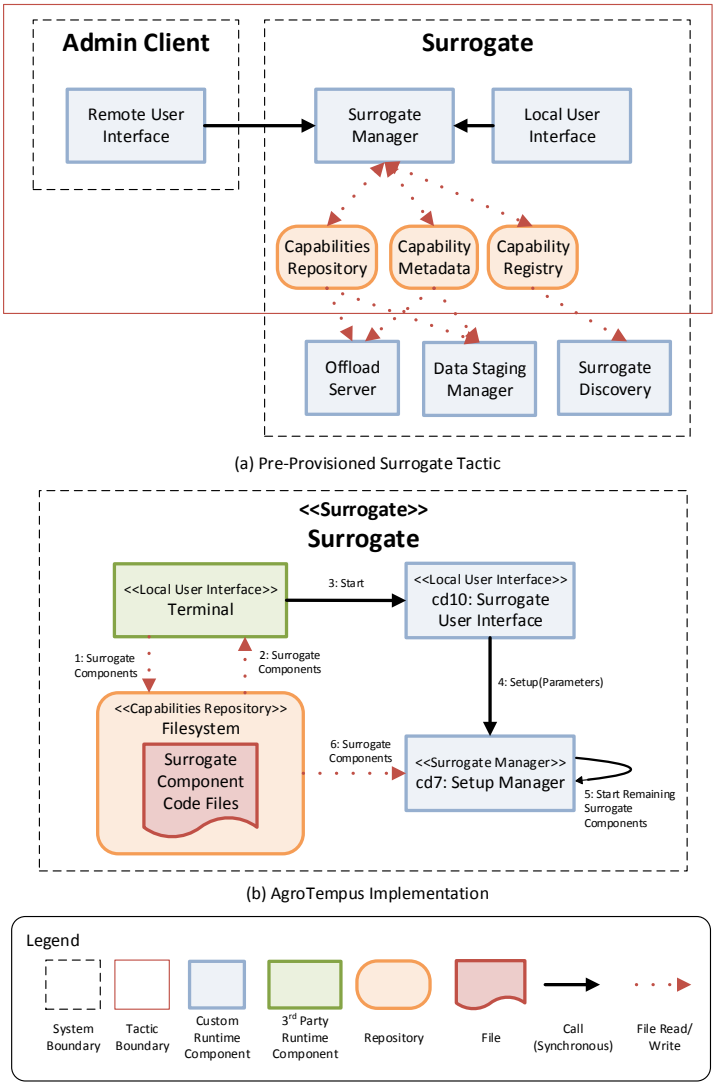

Figure 6.6: Mapping of the AgroTempus Architecture to the Pre-Provisioned Surrogate Tactic

0. The Broadcast Manager running on the Surrogate broadcasts its address.
1. The Cyber-Foraging Enabled Mobile App User Interface requests an offload operation.
2. The Offload Client receives the request and obtains the surrogate address from the Surrogate Discovery Manager.
3. The Offload Client sends the offload operation to the Offload Server at the surrogate address.

The difference between the AgroTempus architecture and the Surrogate Broadcast tactic is that there is no need to find an optimal surrogate because only one surrogate is available for a mobile device. The assumption as stated in Section 6.3.2.3 is that there is only one surrogate per village, and surrogate signals do not overlap. Even though not shown in the Figure 6.7, the surrogate also broadcasts its presence to the mobile hub via the same mechanism.

6.3.6.6 Cached Results

The Cached Results tactic is designed in the AgroTempus architecture as shown in Figure 6.8(b), with numbers to indicate the sequence of operations. The component names in Figure 6.8(a) are used as stereotypes for the components in Figure 6.8(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included. The caching of results on a surrogate takes place as follows:

1-2. The Cyber-Foraging Enabled App User Interface requests to start an offloaded computation with input *Input*.

3. The Offload Server receives the request and invokes the Offloaded Computation Manager.

4-7. The Offloaded Computation Manager assigns the computation a unique identifier called a Ticket, starts the offloaded computation in a separate Java Thread inside the JVM, and returns an Acknowledgment to the Cyber-Foraging Enabled App User Interface with the assigned Ticket.

8. The Offloaded Computation executes and saves the results in the Surrogate Storage Manager with the assigned Ticket.

9-10. The Cyber-Foraging Enabled App User Interface, via the Mobile App Data Exchange Client, sends a request to the Data Request Server on the Surrogate for the results for the received Ticket.

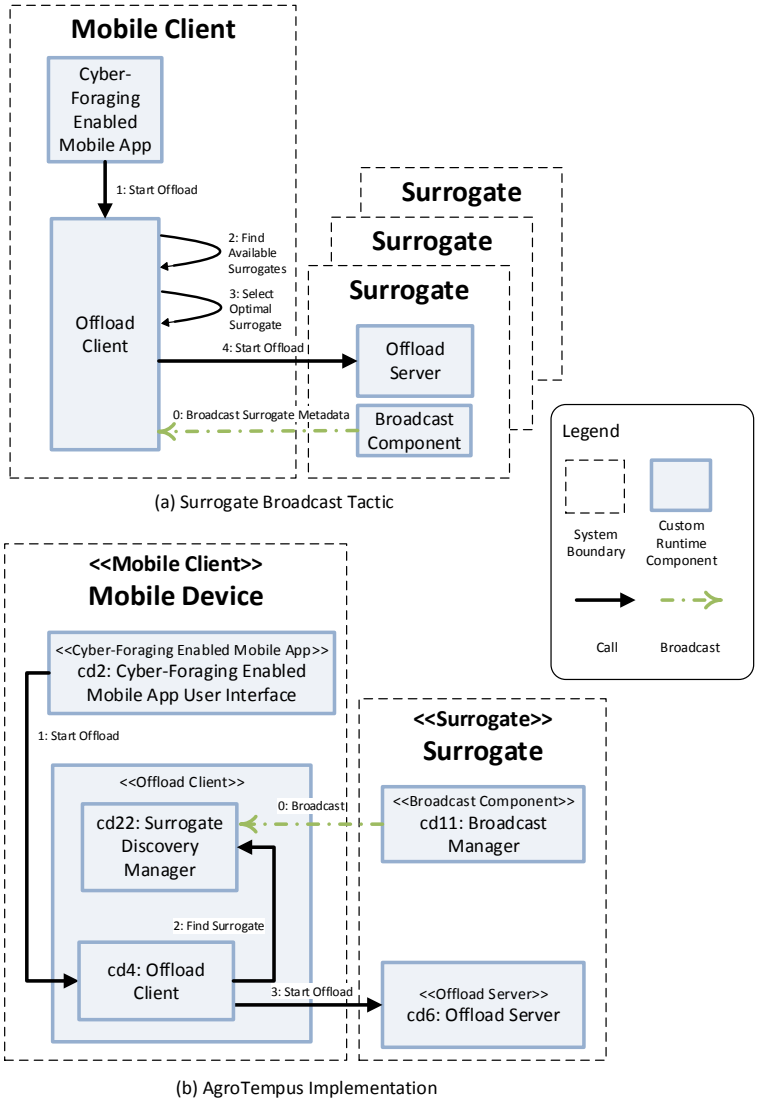

Figure 6.7: Mapping of the AgroTempus Architecture to the Surrogate Broadcast Tactic

11-12. The Data Request Server retrieves the results from the Surrogate Storage Manager.

13. The Data Request Server returns the results to Mobile App Data Exchange Client.

14-16. If the connection to the Mobile Device breaks during the transmission, the results remain on the Surrogate until they can be successfully sent to the Mobile Device.

17. After successful transmission the results associated with the Ticket are deleted from the surrogate.

There are two differences between the AgroTempus architecture and the Cached Results tactic:

1. Because the offloaded computation is expected to be a lengthy operation, the Surrogate always saves the results in the Results Cache instead of attempting the send the results to the Mobile Device immediately.

2. The Surrogate Storage Manager resides outside the Container because it is shared by all offloaded computation and other surrogate components.

6.3.6.7 Client-Side Data Caching

The Client-Side Data Caching tactic is designed in the AgroTempus architecture as shown in Figure 6.9(b), with numbers to indicate the sequence of operations. The component names in Figure 6.9(a) are used as stereotypes for the components in Figure 6.9(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included. The storing of collected data on a mobile device until a surrogate is available takes place as follows:

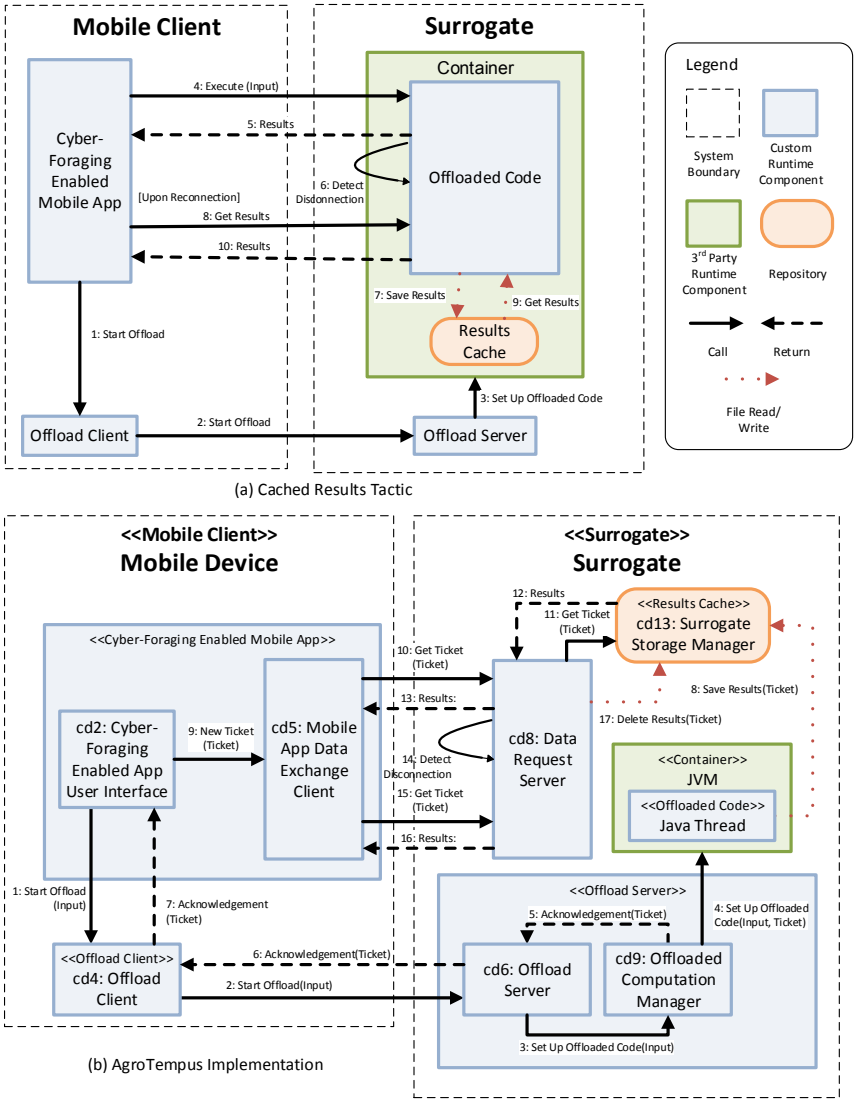

Figure 6.8: Mapping of the AgroTempus Architecture to the Cached Results Tactic

1. The Cyber-Foraging Enabled App User Interface requests the Mobile App Data Exchange Client to add collected weather data its outbound queue.

[Repeat Until Outbound Queue is Empty]

2. The Mobile App Data Exchange client tries to find a surrogate (Section 6.3.6.5).

[If a Surrogate is Found]

3-4. Queued data is sent to the Data Storage Server for storage on the surrogate.

5-6. Is the storage operation is successful the sent data is deleted from the queue.

There are two differences between the AgroTempus architecture and the Client-Side Data Caching tactic:

1. Because the collection of weather data is likely going to be in the field where there will not be a Surrogate in proximity, the Mobile Device always queues the results in the Mobile App Data Exchange Client instead of attempting to send the results to the Surrogate immediately.

2. The Mobile Cache, implemented as a queue, is part of the Mobile Data Exchange Client instead of a separate storage component.

6.3.6.8 Just-in-Time Containers

The Just-in-Time Containers tactic is designed in the AgroTempus architecture as shown in Figure 6.10(b), with numbers to indicate the sequence of operations. The component names in Figure 6.10(a) are used as stereotypes for the components in Figure 6.10(b) to indicate the mapping between components. Only the components that are relevant to the tactic are included. The creation and destruction of containers for offloaded computation takes place as follows:

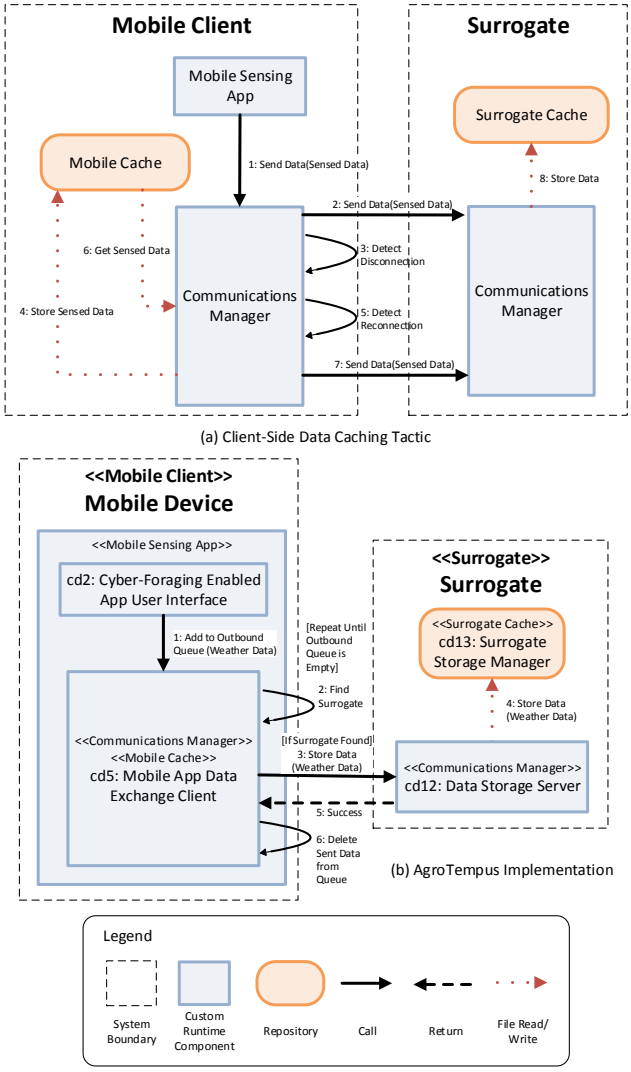

Figure 6.9: Mapping of the AgroTempus Architecture to the Client Side Data Caching Tactic

1-2. The Cyber-Foraging Enabled App User Interface requests to start an offloaded computation with input *Input*.
3. The Offload Server receives the request and invokes the Offloaded Computation Manager.
4. The Offloaded Computation Manager starts the offloaded computation in a separate Java Thread inside the JVM.
5. Upon finishing the execution of the offloaded computation, the thread is terminated, therefore releasing allocated resources.

As with the Computation Offload tactic (Section 6.3.6.1), the main difference between the Just-in-Time Containers tactic and the AgroTempus architecture is that because the offloaded computation is only executed once, the *Input* to the offloaded computation is sent in the initial request to offload.

6.3.7 System Implementation

A demo implementation of the AgroTempus system is available and documented at `http://reuelbrion.github.io/AgroTempus/`. Only the mobile app and surrogate components were developed as part of the demo because this is where the identified tactics are mainly implemented. The mobile hub and cloud components were simulated for the testing and evaluation of the system. The surrogate software was packaged for Raspberry Pi as a Raspbian OS image with an auto-start script. Raspbian is a Linux distribution optimized for Raspberry Pi [105]. The image was tested on a Raspberry Pi 2 Model B with a TP-Link TL-WN722N wireless adapter, as shown in Figure 6.11.

The mobile app (*Mobile Device* components in Figure 6.2) is a Firefox OS app, which is essentially a Web app consisting of HTML pages, CSS style sheets, and Javascript code. Most of the app logic is written in plain Javascript with minimal use of the JQuery library [116].

The surrogate (*Surrogate* components in Figure 6.2) was implemented in Java as a multi-threaded application. The component *CD9: Offloaded Computation Manager* that performs weather data regression and prediction makes use of the Java chart library JFreeChart [91] that offers tools to perform regression on data sets, as well as to generate plot images to visualize the results in common image formats. The same component also makes use of the Apache Commons Codec libraries [115] to convert images generated by JFreeChart into Base64[2] binary string format.

[2]Base64 is a set of binary-to-text encoding schemes commonly used when sending binary data over a network.

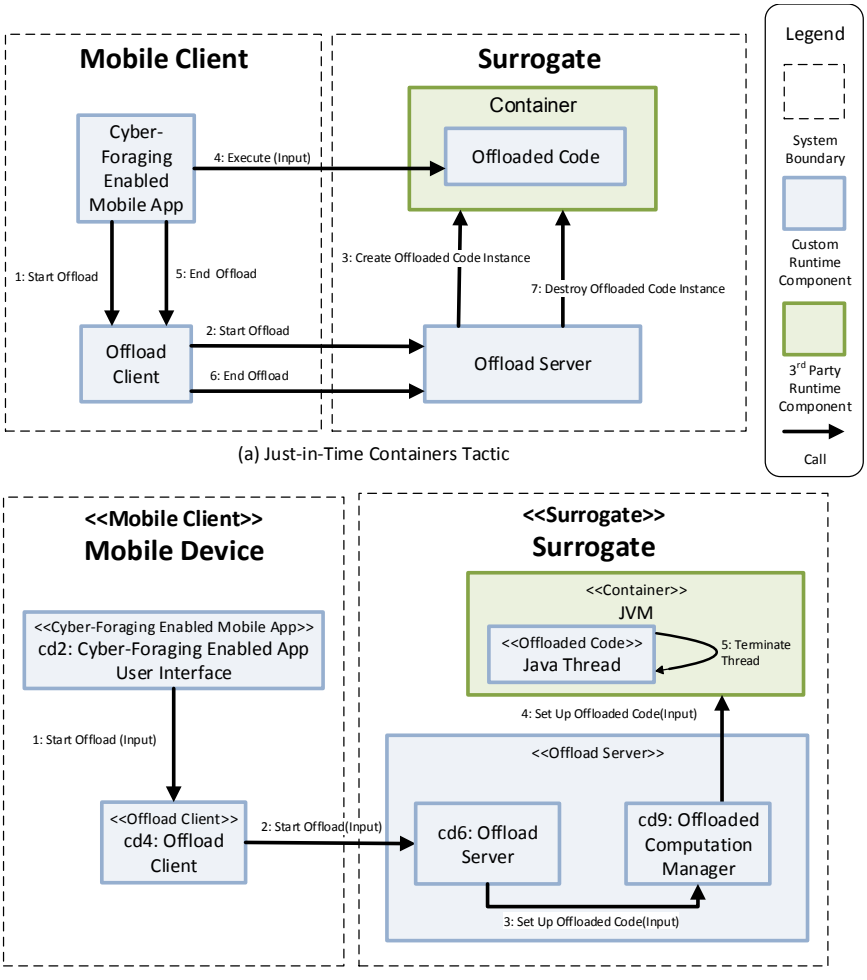

Figure 6.10: Mapping of the AgroTempus Architecture to the Just-inTime Containers Tactic

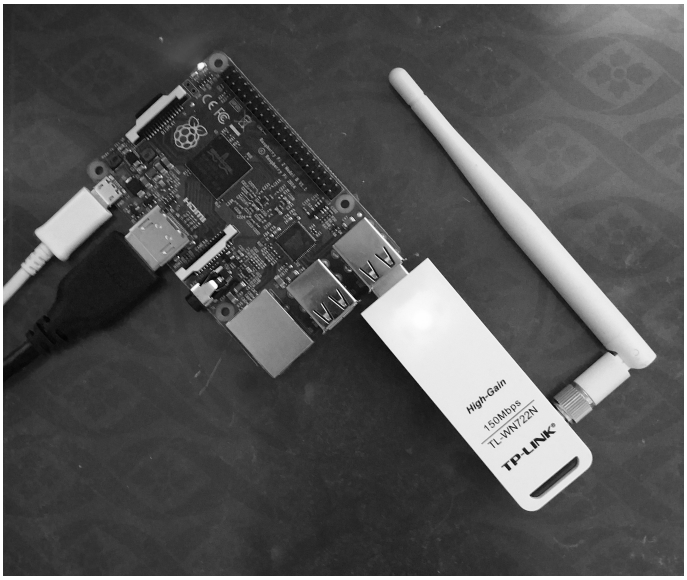

Figure 6.11: AgroTempus Surrogate Setup

For communication between components residing on different nodes, JSON (JavaScript Object Notation) [61] was selected as the standard message and data storage structure. This format is used by free weather APIs such as OpenWeatherMap [94] and works well with Javascript. To be able to use JSON objects in the surrogate code, the system makes use of the JSON.simple toolkit [36]. JSON is also used by IndexedDB, the selected data storage API for FirefoxOS [90].

6.3.8 Analysis

6.3.8.1 System Evaluation

The AgroTempus system implementation included seven of the eight tactics listed in Table 6.1. At implementation time, no working ad-hoc networking library was found for Firefox OS. Therefore, the Surrogate Broadcast tactic could not be used for surrogate discovery in the mobile app. The Local Surrogate Directory tactic (Section 3.2.4.1) was instead used for surrogate discovery. A list of surrogates, including connection details, is maintained on the mobile app. This way, whenever a surrogate service is needed, the mobile app tries

to connect to each surrogate one by one until it can make a connection to a surrogate that provides the needed capabilities.

Extensive testing of the system was performed in order to verify that the implemented system satisfies its intended functional and non-functional requirements. The implementation details for each tactic are detailed below.

The **Computation Offload** tactic was implemented as designed and tested successfully. It is used to perform data regression (FR3) and prediction of future weather values (FR4), two computation-intensive operations. In addition, the generation of the regression chart images is another potentially computation-intensive operation that is also performed on the surrogate. Even though energy consumption was not measured on the mobile device to demonstrate energy efficiency (NFR5), these are two examples of operations that consume and produce small amounts of information compared to their computational requirements, which is known benefit from cyber-foraging [70]. The data regression operation takes as input a weather variable name (Temperature, Humidity, Pressure or Wind Speed), regression type (currently accepts only Linear, but can be easily extended to support other types such as Logistic and Polynomial), a start date, and the number of days to extrapolate, and produces a graph (PNG image) showing all the data points and the regression line. The weather value prediction operation has a weather variable name as input and produces a list of predictions for the variable for the next 7 days. Given that the mobile devices that the AgroTempus app is intended to run on are low-end smartphones with limited computing and storage capabilities, the Raspberry Pi surrogate, although limited as well, still offers more computational power and data storage to increase the capacity of the system (NFR6). The smartphone used for test and evaluation was a ZTE Open C 4.0 with an MSM8210 Dual-Core 1.2GHz CPU and 512MB RAM [131]. The Raspberry Pi 2 Model B has a 900MHz quad-core ARM Cortex-A7 CPU and 1GB RAM [104], and supports SD cards up to 32GB for storage. Given the successful implementation of the tactic as designed, an improvement for the tactics catalog would be to include a variation of the Computation Offload tactic for cases where there is a single request to offload instead of a continued request/response interaction between a mobile device and a surrogate.

The **Out-Bound Pre-Processing** tactic is used for intermediate storage of weather data on the surrogate (FR1) and eventual storage of weather data in the cloud (FR7). It was implemented as designed between the mobile device and the surrogate. Data captured on the mobile device was successfully transmitted and stored on the surrogate. Transmission of the weather data to the mobile hub and eventual storage on the cloud was simulated. As indicated in the evaluation of the Computation Offload tactic, data storage on the surro-

gate is larger than what is available on the mobile device, therefore increasing the storage capability of the system (NFR6). In addition, as will be described in the implementation of the Client-Side Data Caching tactic, weather data is deleted on the mobile device after successful transmission to the surrogate to also increase storage capacity. Although not tested end-to-end with real data, there is potential for the Out-Bound Pre-Processing tactic to implement more than one level of data staging as long as the client and surrogate roles are replicated across levels. An improvement for the catalog would be to include a variation of the Out-Bound Pre-Processing tactic for multi-level data staging.

The **Pre-Fetching** tactic was simulated in the demo implementation by loading a static set of weather data on the surrogate at startup time and tested successfully. The data was used and retrieved by the mobile app (FR2). Because of the lack of a mobile hub and cloud implementation, the complete fetching of data from the cloud to the surrogate (FR9) was not tested. However, the implementation of the fetch and store capabilities implemented in surrogate components *CD8: Data Request Server* and *CD13: Surrogate Storage Manager* would be equivalent to the discover and store capabilities on the mobile hub that would act as an intermediary between the cloud and the surrogate (*CD17: Cloud Synchronization Client* and *CD19: Mobile Hub Storage Manager*). As indicated in the evaluation of the previous two tactics, data storage on the surrogate is larger than what is available on the mobile device, therefore increasing the storage capability of the system (NFR6). Similar to the Out-Bound Pre-Processing tactic, there is potential for the Pre-Fetching tactic to implement more than one level of data staging as long as the client and surrogate roles are replicated across levels. An improvement for the tactics catalog would be to include a variation of the Pre-Fetching tactic for multi-level data staging.

The **Pre-Provisioned Surrogate** was implemented as designed and tested successfully. It enables all the functional requirements of the system, except for the voice interface (FR8) which was not implemented in the demo. All offloaded computation (short and long operations) is loaded on the surrogate upon setup and is packaged inside a Raspbian OS image with auto-start capabilities, as mentioned earlier, to support ease of deployment (NFR2). This same auto-start capability enables surrogate recovery after crashes (NFR9). Similar to the GigaSight system implementation of the Pre-Provisioned Surrogate tactic (Section 5.3.5.1), the AgroTempus implementation confirms that an improvement for a future version of the tactic would be to mark the *Capabilities Metadata* and *Capability Registry* components as optional because they are not necessary when capabilities are not advertised.

The **Surrogate Broadcast** tactic was not implemented in the AgroTem-

pus system as indicated earlier. The **Local Surrogate Directory** tactic was used for surrogate discovery and implemented as indicated in the tactic. Ease of deployment (NFR2) is not as strongly supported by this tactic as would have been with the Surrogate Broadcast tactic. In the current implementation the list of surrogates is hard-coded in the mobile app. The original intent was to include surrogate metadata in a QR code on a sticker that would be placed on the surrogate. A mobile device that would want to make use of the surrogate would read the QR code, which would add the metadata to the list of available surrogates. However, as of the time of implementation, there were no QR libraries available for Firefox OS. Even though it was not tested with a mobile hub, there are multiple options for surrogate broadcast for Java which could be used by the surrogate to broadcast its presence to the mobile hub, such as the ZeroConf protocol used by the Tactical Cloudlets system (Section 4.3.4.3). To satisfy the performance requirement (NFR8), once a surrogate is contacted by a mobile hub, all running threads would be suspended until synchronization with the mobile hub is complete.

The **Cached Results** tactic was implemented in the surrogate as designed and tested successfully. Results of the data regression (FR2) and weather value prediction (FR4) operations are always stored on the surrogate and not sent to the mobile device until requested in order to support fault tolerance (NFR1). This is in case the mobile device moves out of the range of the surrogate before the computation completes. The results are saved until the mobile device connects to the surrogate, therefore promoting availability (NFR7). The change made in the design to always saves results on the surrogate when offloaded operations are expected to be lengthy, instead of attempting to send results to the mobile device immediately, could be added as a variation of the Cached Results tactic.

The **Client-Side Data Caching** tactic was implemented as designed and tested successfully. Data captured in the field (FR1) is stored on the mobile device until a surrogate is available, to promote fault tolerance (NFR1). The results are saved on the mobile device until it can connect to a surrogate, therefore promoting availability (NFR7). Similar to the Cached Results tactic, the change made in the design to always queue the results instead of attempting to send the results to the surrogate immediately could be added as a variation of the Client-Side Data Caching tactic.

The **Just-in-Time Containers** tactic was implemented as designed and tested successfully. When data regression (FR3) and prediction of future weather values (FR4) are offloaded, the system starts the computation in a separate thread, which is destroyed upon completion, therefore increasing the available capacity of the system (NFR6). In addition, because the computation

only runs upon request, energy is saved on the surrogate (NFR5).

Based on this analysis, nine of the ten functional requirements were successfully supported through one or more of the available tactics, as shown in Table 6.1. The Voice Interface requirement (FR8) was not implemented due to project constraints but also because it was known that it would not be implemented through any of the tactics.

Similarly, seven of the ten non-functional requirements were successfully supported through one or more of the available tactics, as also shown in Table 6.1. The usability requirement to support multiple languages (NFR3), similar to the voice interface requirement, was not implemented due to project constraints, but also because it was known that it would not be implemented through any of the tactics. The extensibility requirement to support the development of new services (NFR4) was partially implemented outside of the tactics, through the initial implementation of the project website that contains the mobile app and surrogate code, as well as documentation (`http://reuelbrion.github.io/AgroTempus/`). The current documentation needs to be augmented to fully support the requirement by providing more detailed guidance to developers (e.g., location of extension points, templates for new services). Finally, the data integrity requirement to provide data checks (NFR10) was not implemented due to project constraints, but could be easily be implemented outside of the tactics through input validation code in the user interface components.

6.3.8.2 Developer Observation and Feedback

Throughout the process we met with the developer once a week to check on project status and observe how the tactics were being used. The general development process that was followed is consistent with the structure of this section: (1) requirements elicitation, (2) mapping of requirements to tactics, (3) architecture, (4) mapping of components to architecture, (5) design, (6) implementation, and (7) testing and evaluation. Because of the nature of the case study, the developer was asked to document the project during the entire process.

The developer found the tactics easy to understand and use. The most difficult part for the developer was determining, based on the tactics, which of the components would be needed to implement the requirements. Feedback for a future version of the tactics is to provide differentiation between core and optional components of the tactic, consistent with the findings from the previous two case studies. Another recommendation from the developer was to include sample code and potentially a list of libraries/platforms that can

be used to implement common requirements of cyber-foraging systems. The inclusion of sample code with the tactics is consistent with the feedback from the main developer of the GigaSight system (Section 5.3.5.3).

The developer also found the tactics to be useful in the development of the system. As stated by the developer: "The models that were used as a blueprint during development were in large constructed from the tactics; they were instrumental in providing a good foundation for the application."

6.3.8.3 Findings

The goal of the case study was to discover (1) which of the architectural tactics for cyber-foraging can be used in the development of the AgroTempus system to fulfill its functional and non-functional requirements, and (2) how do the selected tactics support their intended functional and non-functional requirements. The context for the case study is the validation of the tactics in real systems.

Eight tactics were identified to satisfy system requirements, of which seven were implemented in the system, and one had to be replaced by an alternative tactic due to a technology constraint. As in the Tactical Cloudlets and GigaSight systems discussed in the two previous chapters, tactics were identified and implemented to satisfy the main functional requirements for a cyber-foraging system, as presented in Section 3.2:

- Pre-Provisioned Surrogate was used for surrogate provisioning
- Local Surrogate Directory was used for surrogate discovery (replacing Surrogate Broadcast)
- Computation Offload was used to implement computation offload capabilities
- Out-Bound Pre-Processing and Pre-Fetching were used to implement data staging capabilities

Two fault tolerance tactics, Cached Results and Client-Side Data Caching, were used to satisfy fault tolerance and availability requirements. Just-In-Time Containers, a resource optimization tactic, was used to satisfy surrogate capacity and energy efficiency requirements.

All the tactics were implemented as designed, but there were several changes that were made at design time to better fulfill requirements. Even though the essence of each tactic remained the same, these changes create opportunities for improvement of the tactics catalog. In particular, variations to the

the Computation Offload, Out-Bound Pre-Processing, Cached Results, and Client-Side Data Caching tactics were identified.

The case study shows that there are different ways to implement tactics, mainly determined by system constraints and assumptions, but also by mobile device and surrogate computing power and specifications, as well as usage contexts. For example, VMs are used as data and computation containers in the Tactical Cloudlets and GigaSight systems (Chapters 4 and 5) because of the flexibility that they provide, but also because the surrogates are expected to be high-end servers. For the AgroTempus system the selection of using JVMs as computation containers is a better choice because they have less overhead and consume less resources on the machine. They do not provide the flexibility of VMs, but this is not required in the more static usage context of AgroTempus.

The case study also showed that technology selection can sometimes be a barrier to the use of tactics and therefore effective satisfaction of requirements. The use of Firefox OS as the mobile device operating system did not allow the implementation of the Surrogate Broadcast tactic because of the lack of libraries for discovery in this platform. In addition, the lack of libraries for QR code reading also affected the ease of deployment requirement that was associated to the Local Surrogate Directory tactic that replaced that Surrogate Broadcast tactic for surrogate discovery. These technology insights that are gained from the implementation and evaluation of cyber-foraging systems could be added as notes to the tactics to provide even greater value to software architects.

Finally, as more real cyber-foraging systems are deployed, more tactics and non-functional requirements will emerge. For example, recovery was not a requirement that was identified as part of the SLR (Chapter 2) on architectural tactics for cyber-foraging. However, it is highly likely that this will be a requirement for cyber-foraging systems in resource-challenged environments, such as the AgroTempus usage context. Recovery in the AgroTempus system was implemented via the use of Java threads combined with a monitoring capability. Because service instances run in separate threads after the initial connection, a failed service thread will not affect the main service thread. Passing data between threads happens through thread-safe queues (java.util.concurrent.
ConcurrentLinkedQueue). The main surrogate process periodically checks whether all service threads are alive, and crashed threads are restarted. A generalization of this approach could easily be codified as a Surrogate Recovery tactic.

In summary, the tactics were successfully used to create an architecture and

implementation of the AgroTempus system that fulfills most of its relevant requirements, which answers the research questions for the case study. In combination with the utility statement from the system developer, this serves as a validation of the tactics for development of cyber-foraging systems for computation offload and data staging.

6.3.9 Threats to Validity

There are two main threats to validity of the results of this case study. The first is related to *internal validity* because the data collection and analysis was conducted by a single researcher and therefore subjective interpretations might exist. To mitigate this threat, data collected from several sources (evolving system documentation, the code base, and ongoing developer interviews) was confirmed by the developer such that we could have immediate feedback. The second threat is related to *external validity*, specifically whether the findings are generalizable given that the results are drawn from the development of a single system. To mitigate this threat we conducted two additional case studies that are reported in Chapters 4 and 5. The results of this case study are consistent with the previous two case studies, with a confirmation from the developer on the usefulness of the tactics to build cyber-foraging systems.

6.4 Conclusions

This chapter presented the results of the last of three case studies to validate the architectural tactics for cyber-foraging presented in Chapter 3, in the context of RQ2, which is to identify the architectural tactics that can be derived from the architectural design decisions identified by the SLR.

For this case study two research questions were defined for the development of a cyber-foraging system for computation offload and data staging.

1. Which of the architectural tactics for cyber-foraging can be used in the development of the AgroTempus system to fulfill its functional and non-functional requirements?

The analysis of the AgroTempus system resulted in the identification of eight architectural tactics, seven of which were implemented in the system. One tactic had to be replaced due to technology constraints. In addition, elements of these tactics were also used to meet energy efficiency, ease of deployment, and performance requirements. The recovery requirement was implemented via a mechanism that could easily be codified as a new tactic,

especially applicable to cyber-foraging systems in resource-constrained environments. In addition, several tactic variations were identified.

Although based on the analysis of a single system, the results show that a tactics-driven approach can be used for the development of cyber-foraging systems for computation offload and data staging.

2. How do the selected tactics support their intended functional and non-functional requirements?

System testing shows that the implemented tactics meet their intended functional and non-functional requirements. As indicated by the developer of the AgroTempus system, the architectural tactics constituted a strong foundation for the development of the system.

The variety of the usage contexts explored in only these three case studies shows that there is potential for many uses of cyber-foraging systems. The next chapter identifies the usage contexts that benefit from cyber-foraging, along with the functional and non-functional requirements that drive systems development.

6.5 Acknowledgments

Very special thanks to Reuel Brion from VU University Amsterdam for his invaluable support during the execution of this case study.

7

Characterization of Cyber-Foraging Usage Contexts

This chapter addresses research question RQ3 and presents a characterization of usage contexts for cyber-foraging defined in terms of functional and non-functional requirements for cyber-foraging systems. The goal of the characterization is to provide context for software engineering life cycle activities for cyber-foraging systems, such as requirements engineering, software architecture, and quality assurance, with the intent of developing systems that fully realize the benefits of cyber-foraging.

7.1 Introduction

Surrogate-based cyber-foraging enables mobile devices to extend their computing power and storage by offloading computation or data to more powerful servers located in in single-hop proximity (i.e., surrogates). There are many domains and applications that can benefit from the longer battery life and better application performance on mobile devices that is typically associated to the use of cyber-foraging, such as field operations, sensor systems, and entertainment. However, obtaining these benefits in operational systems requires meeting functional and non-functional requirements that vary depending on the usage context of the cyber-foraging system.

This chapter presents the characterization of the usage domains and contexts that benefit from surrogate-based cyber-foraging, defined in terms of functional and non-functional requirements. The goal of this characterization is to provide context for software engineering life cycle activities for cyber-foraging systems, such as requirements engineering, software architecture, and

quality assurance, with the intent of developing systems that fully realize the benefits of cyber-foraging.

The next section describes the analysis that led to the characterization of usage contexts for cyber-foraging. Sections 7.3, 7.4, and 7.5 contain the details of each identified usage context. Finally, Section 7.6 summarizes and concludes the chapter.

7.2 Analysis

In Chapter 2 we presented the results of a systematic literature review (SLR) on architectures for cyber-foraging. Common design decisions present in the cyber-foraging systems described in the primary studies were codified into architectural tactics for cyber-foraging and then grouped into functional and non-functional architectural tactics as shown in Chapter 3.

To identify cyber-foraging usage contexts we started with the same set of primary studies identified in the SLR. In the first phase, for each primary study we extracted the names of the environments and types of applications that were being targeted in the cyber-foraging systems described in each study, either as examples or case studies. We then clustered these results based on similarity. The results of the mapping between usage contexts and primary studies is shown in Table 7.1. The first column contains the name given to the cluster of studies that defines the usage context. The second column contains the set of the applications and domains used in the primary studies as examples or case studies. The last column contains the names of the cyber-foraging systems from the primary studies.

In the second phase we revisited the primary studies in each usage context extracting functional requirements (FRs) and nonfunctional requirements (NFRs) explicitly and implicitly stated in each study, with the goal of identifying recurring requirements in each usage context. Each FR and NFR that was stated in at least three of the primary studies was considered a recurring requirement. The exception is the *Mobile Applications in Hostile Environments* usage context which only has two studies, in which case we considered it recurring if it was stated in both studies.

The identified FRs and NFRs for each usage context are shown in the conceptual model in Figure 7.1, inspired by UML class diagrams and the inheritance relationship. The rectangles with the rounded top corners represent a *context characterization* and include FRs and NFRs that are common across more than one usage context. The rectangles marked with *UC#* represent the usage contexts derived from Table 7.1 and include FRs and NFRs that

are unique to that usage context. Each usage context inherits FRs and NFRs from context characterizations and other usage contexts, as defined by the inheritance relationship between elements. The FRs and NFRs are labeled in order to facilitate the mapping to the description of the usage contexts in the next section, along with the benefits of cyber-foraging for the usage context and the constraints for obtaining these benefits. Some FRs, such as *FR1*, appear in several context characterizations and usage contexts. In this case, the inheriting element is "overriding" the FR with specific details for the context characterization or usage context. As a reference, Table 7.2 included at the end of this chapter contains a summary of recurring functional and non-functional requirements for each usage context.

Table 7.1: Cyber-Foraging Usage Contexts: Mapping of Primary Studies

Usage Context	**Example Applications and Domains**	**Systems in Primary Studies**
Computation-Intensive Mobile Applications (Short Operations)	Image, audio and video processing and manipulation Face detection and recognition Speech recognition Speech translation Antivirus/Anti-malware Gaming (AI-based)	Chroma [9] Computation and Compilation Offload [18] Cloud Media Services [19] CloneCloud [22] HPC-as-a-Service [30] OpenCL-Enabled Kernels [33] Real Options Analysis [35] Collective Surrogates [48] Virtual Phone [55] Single-Server Offloading [56] Android Extensions [57] ThinAV [59] Cuckoo [62] ThinkAir [64] MACS [65] Scavenger [67] AMCO [72] MCo [74] PowerSense [82] AIDE [83] PARM [88] Resource Furnishing System [92] SOME [96] SmartVirtCloud [100] MAPCloud [103] VM-Based Cloudlets [108] IC-Cloud [111] AIOLOS [117] Heterogeneous Auto-Offloading Framework for Mobile Web Browsers [128] Weblets [127] DPartner [129] Elastic HTML5 [126]
Mobile Applications in Low Coverage Environments	Resource-challenged environments Field operations (e.g., researchers, medics, sales and marketing)	Mobile Agents [5] Edge Proxy [6] Mobile Information Access Architecture for Occasionally Connected Computing [8] MAUI [26] 3DMA [39] Spectra [41]

Continued on next page

Table 7.1 – *Continued from previous page*

Usage Context	**Example Applications and Domains**	**Systems in Primary Studies**
Computation-Intensive Mobile Applications (Long Operations)	Service-based applications Workflow-based applications Search-based applications (discrete tasks or single replicated task)	Cloud Operating System to Support Multi-Server Offloading [56] Odessa [101] SPADE [112] Offloading Toolkit and Service [121]
Mobile Applications in Hostile Environments	Emergency response Military operations	Cloudlets [52] Application Virtualization on Cloudlets [84]
Public Surrogates	Everyday use	Collaborative Applications [16] Roam [20] Trusted and Unmanaged Data Staging Surrogates [42] Slingshot [114]
Sensing Applications	Healthcare Intelligent transport systems Ambient intelligence Environmental monitoring Context-aware applications Participatory sensing (Crowdsensing)	mHealthMon [2] C2C [7] Grid-Enhanced Mobile Devices [51] Feel The World [98] Smartphone-Based Social Sensing [102] Large-Scale Mobile Crowdsensing [119] Sonora [120] Mobile Data Stream Application Framework [122]
Data-Intensive Mobile Applications	Mobile cloud applications Online gaming Data-rich domains	Kahawai [26] AlfredO [47] Telemedik [71] Cloud Personal Assistant [93]

7.3 Cyber-Foraging Usage Contexts

As shown in the *General* context characterization in Figure 7.1, cyber-foraging systems in all usage contexts need to satisfy three non-functional requirements.

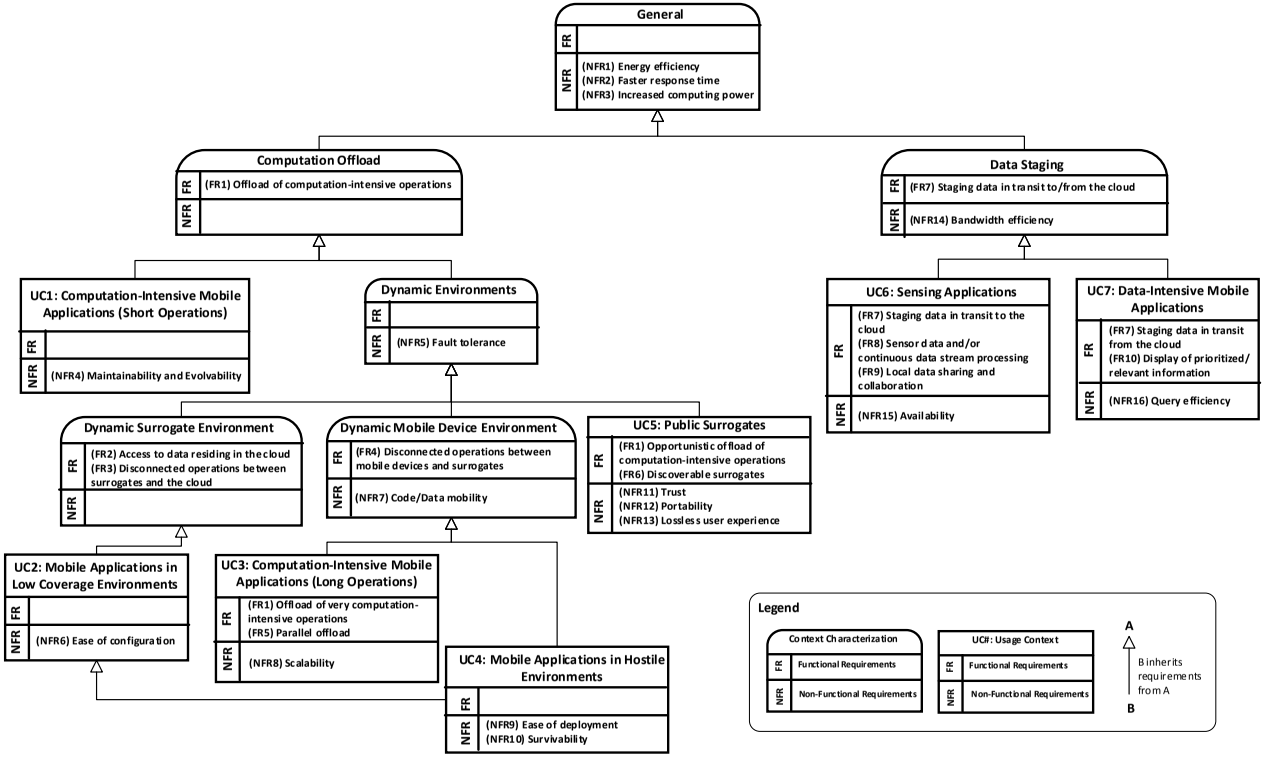

Figure 7.1: Conceptual Model for Cyber-Foraging Usage Contexts

(NFR1) Energy efficiency: Offloading computation should consume less energy than local execution based on the premise that offloading is beneficial when large amounts of computation are needed with relatively small amounts of communication [70].

(NFR2) Faster response time: Offloading computation should lead to a faster response time that local execution.

(NFR3) Increased computing power: Offloading computation and data should take advantage of the greater computing power of surrogates.

Depending on whether the main goal of the usage context is computation offload or data staging, additional FRs and NFRs need to be satisfied, as shown in the following sections.

7.4 Computation Offload Usage Contexts

Cyber foraging systems that perform computation offload need to satisfy a general requirement related to the offload operation, as shown in the *Computation Offload* context characterization in Figure 7.1.

(FR1) Offload of computation-intensive operations: A cyber-foraging-enabled application, upon encountering computation-intensive code explicitly marked for offload, determines if the conditions are appropriate for offload (e.g., surrogate availability, network conditions, remaining battery). If so, the mobile device locates a surrogate for offload, offloads the computation, and waits for a response from the surrogate.

7.4.1 Usage Context 1: Computation-Intensive Mobile Applications (Short Operations)

The systems in this usage context are mobile applications that contain computation-intensive operations which if executed on a mobile device would take in the order of tens of seconds, but if offloaded could improve response time considerably. These are typically request-response, synchronous operations such as:

- Image, audio, and video processing and manipulation
- Face detection and recognition
- Speech recognition and translation
- Virus and malware detection
- Gaming algorithm execution (typically AI-based)

As shown in the usage context *UC1* in Figure 7.1, in addition to FR1, NFR1, NFR2, and NFR3, cyber-foraging systems in this usage context need to satisfy a functional requirement related to maintainability and evolvability.

> **(NFR4) Maintainability and Evolvability:** Systems may perform a runtime decision to offload. In this case, two versions of the same code (local and remote) need to be maintained and evolved over time.

Benefits

The main benefit of cyber-foraging in this usage context is augmented execution capability due to computation offload (FR1) to more powerful resources (NFR3). Computation offload also reduces battery consumption (NFR1) which leads to longer battery life and provides better response times (NFR2) due to offload to proximate resources instead of remote cloud resources [10].

Constraints

Systems that make runtime decisions in this usage context to execute locally or remotely have the advantage of additional battery savings because offload only occurs when conditions are conducive to battery savings based on code characteristics, surrogate availability, and environment conditions (e.g. network quality, available bandwidth). Also, because operations take seconds to execute, restarting an operation locally due to a disconnected surrogate may not have a large negative effect on user experience if recovering does not exceed an acceptable wait time and the user is informed of the situation.

However, care has to be given to maintainability and evolvability (NFR4) because it is likely that two versions of the code have to be maintained: one for the mobile device and one for the surrogate. If not managed carefully it can lead to increased effort in parallel code maintenance and evolution.

7.4.2 Dynamic Environments

Cyber-foraging systems often operate in dynamic environments where connectivity between mobile devices and surrogates, or between surrogates and the cloud, cannot be guaranteed. These systems need to be able to detect and react to periods of disconnection, therefore requiring fault tolerance mechanisms as shown in the *Dynamic Environments* context characterization in Figure 7.1.

> **(NFR5) Fault tolerance:** Mobile devices leveraging surrogates, and surrogates connected to the cloud, should be able to detect and react appropriately to periods of disconnection.

In situations in which connectivity between the surrogates and the cloud cannot be guaranteed, there is a need for surrogates to continue supporting the computational and data needs of mobile devices even during periods of disconnection, as shown in the *Dynamic Surrogate Environment* context characterization in Figure 7.1.

> **(FR2) Access to data residing in the cloud:** Surrogates serve as caches for data located in the cloud that is required by mobile applications.

> **(FR3) Support for disconnected operations between surrogates and the cloud:** Surrogates should contain data that is required by mobile applications, and take advantage of available connectivity to the cloud to synchronize with master data sources and cache data that might be required given changes in context, user preferences, or user actions.

In situations in which connectivity between the mobile devices and surrogates cannot be guaranteed, there is a need for surrogates to save results of offload operations until connectivity is restored, or where computation can move as mobile devices move, as shown on the *Dynamic Mobile Device Environment* context characterization in Figure 7.1.

> **(FR4) Support for disconnected operations between mobile devices and surrogates:** If a mobile device loses contact with the surrogate before it can obtain a result, the surrogate should save the results until the mobile device is reachable.

(NFR7) Code/Data mobility: If multiple connected surrogates are available the system should be able to move code and data to other surrogates to fulfill application needs and continuity of operations.

7.4.2.1 Usage Context 2: Mobile Applications in Low Coverage Environments.

Low coverage environments are characterized by disconnection, or occasional connectivity, between surrogates and the cloud, but potentially good connectivity between mobile devices and surrogates. Examples of applications and domains include:

- Resource-challenged environments: Less-privileged regions characterized by limited Internet access, limited electricity and network access, and potentially low levels of literacy can leverage surrogates, deployed in, for example, kiosks, to obtain information to support their communities.
- Field operations: People that spend time away from their main offices or labs, such as researchers, medics, and sales personnel, can leverage portable surrogates to support their computation and data needs.

As shown in the usage context *UC2* in Figure 7.1, in addition to FR1, FR2, FR3, NFR1, NFR2, NFR3, and NFR5, cyber-foraging systems in this usage context need to satisfy a functional requirement related to ease of configuration.

(NFR6) Ease of configuration: Surrogates should contain capabilities that enable administrators to load surrogates with the computation and data needed to support the mobile applications that will be using it, especially in areas where there might not be technical staff available.

Benefits

Mobile applications in low coverage environments can benefit from cyber-foraging in the following ways:

- Augmented execution due to computation offload (FR1) to more powerful resources (NFR3). In the case of resource-challenged environments, surrogates can execute computation-intensive operations such as speech, image or gesture recognition as alternate forms of input to account for low levels of literacy.

- Reduced battery consumption (NFR1) due to offload of computation-intensive operations which leads to longer battery life, especially in environments where recharging mobile devices is difficult.
- Better response times (NFR2) as well as lower energy consumption (NFR1) due to offload to proximate surrogates instead of remote cloud servers.
- Pre-provisioned surrogates (FR2) can carry all computation and data that is needed by surrogate users and can function disconnected from the cloud (FR3).

Constraints

The benefits of cyber-foraging are only possible if surrogates are properly pre-provisioned, that is, they contain all the data and computation required by the mobile applications that use them. Processes that predict computation and data usage based on user profiles, workflows, or access history are necessary to support ease of configuration (NFR6).

In addition, cyber-foraging systems operating in low coverage environments require fault tolerance (NFR5) mechanisms to be able to detect periods of connection and disconnection between surrogates and the cloud and seamlessly switch between operating in connected and disconnected mode. Surrogates should continue supporting mobile applications when disconnected from the cloud, even if in degraded mode.

7.4.2.2 Usage Context 3: Computation-Intensive Mobile applications (Long Operations).

The systems in this usage context are mobile applications that contain computation-intensive operations which if executed on a mobile device would take minutes to hours, but if offloaded could improve response time considerably. In most cases there is not an option for local execution given the computing requirements of the offloaded operations. These requirements are likely greater than what is available locally, or would drain the battery before returning a result. The types of applications that contain long operations — that are also typically asynchronous to avoid blocking — include:

- Service-based applications: Applications that are composed of a number of possibly independent services which may perform long operations.
- Workflow-based applications: Applications that execute a workflow that may include steps that are long-running, such as business applications in which the mobile application initiates a long-running business process.

- Search-based applications: Applications that require searching through large data sets, such as data analytics applications or applications that combine data from different sources. These applications can be composed of discrete tasks or single replicated tasks (i.e., executing the same search against different data sources).

While systems in this usage context still need to satisfy FR1 (computation offload), it would have to be redefined as *Offload of very computation-intensive operations*, as shown in *UC3* in Figure 7.1. What this means is that upon encountering very computation-intensive code marked for offload, the mobile application locates a surrogate for offload, offloads the code, and either waits for a response from the surrogate (synchronous) or is notified by the surrogate that the operation is complete (asynchronous, i.e., at a later time and without blocking the application).

In addition to the redefined FR1, FR4, NFR1, NFR2, NFR3, NFR5, and NFR7, cyber-foraging systems in this usage context need to satisfy additional requirements related to parallel offload, if this option is available.

> **(FR5) Parallel offload:** If surrogates are connected to other surrogates and operations are parallelizable, the cyber-foraging system should attempt to leverage the combined computing power of the set of available surrogates.

> **(NFR8) Scalability:** If multiple connected surrogates are available, and offloaded operations are parallelizable, the system should be able to determine the optimal amount of surrogates to utilize for execution of the offloaded computation.

Benefits

Mobile applications that contain long computation-intensive operations can benefit from cyber-foraging in the following ways:

- Augmented execution capability due to computation offload (FR1) to more powerful resources (NFR3)
- Reduced battery consumption (NFR1) due to offload of long computation-intensive operations which leads to longer battery life
- Better response times (NFR2) as well as lower energy consumption (NFR1) due to offload to proximate resources instead of remote cloud resources.

Constraints

Long computation-intensive operations may require the resources of more than one surrogate in order to achieve the benefits of cyber-foraging. If possible, due to parallelization of these long computation-intensive operations, multiple connected surrogates would need to implement load balancing for scalability (NFR8, FR5). However, load balancing requires moving computation and data between surrogates, which in turn requires execution containers such as virtual machines that support code and data mobility (NFR7).

In addition, given that a mobile device may lose contact with a surrogate before the operation finishes, mechanisms such as caching data until the mobile device is reconnected, or using alternative communication mechanisms to reach the mobile device (e.g., SMS) are necessary (FR4). A user should be informed when this happens so that he/she knows that the results will not be available until reconnection (NFR5).

7.4.2.3 Usage Context 4: Computation-Intensive Mobile Applications in Hostile Environments.

Hostile environments, such as those in which emergency responders or military personnel operate, are characterized by very dynamic environments in which disconnected operations — or occasionally-connected operations — between surrogates and the cloud, and between mobile devices and surrogates, are highly likely.

In addition to FR1, FR2, FR3, FR4, NFR1, NFR2, NFR3, NFR5 and NFR6, systems in this usage context need to satisfy two non-functional requirements related to the hostility of the environment and potential loss of resources.

> (NFR9) Ease of deployment: It should be easy to deploy surrogates in the field to support a mission (e.g., on vehicles, in tents, or in provisional operations centers).

> (NFR10) Survivability: Surrogates and mobile applications should be able to continue operating in spite of disruptions caused by the operational environment.

Benefits

Mobile applications in hostile environments can benefit from cyber-foraging in the following ways:

- Augmented execution capability due to computation offload (FR1) to more powerful resources (NFR3)
- Reduced battery consumption (NFR1) due to offload of computation-intensive operations which leads to longer battery life, especially in these environments where recharging mobile devices may be difficult.
- Better response times (NFR2) as well as lower energy consumption (NFR1) due to offload to proximate resources instead of remote cloud resources.
- Pre-provisioned surrogates (FR2) can carry all data that is needed by surrogate users executing a mission and can function disconnected from the cloud (FR3).
- Runtime partitioning can be based on a basic algorithm that simply detects surrogate availability such that operations execute locally if a surrogate is not available (FR4).
- In case of disconnection, surrogates can cache offload operation results (FR4) until the mobile device is reconnected.
- If computation is self-contained (e.g., in a VM) and more than one surrogate is available, computation can migrate between surrogates due to mobile device mobility (i.e., mobile device moves beyond the range of a surrogate) and/or surrogate mobility (e.g., in case surrogates reside in vehicles) (FR4, NFR7).

Constraints

The benefits of cyber-foraging are only possible if surrogates are properly pre-provisioned, that is, they contain all the data and computation required by the mobile applications that use them. Processes that predict computation and data usage based on mission profiles, user profiles, workflows, or access history are necessary to support ease of configuration (NFR6).

In addition, cyber-foraging systems operating in hostile environments require fault tolerance mechanisms (NFR5) to be able to detect periods of connection and disconnection, and seamlessly switch between operating in connected and disconnected mode. Because of the uncertainty of connections between mobile devices and surrogates, fallback to local execution is required in case of unavailable surrogates or disconnection during offload operations (FR4).

Finally, hostile environments require systems to continue operating in spite of the uncertainly of the environment in order to ensure the success of missions.

Mechanisms to ensure ease of deployment (NFR9) and configuration (NFR6) such as self-contained capabilities and management consoles can support quick setup of surrogates and capabilities to support a mission. In addition, mechanisms that promote survivability (NFR10) such as multiple discoverable, connected surrogates that can load balance or transfer offloaded computation in case of disconnection (NFR7), are key to reaching the benefits of cyber-foraging in these environments.

7.4.2.4 Usage Context 5: Public Surrogates.

Publicly-available surrogates on which any user can offload computation-intensive operations is a vision for cyber-foraging cited by the studies included in Table 7.1 for this usage context, as well as Balan et al [9]: "*Although deployment of compute servers for public use is not imminent, our work addresses future environments where they may be as common as water fountains, lighting fixtures, chairs or other public conveniences that we take for granted today. When public infrastructure is unavailable, other options may exist.*" The goal of mobile applications that leverage public surrogates is seamless mobility, that is, the capability to move code (and data) between mobile devices and surrogates with minimal human intervention.

Although this usage context falls under *Dynamic Environments*, it is different from the other usage contexts in this group because computation offload is opportunistic instead of user-triggered. This is why FR1 is redefined as *Opportunistic offload of computation-intensive operations.* What this means is that upon discovery of an available surrogate, running mobile applications that are determined to be computation-intensive migrate their execution to the discovered surrogate (either manually or automatically). When the mobile device leaves the vicinity of the surrogate (or because of termination actions such as expiration time or manual intervention), the computation on the surrogate migrates back to the mobile device.

In addition to the redefined FR1, cyber-foraging systems in this usage context need to satisfy a functional requirement related to surrogate discovery.

> **(FR6) Discoverable surrogates:** Surrogates should broadcast their presence to cyber-foraging-enabled mobile applications for discovery.

Also, in addition to NFR1, NFR2, NFR3 and NFR5, cyber-foraging systems in this usage context need to satisfy other non-functional requirements related to computation migration between mobile devices and offload to public surrogates.

(NFR11) Trust: When a mobile device discovers a surrogate it expects a trustworthy surrogate execution environment, meaning that once an offload operation starts, code and data are not maliciously modified or stolen, and that it provides trustful services. In the same way, a surrogate expects that a mobile device is a valid client and that it will not offload malicious code or use it as a vehicle to other code and data offloaded by other mobile devices.

(NFR12) Portability: Offloaded computation should be able to run on a variety of surrogate platforms.

(NFR13) Lossless user experience The migration of computation (and data) between a mobile device and a surrogate should cause minimal disruption to a user, other than what is defined in the migration process or protocol (e.g, authentication, manual disconnection).

Benefits

A benefit of cyber-foraging using public surrogates is augmented execution due to opportunistic computation offload (FR1) to more powerful, discoverable resources (FR6, NFR3). Reduced battery consumption (NFR1) due to offload of computation-intensive operations, which leads to longer battery life, is also a benefit. Finally, faster response times (NFR2) as well as lower energy consumption (NFR1) are expected due to offload to proximate, more powerful resources.

Constraints

Offload to public surrogates implies that the mobile user does not own the surrogate. Trust (NFR11) has to be built into the cyber-foraging system such that the mobile user trusts that code and data offloaded to the surrogate is not going to be compromised, and the surrogate trusts that the user will not use it to install malicious code.

In addition, given that the relationship between the mobile device and the surrogate is transient, fault tolerance (NFR5) mechanisms are required to detect when a mobile device is in proximity of a surrogate and when it is not such that it can seamlessly switch between local execution and remote execution (NFR13).

Finally, in public surrogates there is likely no control over their configuration. Portability of offloaded code and data (NFR12) is required in order to adapt to multiple execution environments. Virtual machines as execution

containers would be a good match for public surrogates.

7.5 Data Staging Usage Contexts

Cyber foraging systems that perform data staging need to satisfy two general requirements related to data staging and efficiency, as shown in the *Data Staging* context characterization in Figure 7.1.

> **(FR7) Staging data in transit to/from the cloud:** Surrogates should act as intermediate data caches between mobile devices and the cloud.

> **(NFR14) Bandwidth efficiency:** Mobile devices should offload data to surrogates, and surrogates should send data to mobile devices, only when conditions are conducive to bandwidth efficiency, such as when network quality is above an established threshold, when network traffic is below an established threshold, or when cached data reaches an established bundle size for sending.

7.5.1 Usage Context 6: Sensing Applications

The systems in this usage context are mobile applications that perform context, environment, or urban sensing using on-board sensors (e.g., camera, microphone, accelerometer) or connected sensors (e.g. gas, ambient temperature). The sensing applications collect data from these sensors and send it to surrogates as these become available. Examples of domains and applications in this usage context include:

- Context-aware applications: A mobile application uses sensors to acquire contextual information and send it to surrogates for processing, such as complex activity or scene recognition
- Healthcare: A mobile application is used by patients carrying body sensors to gather data from these sensors and send it on to surrogates for analysis.
- Intelligent transport systems: A mobile application integrated into a vehicle can obtain readings from multiple sensors and send the data to surrogates located at various points throughout the city to, for example,

perform traffic analysis and control, surveillance, or emergency management.

- Ambient intelligence: Ambient intelligence can be supported by mobile applications that sense contextual data and send it to surrogates for rapid processing to provide personalized, adaptive, and anticipatory services such as ambient control (e.g., lighting, music, temperature) and calendar management.
- Environmental monitoring: Mobile applications equipped with environmental sensors such as gas, pressure, or temperature collect data to send to surrogates for processing for disaster prevention, detection, and response activities.
- Participatory sensing (Crowdsensing): Crowdsensing refers to individuals using mobile devices with sensors that share information about an event or task of interest such as environmental monitoring, public safety, traffic monitoring, or collaborative searches.

In this usage context, surrogates typically act as intermediaries as sensed data flows from the mobile devices to the cloud, which is why FR7 (data staging) needs to be redefined as *Staging data in transit to the cloud* as shown in *UC6* in Figure 7.1. This means that data collected on surrogates is stored for upload to the enterprise cloud when possible.

In addition to the redefined FR7, NFR1, NFR2, NFR3, and NFR14, cyber-foraging systems in this usage context need to satisfy additional requirements related to the role of surrogates as proximate, intermediate data caches.

> **(FR8) Sensor and/or continuous data stream processing:** As surrogates become available, sensor data collected by the mobile device is sent to the surrogate for processing and storage.

> **(FR9) Local data sharing and collaboration:** Surrogates store and process collected data to make it available to mobile devices that they are serving.

> **(NFR15) Availability:** Surrogates should be available for data offload from mobile devices. A corollary to this requirement is that mobile devices need to be able to deal with unavailable surrogates.

Benefits

There are multiple benefits of cyber-foraging for sensing applications:

- Offloading data (FR7) to surrogates releases storage space on mobile devices to continue data collection activities (the surrogate storage can be considered an extension to mobile device storage (NFR3)).
- Data staging on surrogates (FR8) enables data sharing and collaboration (FR9) between mobile devices leveraging the same surrogate and eventual upload of that data to the enterprise cloud (FR7).
- Similar to computation offload systems, offloading data processing operations to surrogates minimizes battery consumption (NFR1) on the mobile device.
- Proximate surrogates also enable faster response times (NFR2) for data processing and queries than sending data/queries to remote clouds.
- Implementing a runtime decision mechanism for offloading data to surrogates optimizes available bandwidth (NFR14) and minimizes data transfers thereby minimizing battery consumption (NFR1).

Constraints

Availability (NFR14) of the surrogate is key to realizing most of the stated benefits of cyber-foraging for sensing applications. In addition to implementing availability tactics on the surrogate, such as fault detection, recovery, and prevention [13], a sensing application needs to detect surrogate unavailability, cache data when the surrogate is unavailable, and make decisions on what to do when operating in disconnected mode and storage capacity limits are reached (e.g., perform local data processing, discard data, or stop operations).

7.5.2 Usage Context 7: Data-Intensive Mobile Applications

Data-intensive mobile applications rely on large sets of data to provide their functionality. Data typically resides in data centers or in the enterprise cloud. Examples of data-intensive applications and domains include:

- Mobile cloud applications: These applications provide a front end to data residing in the cloud, such as social media apps, map and navigation apps, and e-commerce applications.
- Online gaming applications: Online gaming requires continuous streaming of data to and from the cloud in order to synchronize with other players.

- Data-rich domains: Healthcare and other data-rich domains are characterized by large sets of connected data, which means that queries for one type of data typically trigger queries for other sets of related data.

Data-intensive mobile applications require large amounts of data that resides in the cloud and surrogates serve as intermediaries between mobile devices and the cloud to avoid direct communication to the cloud for every data operation. FR7 (data staging) is therefore redefined as *Staging data in transit from the cloud*, as shown in *UC7* in Figure 7.1.

In addition to the redefined FR7, NFR1, NFR2, NFR3 and NFR14, cyber-foraging systems in this usage context need to satisfy additional requirements related to efficient data flows between the surrogate and the mobile device.

(FR10) Display of prioritized/relevant information: Mobile devices have small(er) screen sizes that limit the amount of information that can be displayed at a time. Surrogates pre-process data that is retrieved or pushed from the cloud, such that mobile devices receive data that is ready to be displayed, or filtered such that they only receive data of interest or relevance.

(NFR16) Query Efficiency: Queries should be executed against data located in proximate surrogates instead of data residing in the cloud.

Benefits

For data-intensive mobile applications, surrogates can cache data from the cloud (FR7) to minimize high latency communication between mobile devices and the cloud, which decreases response time (NFR2); provides extended, proximate data storage for applications (NFR3); and reduces battery consumption (NFR1).

Surrogates can perform data filtering and priorization (FR10) so that mobile device users receive only the data that they need (NFR2).

Constraints

Data-intensive mobile applications only benefit from cyber-foraging if the data that they need is already on the surrogate, in order to avoid direct communication to the cloud. This means that there have to be mechanisms on the surrogate to predict what data will be needed next by mobile applications (NFR16). Data may be pre-fetched based on mobile device context (e.g., location), user profile (e.g., preferences), access history (i.e., data that the user

has accessed in the past), or data relations (e.g., querying a purchase order also fetches vendor, product, and other data related to that order).

7.6 Summary and Conclusions

This chapter presented a characterization of usage contexts for cyber-foraging defined in terms of functional and non-functional requirements for cyber-foraging systems. The usage contexts ranged from the typical offload of short computation operations, to mobile applications in low coverage environments, to the much more demanding and visionary use of public surrogates. Each usage context showed that NFRs can encompass both benefits and constraints. What this means is that there are NFRs that enable a system to achieve the benefits of cyber-foraging, and there are other NFRs that, if not met, will compromise the benefits of cyber-foraging for mobile systems.

The goal for characterizing usage contexts is to help developers of computation- and data-intensive mobile systems (1) determine if cyber-foraging is the appropriate paradigm for reaching desired functional and non-functional requirements, (2) better understand the requirements that need to be met to realize the full benefits of cyber-foraging as well as the constraints for realizing those benefits, (3) develop scenarios and test cases that can be used to determine if requirements are being met.

The goal of the model is to provide context for software engineering life cycle activities for computation- and data-intensive mobile systems, with the intent of developing systems that fully realize the benefits of cyber-foraging.

- Requirements engineers can use the model to determine if cyber-foraging is the appropriate paradigm for reaching desired functional and non-functional requirements
- Software architects and designers can use the model to better understand the requirements that need to be met to realize the full benefits of cyber-foraging, as well as the constraints for realizing those benefits
- Quality assurance personnel can develop scenarios and test cases that can be used to determine if system requirements are being met.

As cyber-foraging becomes a standard feature for computation- and data-intensive mobile systems, it will become even more important to have models such as the one presented in this chapter. These usage contexts combined with the architectural tactics for cyber-foraging identified in Chapter 3 provide a standard language and set of reusable design decisions that will help in

developing better and more standard mobile systems that leverage all the potential benefits of cyber-foraging, as well as mobile devices and operating systems that enable and facilitate these benefits.

Table 7.2: Cyber-Foraging Usage Contexts: Functional and Non-Functional Requirements

Usage Context	**Recurring FRs**	**Recurring NFRs**
UC1: Computation-Intensive Mobile Applications (Short Operations)	(FR1) Offload of computation-intensive operations	(NFR1) Energy efficiency (NFR2) Faster response time (NFR3) Increased computing power (NFR4) Maintainability and Evolvability
UC2: Mobile Applications in Low Coverage Environments	(FR1) Offload of computation-intensive operations (FR2) Access to data residing in the cloud (FR3) Disconnected operations between surrogates and the cloud	(NFR1) Energy efficiency (NFR2) Faster response time (NFR3) Increased computing power (NFR5) Fault tolerance (NFR6) Ease of configuration
UC3: Computation-Intensive Mobile Applications (Long Operations)	(FR1) Offload of very computation-intensive operations (FR4) Disconnected operations between mobile devices and surrogates (FR5) Parallel offload	(NFR1) Energy efficiency (NFR2) Faster response time (NFR3) Increased computing power (NFR5) Fault tolerance (NFR7) Code/Data mobility (NFR8) Scalability
UC4: Mobile Applications in Hostile Environments	(FR1) Offload of computation-intensive operations (FR2) Access to data residing in the cloud (FR3) Disconnected operations between surrogates and the cloud (FR4) Disconnected operations between mobile devices and surrogates	(NFR1) Energy efficiency (NFR2) Faster response time (NFR3) Increased computing power (NFR5) Fault tolerance (NFR6) Ease of configuration (NFR7) Code/Data mobility (NFR9) Ease of deployment (NFR10) Survivability

Continued on next page

Table 7.2 – *Continued from previous page*

Usage Context	**Recurring FRs**	**Recurring NFRs**
UC5: Public Surrogates	(FR1) Opportunistic offload of computation-intensive operations (FR6) Discoverable surrogates	(NFR1) Energy efficiency (NFR2) Faster response time (NFR3) Increased computing power (NFR5) Fault tolerance (NFR11) Trust (NFR12) Portability (NFR13) Lossless user experience
Sensing Applications	(FR7) Staging data in transit to the cloud (FR8) Sensor data and/or continuous data stream processing (FR9) Local data sharing and collaboration	(NFR1) Energy efficiency (NFR2) Faster response time (NFR3) Increased computing power (NFR14) Bandwidth efficiency (NFR15) Availability
Data-Intensive Mobile Applications	(FR7) Staging data in transit from the cloud (FR10) Display of prioritized/relevant information	(NFR1) Energy efficiency (NFR2) Faster response time (NFR3) Increased computing power (NFR14) Bandwidth efficiency (NFR16) Query efficiency

8

Decision Model for Cyber-Foraging Systems

This chapter addresses research question RQ4 and presents a decision model based on a mapping of functional and non-functional requirements for cyber-foraging systems to the architectural tactics presented in Chapter 3, as well as relationships between tactics. The goal of the decision model is to guide the architecture and evolution of cyber-foraging systems that meet their intended functional and non-functional requirements, while understanding the effects of architectural decisions.

8.1 Introduction

There is a large amount of research in cyber-foraging as shown in Chapter 2, but the reality is that there are not many deployed, operational cyber-foraging 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 [109], the need for cyber-foraging systems will arise from industry and government, along with a need for guidance for system architects and developers.

We present a decision model for cyber foraging systems that maps functional and non-functional requirements to the architectural tactics presented in Chapter 3. 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.

8.2 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 is represented as a set of design elements [14][58]. 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 8.1. A 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 [13]. The benefits of using a tactic are represented in the decision model 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.

To represent that a tactic could be used in combination with another tactic to address tradeoffs (i.e., applied together), we use a line with a double-headed arrow between tactics, as shown by the *complements* relationship in Figure 8.1. 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 that come from using the initial tactic also apply to the combination of the tactics. For example, in Figure 8.1, the use of Tactic N to complement Tactic 1 means that the resulting system qualities of using the combination of the tactics are *System Quality 1*, *Tradeoff 1*, *System Quality N*, and *Tradeoff N*. 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 and lead to the same result, we use the label *[alternatives]* to qualify the *complements* relationship. For example, in Figure 8.1, Tactics 3 and 4 are

alternatives for complementing Tactic 2. Note that the even though the result of applying either tactic is the same, the effect on system qualities can be different, as shown by *System Quality 3*, *Tradeoff 3*, *System Quality 4*, and *Tradeoff 4*

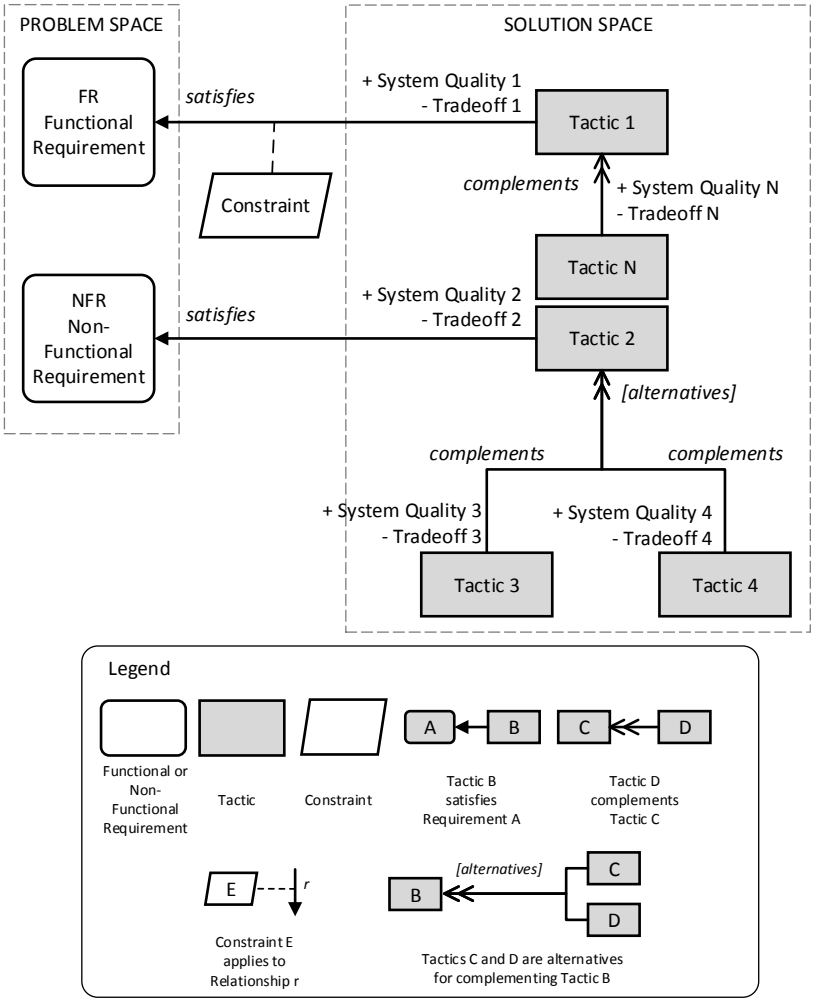

Figure 8.1: Decision Model Notation

Figure 8.1 will be used as the template for the decision models for cyber-foraging systems described in the following sections. The legend in this figure applies to all the decision model figures. The requirements in the solution space are derived from the requirements presented in Chapter 7, complemented with requirements from the case studies presented in Chapters 4, 5, and 6. The tactics in the problem space are the architectural tactics for cyber-foraging presented in Chapter 3.

8.3 How to Use the Decision Models

Cyber-foraging systems have, at a minimum, the following combination of functional requirements (Section 3.2), which map to the first four tactic selection steps in Figure 8.2.

- 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

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 8.2. 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. Architecture evaluation techniques such as ATAM [13] 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.

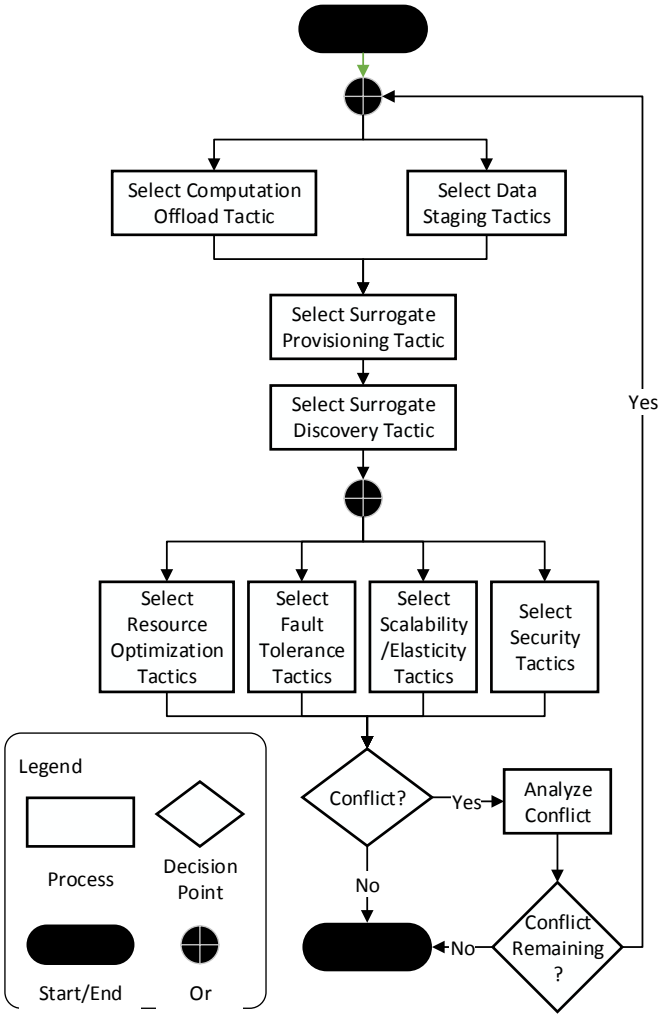

Figure 8.2: How to Use the Decision Models

8.4 Decision Models for Cyber-Foraging Systems

The Computation Offload tactic (Section 3.2.1) enables mobile clients to offload expensive computation to surrogates. The decision model in Figure 8.3 starts from a functional requirement stating that a mobile system has a computing requirement in which the cost to execute the computation locally on the mobile client is greater than the cost to send and execute the computation on a surrogate. The result of using this tactic is *increased computing power* and *increased energy efficiency*. However, this tactic is based on an always-offload strategy (Section 2.5.1.2), which means that computation is always offloaded to a surrogate and never executed locally. There are two assumptions in this case that might not always be true (1) the surrogate is always available, and (2) it is always more beneficial to offload. The result of the first assumption is the potential for *reduced availability* because the computation will only execute if a surrogate is available. The result of the second assumption is the potential for *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 (e.g., battery, memory, bandwidth) due to retransmissions and switching between power modes [29].

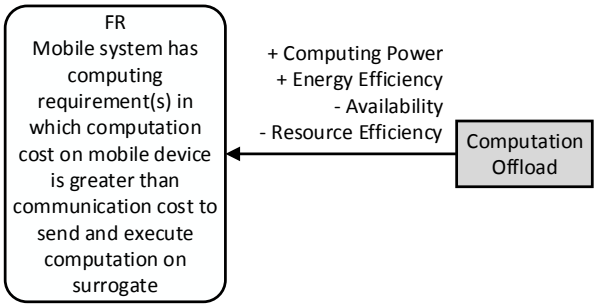

Figure 8.3: Decision Model for Computation Offload

8.4.1 Data Staging

Figure 8.4 presents a decision model for selecting data staging tactics. The Out-Bound Pre-Processing tactic (Section 3.2.2.3) enables mobile devices to collect data in the field, which is then stored on surrogates that can pre-process the data, such that the data that is sent on to the cloud is ready for

consumption and serves an immediate need. Data is uploaded to the cloud when network connectivity is available. This tactic *increases computing power* in the form of greater storage and data processing capabiities on the surrogate. It also provides *increased energy efficiency* because of the energy savings from using WiFi or short-range radio instead of broadband wireless to connect to the cloud [10]. Finally, it provides *increased bandwidth efficiency* between the surrogate and the cloud 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 (Section 3.2.2.2) 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 data of interest or relevance. This tactic provides *increased computing power* in the form of greater storage and data processing power on the surrogate. 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, but in addition by (1) avoiding direct communication to the cloud for every data operation, and (2) processing data for adequate 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 to function properly.

The Pre-Fetching tactic (Section 3.2.2.1) can be used to complement the In-Bound Pre-Processing tactic, but can also be used on it 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* 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. Access to the cloud is therefore only necessary when the data is not already available on the surrogate. However, it can *affect bandwidth efficiency negatively* between surrogates and the cloud if the pre-fetching algorithm retrieves more data than is necessary, or 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,

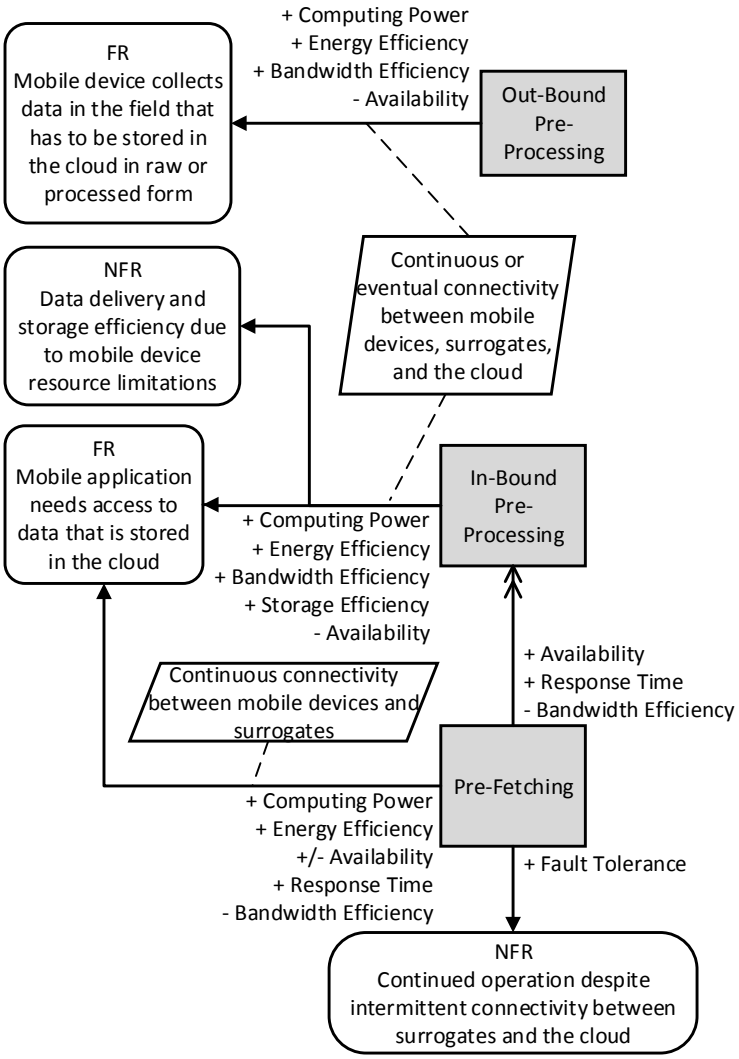

Figure 8.4: Decision Model for Data Staging

it also has a *negative effect on availability.*

8.4.2 Surrogate Provisioning

Figure 8.5 presents a decision model for selecting a surrogate provisioning tactic. In the Pre-Provisioned Surrogate tactic (Section 3.2.3.1), offloaded computation and/or data processing operations are already installed on the surrogate at deployment time. This tactic 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.

In the Surrogate Provisioning from the Cloud tactic (Section 3.2.3.3) 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. This tactic offers *greater flexibility* than the Pre-Provisioned Surrogate tactic because capabilities are not limited by what is already installed on the surrogate, 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.* This tactic also *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 the 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 (Section 3.2.3.2), the mobile device sends the offloaded computation or data processing operations to the surrogate at runtime. The surrogate installs the computation inside an execution container and starts the application on behalf of the mobile device. This tactic offers the *greatest flexibility* because of the potential for executing any offloadable capability that resides on the mobile device, which

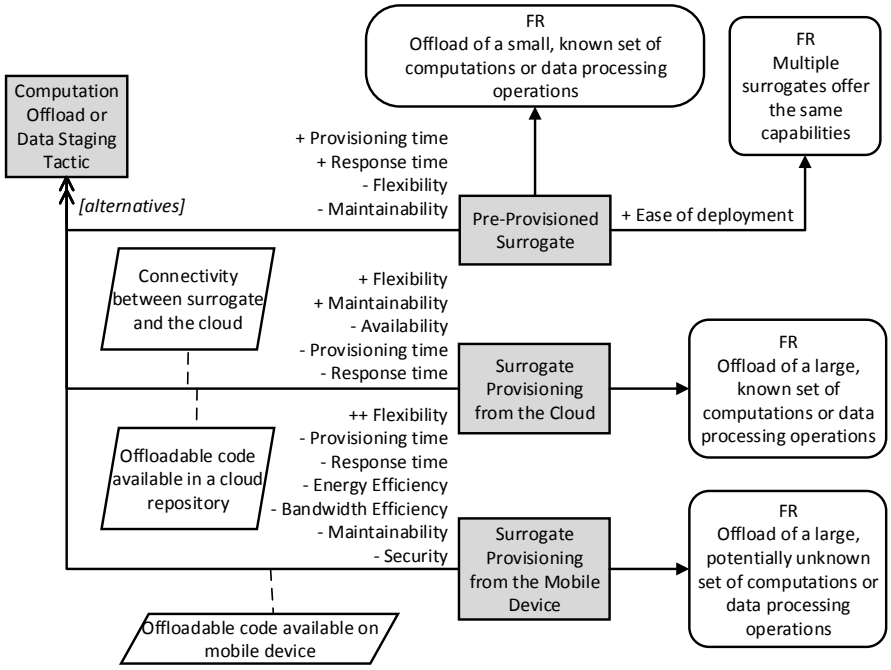

Figure 8.5: Decision Model for Surrogate Provisioning

makes it a good match for public surrogates (Section 7.4.2.4) or other usage contexts 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. This also *increases response time*, at least for the first time the offloaded capability is executed. *Energy efficiency is negatively affected* because of the battery power required on the mobile device for sending the offloaded computation. In addition, depending on the size of the capability that is transferred, *bandwidth efficiency could be negatively affected*, especially in resource-constrained and hostile environments (Sections 7.4.2.1 and 7.4.2.3). *Maintainability is also reduced* 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.

8.4.3 Surrogate Discovery

Figure 8.6 presents a decision model for selecting a surrogate discovery tactic. In the Local Surrogate Directory tactic (Section 3.2.4.1) 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 usage contexts 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 surrogate 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. This tactic 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 (Section 3.2.4.2) the mobile device contacts a cloud server that maintains a list of 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, surrogate load, or any other data that is available in the directory or was provided by the mobile device as query parameters, 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. All surrogate metadata is populated and updated in this central repository, which *increases maintainability*. It also *increases flexibility* because all the mobile device needs to know is the address of the cloud server, which maintains the list of all potential

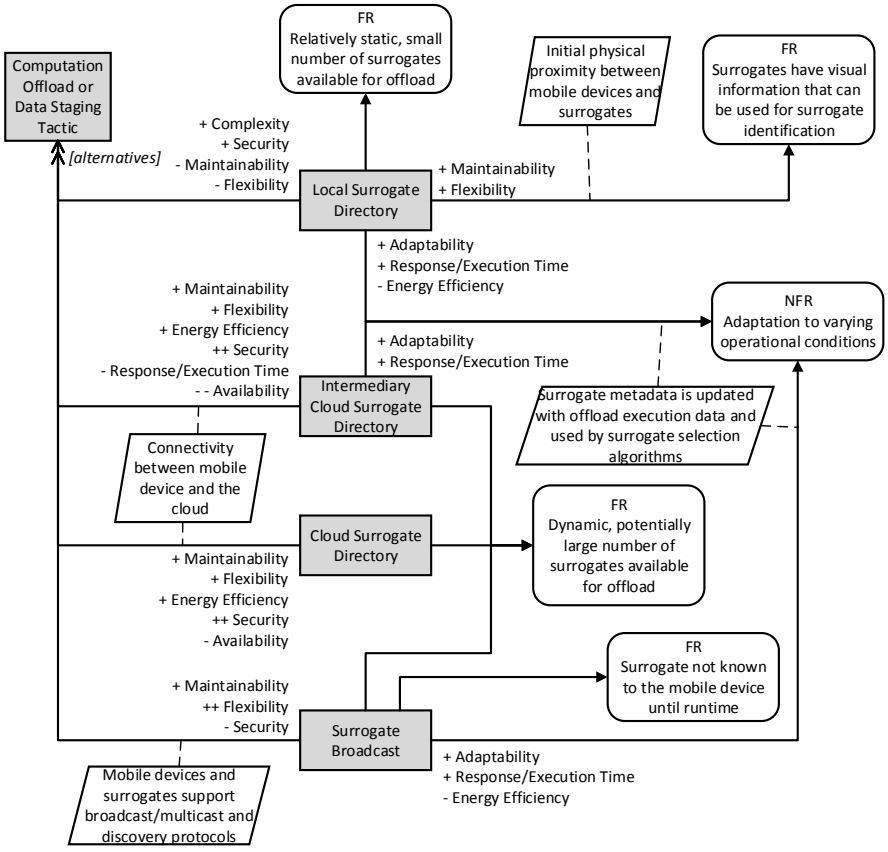

Figure 8.6: Decision Model for Surrogate Discovery

surrogates. The cloud server then selects the surrogate that is the best match for the offloaded task. The centralization aspect enables this tactic to handle a dynamic, potentially large number of surrogates available for offload. *Security is highly increased* by this tactic because the mobile device only needs to trust the cloud surrogate directory server and can pre-exchange credentials for authorization (Section 8.4.7). 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 if it is unavailable to mobile devices for surrogate discovery.

In the Intermediary Cloud Surrogate Directory tactic (Section 3.2.4.2), 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 the selected surrogate, and then returns the results to the mobile device. In essence, the surrogate directory server acts as an intermediary between the mobile device and the surrogate. Similar to the Cloud Surrogate Directory tactic, it *increases maintainability, flexibility, energy efficiency, and security.* An additional advantage of this tactic 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, and has the *potential to improve response/execution time* if the selection algorithm uses the updated metadata. However, *response/execution time can increase* because the mobile devices communicate with surrogates through the surrogate directory server and not directly, potentially offsetting any gains from updating surrogate metadata with offload execution data. Considering that a surrogate can serve multiple mobile devices, it is 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 (Section 3.2.4.3), surrogates broadcast or advertise their presence to mobile devices. This removes the burden of 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 availability and flexibility.* The broadcast aspect enables this tactic to handle a dynamic, potentially large number of surrogates available for offload. *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 8.4.7).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.

8.4.4 Resource Optimization

Figure 8.7 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 (Section 3.3.1.1) 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 runtime environment. However, the execution of a very complex partitioning algorithm per offload operation could *also lead to reduced energy efficiency.* As stated by the constraint in Figure 8.7, the offloaded code has to exist on both the mobile device and the surrogate. This can lead to *decreased maintainability*, especially if the mobile and surrogate platforms are different, and also *reduces legacy leverage* because the code would in many cases 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 [29].

The Runtime Profiling tactic (Section 3.3.1.2) enables mobile devices to gather data about current conditions to update the profiling data and models that are used in the calculation of the optimization function. The use of this tactic to complement the Runtime Partitioning tactic *increases energy efficiency* and *further increases resource efficiency* because current conditions are considered in the offload decision. Because the data used by the optimization offload is updated either periodically or after every offload operation, any errors in the initial models and data are adjusted over time, therefore *increasing system performance.* A constraint for the use of this tactic, as shown in Figure 8.7, 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

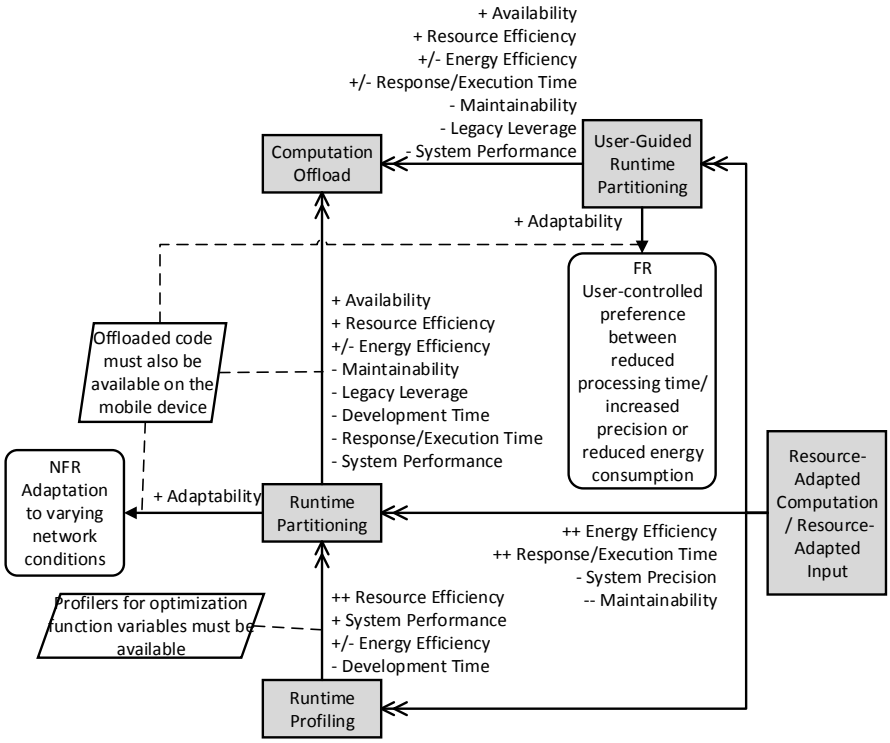

Figure 8.7: Decision Model for Resource Optimization

time or increased precision are preferred over reduced energy consumption. The assumption in this case is that the greater precision or reduced processing time would consume more energy on the mobile device if executed locally. The User-Guided Runtime Partitioning tactic (Section 3.3.1.1), a variation of the Runtime Partitioning tactic, enables users to select the goal of the optimization function therefore *increasing availability.* 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 (Section 3.3.1.3) 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 (Section 3.3.1.3), a variation of the Resource-Adapted Computation 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 the equivalent code have to be maintained, even if platforms are compatible.

8.4.5 Fault Tolerance

Figure 8.8 presents a decision model to select fault tolerance tactics to complement Computation Offload tactics. The Local Fallback tactic (Section 3.3.2.1) 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.* Because this tactic requires offloadable code to exist on the mobile device and the surrogate, *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. This is why this tactic is best fit for stateless, request/response operations (Section 3.3.2.1).

The Alternate Communications tactic (Section 3.3.2.4) 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 (Section 3.3.2.5) enables a surrogate to migrate offloaded computation to a connected surrogate when it detects that it might

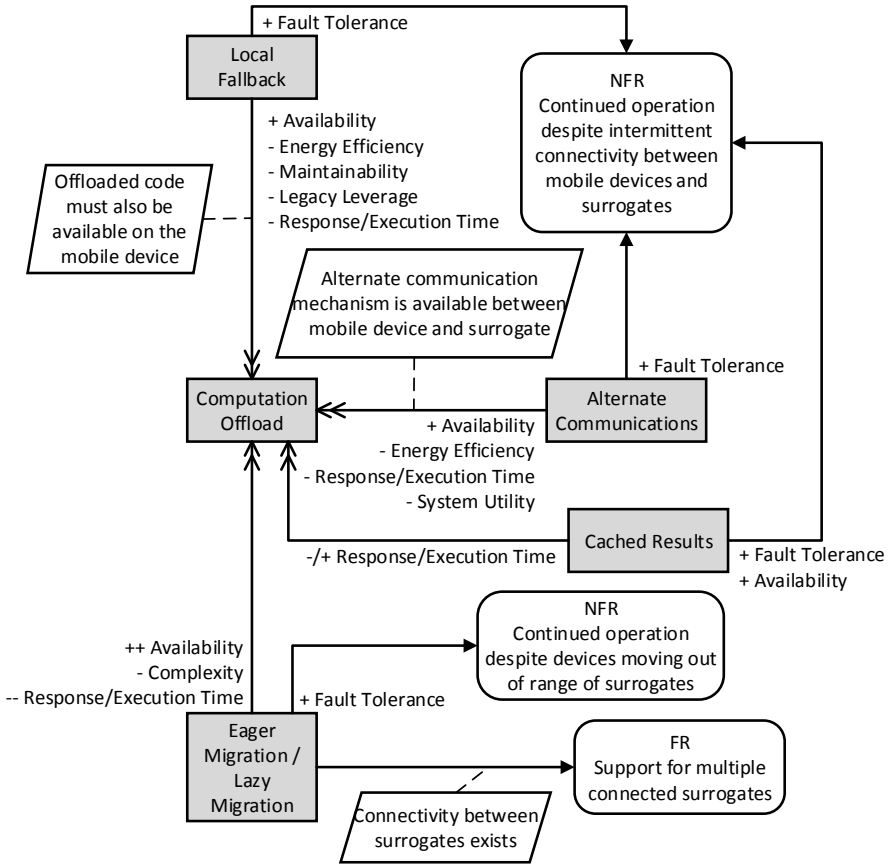

Figure 8.8: Decision Model for Fault Tolerance for Computation Offload

not be able to continue serving the mobile device that generated the offload request. 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 range of the mobile device. *Availability is greatly increased* because these tactics take a more proactive approach to detecting disconnection, as opposed to the other tactics which react after disconnection has been detected. 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/container that has to be migrated between surrogates, although this is a one-time cost upon migration. In Lazy Migration, *response/execution time increases* due to the rerouting that takes place with every offload operation.

The Cached Results tactic (Section 3.3.2.3) 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 8.9 presents a decision model to select fault tolerance tactics to complement Data Staging tactics. The Client-Side Data Caching tactic (Section 3.3.2.3), 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 *provides fault tolerance and increases 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 and storage is limited on the mobile device, 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. In addition, *data integrity is negatively affected* due to the potential for data loss if storage on the mobile devices becomes full.

The Opportunistic Mobile-Surrogate Data Synchronization tactic (Section 3.3.2.2) 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 *provides fault tolerance and increases availability.* Because data is stored on the mobile device, *response time improves for data requests* over having to send data requests to surrogates. 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 depending* if the complexity of the algorithms used to keep data synchronized is high. In addition, if data sets are synchronized that may never be used by the mobile apps running on the device, or data synchronization policies do not match the operational environment, it can lead to *bandwidth inefficiency*, especially in resource-constrained and hostile environ-

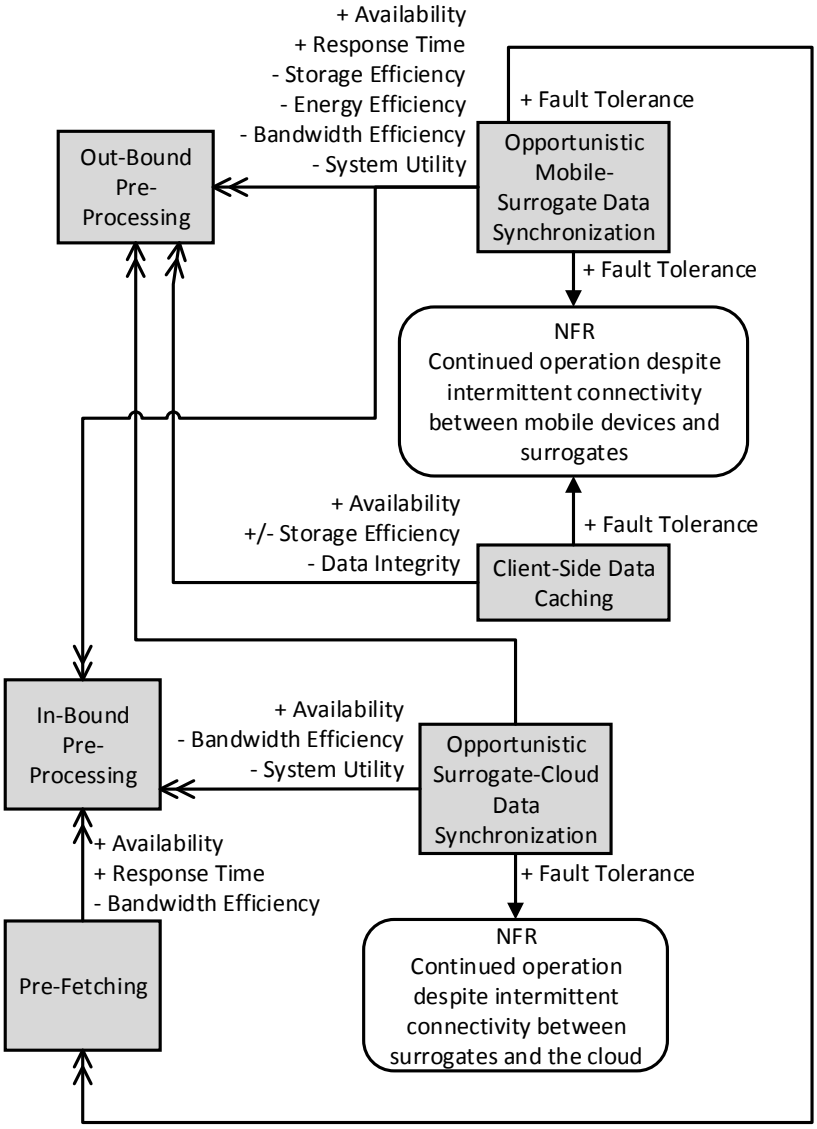

Figure 8.9: Decision Model for Fault Tolerance for Data Staging

ments (Sections 7.4.2.1 and 7.4.2.3). Finally, *system utility may be reduced* if data on the mobile device becomes stale when not synchronized over a long period of time.

The Opportunistic Surrogate-Cloud Data Synchronization tactic (Section 3.3.2.2), 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 *provides fault tolerance and increases 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.

8.4.6 Scalability and Elasticity

Figure 8.10 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 (Section 3.3.3.1) 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 (Section 3.3.3.2) creates execution containers that are of the appropriate size for the offloaded computation 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 (Section 3.3.3.2), starts offloaded computation in a container of a predefined default size, but if an error occurs at runtime that indicates that the container does not have the necessary computing power for the task, 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*

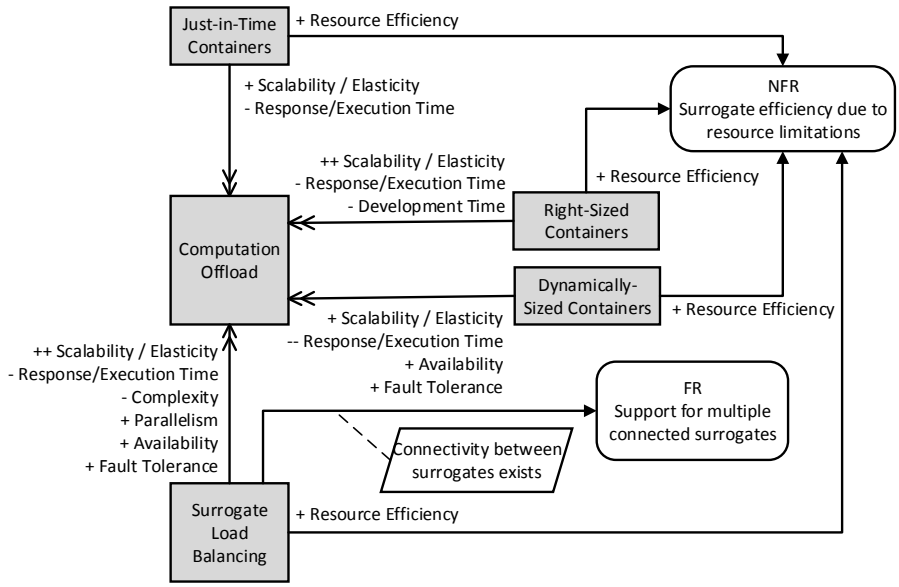

Figure 8.10: Decision Model for Scalability and Elasticity

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, especially if the default container is not sized appropriately (i.e., too small for many offloaded tasks). In exchange, this tactic *provides fault tolerance and increases availability* because of the continued operation of the offloaded task despite initially insufficient resource errors.

The Surrogate Load Balancing tactic (Section 3.3.3.3) enables surrogates to send offloaded computation to other less-loaded, connected surrogates in order to provide a better user experience to all the mobile devices that it serves. *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 of the system* as it requires at least a load balancer and a system monitor to detect when thresholds have been reached and load have to to migrated. In exchange,

the tactic *provides fault tolerance and increases availability* because offload requests are migrated before the system is overloaded and stops responding.

8.4.7 Security

One of the main findings from the primary studies (Section 2.6) is that there is very little discussion of system-level concerns that have to be considered when moving from experimental prototypes to operational systems. One of these system-level concerns is security.

The decision model in Figure 8.11 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 these two steps are typical of any trusted exchange between two system nodes, what is shown in the decision model are the different options for implementing the tactic. The decision model informally presents different options for implementing the tactic instead of complementary tactics. Combinations of these options could become variations of the Trusted Surrogates tactic if implemented and validated in cyber-foraging systems.

8.4.7.1 Credential Exchange

Security credentials have to be exchanged in order to create a trusted relationship between mobile devices and surrogates. Examples of credentials that could be used in a cyber-foraging system include username/password, symmetric and asymmetric keys, certificates, biometrics (e.g., fingerprints, face recognition, voice recognition, retinal scans), and behaviometrics (e.g., keystroke analysis, handwriting, gestures). There are several options that could be used to exchange credentials between a mobile device and a surrogate, including some of the methods outlined in a survey by Alizadeh et al in [4] in the context of mobile cloud computing.

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. How this takes place could be as simple as manually loading the credentials on each node, or could be a more complex registration process. The advantage of pre-usage exchange is *reduced complexity* as this is a well-known mechanism commonly used in client/server systems. However, there is a *negative effect on flexibility* because mobile devices can only interact with surrogates with pre-existing

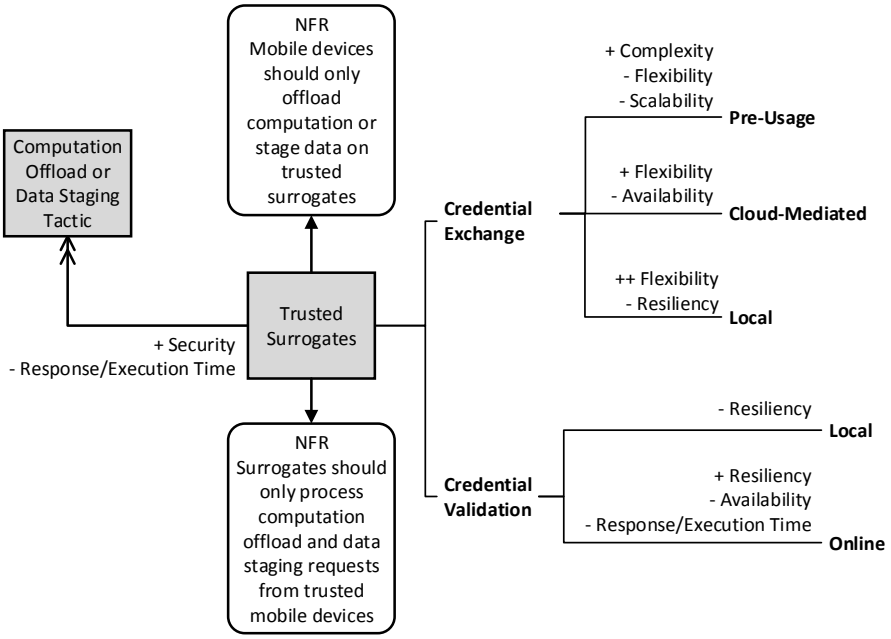

Figure 8.11: Decision Model for Security

security relationships. This method also *may not scale* for systems where there are many-to-many relationships between mobile devices and surrogates.

2. Cloud-Mediated Credential Exchange: In this method a security relationship does not have to pre-exist between mobile devices and surrogates. Credential exchange takes place as part of the offload process the first time that a mobile device uses a surrogate. During the offload process, the mobile device contacts a cloud-based system that is trusted by the surrogate to register its credentials. Context and mobile device sensors could be leveraged as part of multi-factor authentication for the mobile device [4]. The cloud-based system validates the mobile device credentials and, if valid, sends the mobile device the surrogate credentials; it also sends the mobile device credentials to the surrogate. The advantage of this method 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: In this method a security relationship does not have to pre-exist between mobile devices and surrogates, and there is no need for a cloud-based intermediary. 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. This method 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 [37][53][79].

8.4.7.2 Credential Validation

Once credentials have been exchanged, either in advance or during the offload process, these credentials have to be validated at runtime to make sure that surrogates and mobile devices are legitimate.

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. Once credentials have been exchanged there is no way to remove them unless they have an expiration time after which they are no longer valid.

2. Online Validation: Credentials are sent to an online trusted authority for validation. The advantage is that it protects against revoked credentials because these are centrally validated with every interaction between a mobile device and a surrogate. However, there is a *negative effect on availability* because the offload cannot happen unless there is connectivity to the online trusted third party. In addition, *response/execution time increases* because of the additional time needed for online validation.

8.5 Validation

After the execution of the case studies, the developers of the Tactical Cloudlets (Chapter 4), GigaSight (Chapter 5), and AgroTempus (Chapter 6) systems

were presented with the decision models 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 development experience) while the developer of the AgroTempus system is junior (<1 year of development experience).

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 that came from the developer of the Tactical Cloudlets system was regarding the effect on response/execution time of the Cached Results tactics (Section 8.4.5). After some discussion we agreed to separate the effect as negative for the initial offload operation and positive when the mobile device reconnects, as is now shown in Figure 8.8.

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 more junior developer of the AgroTempus system 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 the 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 of the three systems also agreed that the decision models would help with architecture evaluation. 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 8.3. The decision models help to identify the tradeoffs, but not to quantify

the concrete effect because it varies depending on requirements and operational conditions. The specific example provided by one developer is related to energy consumption. Sending one byte of information is typically more power-hungry for a wireless network interface card (NIC) than receiving information, but this also depends on the interference on the link and other factors [3]. 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. This addition would also favor an agile/iterative approach such that you start from tactics, you build a prototype, and then you evaluate it using the recommended profiling tool or method. This may lead to a refinement of the architecture based on the results, possibly using a different 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 educating and communicating the different options and complexities of developing cyber-foraging systems
   - Input for building developer tools that (1) automatically generate components based on the tactics. (2) serve as input for estimating cost and effort for a system, (3) assign weights to the functional and non-functional requirements such that it can guide the selection of tactics
   - Aid for understanding system qualities that were likely influential when reverse engineering a system

8.6 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) [81], 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. Gu et al [50] propose a template for decision making in service-oriented systems based on QOC. Similarly, Zimmermann et al [130] present a tool for architecture decision guidance across projects with QOC diagram suport. Another popular

approach from the field of Software Measurement is Goal Questions Metric (GQM) [12]. 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.

We present a decision model for the domain of cyber-foraging systems that links tactics to functional and non-functional requirements. Similarly, Gross and Yu [49] 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). Zdun [124] 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 [54] 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.

8.7 Conclusions

This chapter 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.

Developing the decision model highlighted the many tradeoffs that architects must make when designing a system. It also highlighted the fact that even though there is great value in the qualitative analysis supported by the decision model, quantitative analysis is required in order to understand the

concrete impact of architecture decisions. We expect this thesis and the resulting decision model to motivate future quantitative analyses to measure the impact associated to the usage of the architectural tactics for cyber-foraging systems, as will be further described in Section 9.2.2.

9 Conclusions

Cyber-foraging has tremendous potential for supporting mobile computing at the edge. With increasing number of mobile devices and users [46][66], increased network traffic cause by trends in the Internet of Things (IoT) [44][85], and increasing complexity of an always-connected-user experience [45][87], there is reason to believe that cyber-foraging will become a standard feature of mobile applications. However, while there is a large amount of research in cyber-foraging, the reality is that there are not many deployed, operational cyber-foraging systems. As these systems become more prevalent due to their proven benefits, in terms of energy efficiency, reduced latency, and increased availability, combined with the emergence of micro data centers and edge clouds, a need will arise for guidance on their architecture and development. This thesis starts to provide that guidance by presenting software architecture strategies for cyber-foraging systems, in the form of architectural tactics and a decision model. In this chapter we revisit the research questions presented in Chapter 1, summarize our contributions with respect to these questions, and discuss future research.

9.1 Contributions

The goal of this thesis is to develop concrete software architecture guidance for the development of cyber-foraging systems that meet critical system qualities such as resource optimization, fault tolerance, scalability, and security, while conserving resources on the mobile device. Therefore, the main research question for this thesis is *"What software architecture strategies can be used to build cyber-foraging systems?"* In Chapter 1 we identified four research sub-questions that further characterize the research problem. This section

summarizes the answers to these questions according to our findings.

9.1.1 RQ1: What Software Architecture Design Decisions for Cyber-Foraging Systems can be Identified in the Literature?

Chapter 2 presented the results of a Systematic Literature Review (SLR) to discover architectural design decisions in cyber-foraging systems. As an answer to RQ1, a total of 58 primary studies were identified that contained 53 computation offload systems and 8 data staging systems. The identified 61 cyber-foraging systems were analyzed using a categorization of architecture decisions related to what, when and where to offload computation and data from mobile devices.

What we found from the analysis of the systems is that the main focus of the studies is on the development of different and novel computation offload and data staging systems targeted at guaranteeing fidelity of results, and optimizing attributes such as energy consumption, network bandwidth usage, and performance/response time. For computation offload systems, the offload mechanisms range from dynamic approaches in which the computation is provisioned from the mobile device to more static approaches in which the computation already exists on the offload target. For data staging systems, the capabilities of the offload target range from an extension of the mobile device's storage to sophisticated algorithms that predict and stage the data that will likely be needed by the mobile device. As stated earlier, the number of computation offload systems (53) is much larger than the number of data staging systems (8). The architecture design decisions that we identified in this systems served as the basis for the definition of the architectural tactics presented in Chapter 3.

What we also learned from these studies is that although there is a large amount of research in the area of cyber-foraging systems, there is very little discussion of system-level concerns that have to be addressed when moving from experimental prototypes to operational systems. In particular, the analysis allowed us to identify gaps and opportunities for research in (1) non-functional requirements that are not widely addressed but are relevant to cyber-foraging systems, such as ease of deployment, resiliency, and security, (2) system-level architecture analysis, (3) large-scale evaluations, and (4) architectures for data staging systems.

In summary, the results show that this is an area with many opportunities for research that will enable cyber-foraging systems to become widely adopted

and move out of the labs and into real operational scenarios.

9.1.2 RQ2: What Architectural Tactics can be Derived from the Identified Architectural Design Decisions?

Chapter 3 presented a catalog of architectural tactics for cyber-foraging that was derived from the results of the systematic literature review on architectures for cyber-foraging systems presented in Chapter 2. As an answer to RQ2, a total of 30 tactics were identified and divided into functional and non-functional tactics. Functional tactics are broad and basic in nature and correspond to the architectural elements that are necessary to meet cyber-foraging functional requirements, such as computation offload, data staging, surrogate discovery, and surrogate provisioning. Non-functional tactics are more specific and correspond to architecture decisions made to promote certain quality attributes such as resource optimization, fault tolerance, scalability and elasticity, and security. Non-functional tactics have to be used in conjunction with functional tactics.

To validate the architectural tactics we conducted three case studies to investigate the use of the tactics in real cyber-foraging systems.

- Chapter 4 presents a case study based on *Tactical Cloudlets*, a computation offload system for use in tactical environments. 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.

- Chapter 5 presents a case study based on *GigaSight*, a data staging system for scalable crowd-sourcing of video from mobile devices. Case study protocol and developer involvement was the same as in the Tactical Cloudlets system.

- Chapter 6 presents a case study based on *AgroTempus*, a computation offload and data staging system targeted at agricultural knowledge exchange in resource-challenged regions. 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.

To further answer RQ2, the results of the case studies show not only the validity of the tactics, but the potential for taking a tactics-driven approach to fulfill functional and non-functional requirements for cyber-foraging systems.

9.1.3 RQ3: What are the Usage Domains and Contexts (Defined in Terms of Functional and Non-Functional Requirements) that Benefit from Cyber-Foraging?

Chapter 7 presented a characterization of the usage domains and contexts that benefit from surrogate-based cyber-foraging, defined in terms of functional and non-functional requirements. We started from the set of primary studies identified in Chapter 2 and conducted a literature study to identify usage contexts and domains for cyber-foraging systems, and then mapped them to relevant functional and non-functional requirements in each context. The goal of this characterization is to provide context for software engineering life cycle activities for cyber-foraging systems, such as requirements engineering, software architecture, and quality assurance, with the intent of developing systems that fully realize the benefits of cyber-foraging.

As an answer to RQ3, we identified seven usage contexts that ranged from the typical offload of short computation operations, to mobile applications in low coverage environments, to the much more demanding and visionary use of public surrogates. A finding from the study is that each usage context has non-functional requirements that can be both benefits and constraints. What this means is that there are non-functional requirements that enable a system to achieve the benefits of cyber-foraging, and there are other non-functional requirements that, if not met, will compromise the benefits of cyber-foraging for mobile systems. Understanding these tradeoffs is a motivator for the definition of decision models such as the one presented in Chapter 8.

In addition, these usage contexts combined with the architectural tactics for cyber-foraging identified in Chapter 3 provide a standard language and set of reusable design decisions that will help in developing better and more standard mobile systems that leverage all the potential benefits of cyber-foraging, as well as mobile devices and operating systems that enable and facilitate these benefits.

9.1.4 RQ4: How to Support Architectural Design Decision Making in Cyber-Foraging Systems?

Chapter 8 presented a decision model for cyber foraging systems that maps functional and non-functional requirements to the architectural tactics presented in Chapter 3, and shows the relationships between tactics. The functional and non-functional requirements are largely derived from the usage contexts in Chapter 7 and complemented with the requirements from the case studies in Chapters 4, 5, and 6. The goal of the decision model is to guide the

architecture and evolution of cyber-foraging systems that meet their intended functional and non-functional requirements, while understanding the effects of architectural decisions.

To answer RQ4, we defined a process that starts with the selection of basic tactics for cyber foraging: computation offload, data staging, surrogate provisioning, and surrogate discovery. Then, based on additional functional and non-functional requirements, such as fault tolerance, resource optimization, scalability/elasticity, and security, complementary tactics are selected. A decision model was created for each set of tactics 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 the cyber-foraging systems in the case studies who agreed on the correctness and usefulness of the model for architecture and design of these types of systems.

While we believe that the decision model is a valuable instrument for moving cyber-foraging systems out of the labs and into real operational settings, there is the need for more research in this area, which leads to the next section on future research.

9.2 Future Research

This section concludes the thesis, but the work in software architecture strategies for cyber-foraging systems has just started. Some ideas for future research have already been discussed in previous chapters but we summarize them in this section.

9.2.1 Extension of the Tactics Catalog

The case studies in Chapters 4, 5, and 6 presented a number of opportunities for extension of the tactics catalog in several ways:

- The case studies showed that the existing cyber-foraging systems that were evaluated, as well as the newly developed system, contained a general implementation of the core essence of each tactic. However, small variations were identified, or made to the implementation of the tactic, to satisfy specific requirements or elements of the usage context, or due to technology selection. As the tactics are used in the development and evaluation of cyber-foraging systems, a valuable extension to the tactics

catalog would be to add variations to each tactic to address specific requirements or constraints. In adding these variations, the tactics could be annotated with the core elements (common across all variations) and optional elements (specific to one or more variations).

- Consistent with the findings in Chapter 2 and in the case studies, there are multiple system qualities that are not present in the tactics catalog such as ease of deployment, manageability, recovery, privacy, and surrogate energy efficiency, or minimally covered qualities, such as security. These are system qualities that are necessary for moving from experimental prototypes to operational systems. A valuable extension to the tactics catalog would be to add tactics for these types of requirements, in order to provide full coverage for the requirements for the usage contexts presented in Chapter 7, and start addressing some of these critical operational requirements.

- As more cyber-foraging systems are deployed, architects will discover that technology selection decisions may enable or constrain the use of certain tactics, such as the case with the Surrogate Broadcast tactic in the AgroTempus system (Section 6.3.8.1). Annotating each tactic with technology options and constraints would further increase the utility of the catalog for decision making.

9.2.2 Quantitative Analysis of the Impact of Tactics Selection

An observation resulting from the development of the decision model in Chapter 8 is that the model helps to identify the tradeoffs related to tactics selection, but not to quantify the impact of the selection because it varies depending on requirements and operational conditions. For example, if the selection of a tactic favors energy efficiency over availability, how much is the energy efficiency gain compared to the availability loss?

Architecture guidance has a qualitative component and a quantitative component, and they both provide value to software architects. The decision model presented in Chapter 8 provides support for qualitative analysis of the tradeoffs associated with tactics selection in the development of cyber-foraging systems. The value of a tactics-based approach such as the one presented in this thesis is that it helps a software architect design for system qualities, i.e., how do you architect a system for energy efficiency? A natural component and quantitative component would be support for quantifying the energy efficiency

of the resulting architecture, in itself, but also relative for other architecture options.

The advantage is that we know how to measure qualities in implemented systems; for example:

- The development of the Tactical Cloudlets system presented in Chapter 4 required measuring cloudlet provisioning time, energy consumption on the mobile device, payload size and response time, in order to decide on the best method for provisioning cloudlets in the field.
- The development of the GigaSight system presented in Chapter 5 required measuring throughput, cloudlet performance, algorithm accuracy, and energy consumption on the mobile device, in order to mainly understand the tradeoffs between performance and precision of different deployment and replication options.
- Procaccianti et al in [99] empirically studied the energy impact of two best practices for energy-efficient software. Although the system measured was not a cyber-foraging system, it represents the type of measurements that would be necessary in order to quantify the tradeoffs of architecture decisions.

A complementary quantitative component of the work presented in this thesis is support for quantitative analysis of the impact of tactics selection, to more clearly understand the tradeoffs. As an example, we have started work to quantify the energy efficiency, bandwidth efficiency, and maintainability associated to the different tactics for surrogate provisioning (pre-provisioning, provisioning from the mobile device, and provisioning from the cloud). The results of this work can then be added to the tactics catalog as concrete metrics and measures, but also as benchmarks for architects to conduct their own measurements.

9.2.3 Tools for the Development and Analysis of Cyber-Foraging Systems

When the developers that validated the decision model were asked about other uses for the model beyond decision-making, a common answer was that it could be used as a basis for architecture analysis and code generation tools. Tactics and tactic components could become the building blocks for modeling cyber-foraging systems, and generating code from the models.

Related to the need or quantitative analyses, tools and methods to profile the parameters that influence the positive or negative effect on a system quality

when using a particular tactic is a research area that would be very beneficial for cyber-foraging, and architecture decision-making in general.

9.2.4 Architecture Patterns for Cyber-Foraging Systems

While the difference between patterns and tactics is a common topic of discussion in the software architecture community, Harrison and Avgeriou [54] provide a distinction that is consistent with our findings: "Architecture patterns describe the high-level structure and behavior of software systems as the solution to multiple system requirements, whereas tactics are design decisions that improve individual quality attribute concerns." In essence, architecture patterns are composed of multiple tactics in order to address multiple system requirements. The development of architecture patterns for cyber-foraging systems based on the tactics in the catalog, that also maps to the usage contexts in Chapter 7 is a natural continuation of this research.

Summary

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 to proximate servers called surrogates. 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 capability. The second is data staging to improve data transfers between mobile devices and the cloud by temporarily staging data in transit on intermediate surrogates.

One of the main challenges of building cyber-foraging systems is the dynamic nature of the environments that they operate in. For example, the connection to a surrogate may not be available when needed, or may become unavailable during a computation offload or data staging operation. As another example, multiple surrogates may be available but not all have the required capabilities. Adding capabilities to deal with the dynamicity of the environment has to be balanced against resource consumption on the mobile device so as to not defeat the benefits of cyber-foraging. Being able to reason about the behavior of a cyber-foraging system in light of this uncertainty is key to meeting all its desired qualities, which is why software architectures are especially important for cyber-foraging systems.

While there is a large amount of research in cyber-foraging, 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.

This dissertation starts providing this guidance in the form of software architecture strategies for cyber-foraging systems. First, a catalog of architectural tactics for cyber-foraging systems is presented. These tactics were validated through three case studies and can be used by software architects to achieve system qualities such as resource optimization, fault tolerance, scalability, and security, while conserving resources on the mobile device. Secondly, a characterization of usage contexts for cyber-foraging, defined in terms of functional and non-functional requirements is presented in order to understand the usage contexts that benefit the most from cyber-foraging. Finally, a decision model for cyber-foraging systems is presented that maps functional and non-functional requirements for cyber-foraging systems to the set of architectural tactics. The end goal is to help software architects extend their design reasoning towards cyber-foraging as a way to support the mobile applications of the present and the future, while understanding the effects of their decisions.

Samenvatting

Cyber-foraging is een techniek die mobiele apparaten toestaat om hun rekenkracht en geheugen uit te breiden door het uitbesteden van berekeningen en data naar krachtigere server in de cloud of naar naburige servers, zogenoemde surrogaten. Er zijn twee hoofdvormen van cyber-foraging. Eén vorm is computation offload, waarbij zware berekeningen worden verplaatst naar een server om de batterijduur en rekenkracht te verbeteren. De tweede vorm is data staging, hiermee worden gegevensoverdrachten tussen mobiele apparaten en de cloud verbeterd door het tijdelijk laden van gegevens in overdracht op tussenliggende surrogaten.

Een van de belangrijkste uitdagingen bij het opzetten van cyber-foraging systemen is het dynamische karakter van de omgevingen waarin deze werken. De verbinding met een surrogaat kan bijvoorbeeld niet beschikbaar zijn wanneer deze nodig is, of kan onbeschikbaar worden gedurende computation offload of data staging operaties. Daarnaast kan het bijvoorbeeld ook zo zijn dat er meerdere surrogaten beschikbaar zijn, maar deze niet allemaal aan de vereisten voldoen. Het toevoegen van functionaliteit voor het omgaan met omgevingsdynamiek moet afgewogen worden tegen het extra verbruik op mobiele apparaten, om te voorkomen dat de voordelen van cyber-foraging teniet worden gedaan. Het kunnen redeneren over het gedrag van een cyber-foraging systeem in de context van deze onzekerheid is cruciaal om te voldoen aan alle gewenste eigenschappen. Dit is waarom vooral software-architectuur belangrijk is voor cyber-foraging systemen.

Hoewel er veel onderzoek gedaan is naar cyber-foraging, zijn er in de praktijk weinig operationele cyber-foraging systemen. Aangezien deze systemen steeds belangrijker worden, vanwege hun aangetoonde voordelen in combinatie met de opkomst van micro data centra en edge clouds, zal er behoefte ontstaan naar richtlijnen voor hun architectuur en ontwikkeling.

Dit proefschrift geeft een aanzet voor deze richtlijnen in de vorm van software-architectuurstrategien voor cyber-foraging systemen. Ten eerste wordt een catalogus van architectuurstrategien voor cyber-foraging systemen gepresenteerd. Deze strategien zijn gevalideerd door middel van drie case studies, en kunnen gebruikt worden door software-architecten om eigenschappen zoals resource-
optimalisatie, fouttolerantie en veiligheid te realiseren, zonder de mobiele apparaten teveel te belasten. Ten tweede wordt een karakterisering van toepassingscontexten voor cyber-foraging gepresenteerd in de vorm van functionele en niet-functionele eisen, om inzicht te bieden in welke toepassingscontexten het meest geprofiteerd kan worden van cyber-foraging. Ten slotte wordt een beslis-

singsmodel voor cyber-foraging systemen gepresenteerd, waarin functionele en niet-functionele eisen van cyber-foraging systemen worden gekoppeld aan de gepresenteerde architectuurstrategien. Het uiteindelijke doel is om software-architecten te helpen hun ontwerpen richting cyber-foraging systemen uit te breiden, om zo goed genformeerd de mobiele applicaties van nu en de toekomst te ondersteunen.

Bibliography

[1] Saeid Abolfazli, Zohreh Sanaei, Ejaz Ahmed, Abdullah Gani, and Rajkumar Buyya. Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges. *IEEE Communications Surveys Tutorials*, 16(1):337–368, 2014.

[2] JongHoon Ahnn and Miodrag Potkonjak. mHealthMon: Toward energy-efficient and distributed mobile health monitoring using parallel offloading. *Journal of Medical Systems*, 37(5):1–11, 2013.

[3] Farhan Azmat Ali, Pieter Simoens, Tim Verbelen, Piet Demeester, and Bart Dhoedt. Mobile device power models for energy efficient dynamic offloading at runtime. *Journal of Systems and Software*, 113:173 – 187, 2016.

[4] Mojtaba Alizadeh, Saeid Abolfazli, Mazdak Zamani, Sabariah Baharun, and Kouichi Sakurai. Authentication in mobile cloud computing: A survey. *Journal of Network and Computer Applications*, 2015.

[5] Pelin Angin and Bharat Bhargava. An agent-based optimization framework for mobile-cloud computing. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 4:1–17, 2013.

[6] Trevor Armstrong, Olivier Trescases, Cristiana Amza, and Eyal de Lara. Efficient and transparent dynamic content updates for mobile clients. In *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services*, pages 56–68. ACM, 2006.

[7] Andrius Aucinas, Jon Crowcroft, and Pan Hui. Energy efficient mobile M2M communications. In *Proceedings of ExtremeCom '12*, 2012.

[8] Ali Bahrami, Changzhou Wang, Jun Yuan, and Anne Hunt. The workflow based architecture for mobile information access in occasionally connected computing. In *Proceedings of the IEEE International Conference on Services Computing (SCC'06)*, pages 406–413. IEEE, 2006.

[9] Rajesh Krishna Balan, Darren Gergle, Mahadev Satyanarayanan, and James Herbsleb. Simplifying cyber foraging for mobile devices. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services*, MobiSys '07, pages 272–285, New York, NY, USA, 2007. ACM.

[10] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun 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*, IMC '09, pages 280–293, New York, NY, USA, 2009. ACM.

[11] Baseline. Nine Mobility Trends You Must Watch in 2015, 2015. URL: `http://www.baselinemag.com/mobility/slideshows/nine-mobility-trends-you-must-watch-in-2015.html`.

[12] Victor Basili, Gianluigi Caldiera, and H Dieter Rombach. The goal question metric approach. *Encyclopedia of Software Engineering*, pages 528–532, 1994.

[13] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 3rd edition, 2012.

[14] Kathrin Berg, Judith Bishop, and Dirk 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*, SAICSIT '05, pages 182–191, 2005.

[15] Pearl Brereton, Barbara Kitchenham, David Budgen, and Zhi Li. Using a protocol template for case study planning. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, 2008.

[16] Yu-Shuo Chang and Shih-Hao Hung. Developing collaborative applications with mobile cloud: A case study of speech recognition. *Journal of Internet Services and Information Security (JISIS)*, 1(1):18–36, 2011.

[17] Marat Charlaganov, Philippe Cudré-Mauroux, Cristian Dinu, Christophe Guéret, Martin Grund, and Teodor Macicas. The entity registry system: Implementing 5-star linked data without the web. *arXiv preprint arXiv:1308.3357*, 2013.

[18] Guangyu Chen, B-T Kang, Mahmut Kandemir, Narayanan Vijaykrishnan, Mary Jane Irwin, and Rajarathnam Chandramouli. Studying energy trade offs in offloading computation/compilation in Java-enabled mobile devices. *IEEE Transactions on Parallel and Distributed Systems*, 15(9):795–809, 2004.

[19] Bin Cheng and Michael Probst. HBB-NEXT I D4.4.1: Intermediate middleware software components for cloud service offloading. Technical report, HBB-NEXT Consortium 2013, 2013.

[20] Hao-hua Chu, Henry Song, Candy Wong, Shoji Kurakake, and Masaji Katagiri. Roam, a seamless application framework. *Journal of Systems and Software*, 69(3):209–226, 2004.

[21] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. CloneCloud: Elastic execution between mobile device and cloud. In *Proceedings of the 6th Conference on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[22] Byung-Gon Chun and Petros Maniatis. Augmented smartphone applications through clone cloud execution. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems*. USENIX Association, 2009.

[23] Cisco. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 20152020 White Paper, 2015. URL: `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html`.

[24] comScore, Inc. Number of Mobile-Only Internet Users Now Exceeds Desktop-Only in the U.S., 2015. URL: `http://www.comscore.com/Insights/Blog/Number-of-Mobile-Only-Internet-Users-Now-Exceeds-Desktop-Only-in-the-U.S`.

[25] CTIA. Wireless Quick Facts, 2015. URL: `http://www.ctia.org/your-wireless-life/how-wireless-works/wireless-quick-facts`.

[26] Eduardo Cuervo. *Enhancing Mobile Devices through Code Offload*. PhD thesis, Duke University, 2012.

[27] Glenn Daneels, Jeroen Famaey, Steven Bohez, Pieter Simoens, and Steven Latré. Upstream content scheduling in Wi-Fi DenseNets during large-scale events. In *2015 IEEE GLOBECOM Workshops (GC Wkshps)*. IEEE, 2015.

[28] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13:1587–1611, 2011.

[29] Mian Dong and Lin 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*, MobiSys '11, pages 335–348, New York, NY, USA, 2011. ACM.

[30] Naod Duga. Optimality analysis and middleware design for heterogeneous cloud HPC in mobile devices. Master's thesis, Addis Ababa University, 2011.

[31] Tore Dyba, T. Dingsoyr, and G.K. Hanssen. Applying systematic reviews to diverse study types: An experience report. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 225–234, September 2007.

[32] Sebastian Echeverria, Grace A. Lewis, James Root, and Ben Bradshaw. Cyber-foraging for improving survivability of mobile systems. In *2015 IEEE Military Communications Conference (MILCOM 2015)*, pages 1421–1426, Oct 2015.

[33] Holger Endt and Kay Weckemann. Remote utilization of OpenCL for flexible computation offloading using embedded ECUs, CE devices and cloud servers. In *Applications, Tools and Techniques on the Road to Exascale Computing*, volume 22 of *Advances in Parallel Computing*, pages 133–140. IOS Press EBooks, 2011.

[34] Ericsson AB. Ericsson Mobility Report Appendix, Sub-Saharan Africa, 2013. URL: `http://www.ericsson.com/res/docs/2013/emr-nov13-rssa.pdf`.

[35] Rodolfo G Esteves, Michael D McCool, and Christiane Lemieux. Real options for mobile communication management. In *2011 IEEE GLOBECOM Workshops (GC Wkshps)*, pages 1241–1246. IEEE, 2011.

[36] Yidong Fang, Chris Nokleberg, and Dave Hughes. JSON.simple - A simple Java toolkit for JSON - Google Project Hosting, 2012. URL: `https://code.google.com/p/json-simple/`.

[37] Michael Farb, Yue-Hsun Lin, Tiffany Hyun-Jin Kim, Jonathan M. McCune, and Adrian Perrig. SafeSlinger: Easy-to-use and secure public-key exchange. In *Proceedings of ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2013.

[38] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29:84–106, 2012.

[39] Tore Fjellheim, Stephen Milliner, and Marlon Dumas. Middleware support for mobile applications. *International Journal of Pervasive Computing and Communications*, 1(2):75–88, 2005.

[40] Jason Flinn. Cyber foraging: Bridging mobile and cloud computing. In Mahadev Satyanarayanan, editor, *Synthesis Lectures on Mobile and Pervasive Computing*. Morgan & Claypool Publishers, 2012.

[41] Jason Flinn, Soyoung Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 217–226, 2002.

[42] Jason Flinn, Shafeeq Sinnamohideen, Niraj Tolia, and Mahadev Satyanarayanan. Data staging on untrusted surrogates. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST03)*, 2003.

[43] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[44] Gartner. Gartner Says 4.9 Billion Connected "Things" Will Be in Use in 2015, 2015. URL: `http://www.gartner.com/newsroom/id/2905717`.

[45] Gartner. Gartner Says By 2018, More Than 50 Percent of Users Will Use a Tablet or Smartphone First for All Online Activities, 2015. URL: `http://www.gartner.com/newsroom/id/2939217`.

[46] GfK. Tech Trends 2015, 2015. URL: `http://www.gfk.com/Documents/GfK-TechTrends-2015.pdf`.

[47] Ioana Giurgiu, Oriana Riva, Dejan Juric, Ivan Krivulev, and Gustavo Alonso. Calling the cloud: Enabling mobile phones as interfaces to cloud applications. In Jean M. Bacon and Brian F. Cooper, editors, *Middleware 2009*, volume 5896 of *Lecture Notes in Computer Science*, pages 83–102. Springer Berlin Heidelberg, 2009.

[48] Sachin Goyal. *A Collective Approach to Harness Idle Resources of End Nodes*. PhD thesis, School of Computing, University of Utah, 2011.

[49] Daniel Gross and Eric Yu. From non-functional requirements to design through patterns. *Requirements Engineering*, 6(1):18–36, 2001.

[50] Qing Gu, P. Lago, and H. van Vliet. A template for SOA design decision making in an educational setting. In *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pages 175–182, Sept 2010.

[51] Tao Guan. *A System Architecture to Provide Enhanced Grid Access for Mobile Devices*. PhD thesis, University of Southampton, 2008.

[52] Kiryong Ha, Grace Lewis, Soumya Simanta, and Mahadev Satyanarayanan. Cloud offload in hostile environments. Technical report, Carnegie Mellon University, 2011.

[53] Jun Han, Yue-Hsun Lin, Adrian Perrig, and Fan Bai. MVSec: Secure and easy-to-use pairing of mobile devices with vehicles. Technical Report CMU-CyLab-14-006, Carnegie Mellon University, May 2014.

[54] Neil B Harrison and Paris Avgeriou. How do architecture patterns and tactics interact? A model and annotation. *Journal of Systems and Software*, 83(10):1735–1758, 2010.

[55] Shih-Hao Hung, Jeng-Peng Shieh, and Chen-Pang Lee. Migrating Android applications to the cloud. *International Journal of Grid and High Performance Computing (IJGHPC)*, 3(2):14–28, 2011.

[56] Shigeru Imai. Task offloading between smartphones and distributed computational resources. Master's thesis, Rensselaer Polytechnic Institute, 2012.

[57] Anantharaman Narayana Iyer et al. Extending Android application programming framework for seamless cloud integration. In *2012 IEEE First International Conference on Mobile Services (MS)*, pages 96–104. IEEE, 2012.

[58] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA 2005)*, pages 109–120, 2005.

[59] Chris Jarabek, David Barrera, and John Aycock. ThinAV: truly lightweight mobile cloud-based anti-malware. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 209–218. ACM, 2012.

[60] R. Jithin and Priya Chandran. Virtual machine isolation. In *Recent Trends in Computer Networks and Distributed Systems Security*, volume 420 of *Communications in Computer and Information Science*, pages 91–102. Springer Berlin Heidelberg, 2014.

[61] JSON.org. Introducing JSON, 2015. URL: `http://www.json.org/`.

[62] Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: a computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services*, pages 59–79. Springer, 2012.

[63] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

[64] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proceedings IEEE INFOCOM 2012*, pages 945–953. IEEE, 2012.

[65] Dejan Kovachev and Ralf Klamma. Framework for computation offloading in mobile cloud computing. *International Journal of Interactive Multimedia and Artificial Intelligence*, 1(7):6–15, 2012.

[66] KPCB. Internet Trends 2015, 2015. URL: `http://www.kpcb.com/internet-trends`.

[67] Mads Daro Kristensen. *Empowering Mobile Devices Through Cyber Foraging*. PhD thesis, Aarhus University, 2010.

[68] Philippe Kruchten, Patricia Lago, and Hans van Vliet. Building up and reasoning about architectural knowledge. In Christine Hofmeister, Ivica Crnkovic, and Ralf Reussner, editors, *Quality of Software Architectures*, volume 4214 of *Lecture Notes in Computer Science*, pages 43–58. Springer Berlin Heidelberg, 2006.

[69] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, February 2013.

[70] Karthik Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, April 2010.

[71] Suman Kundu, Jayanta Mukherjee, Arun K Majumdar, Bandana Majumdar, and Sirsendu Sekhar Ray. Algorithms and heuristics for efficient medical information display in PDA. *Computers in Biology and Medicine*, 37(9):1272–1282, 2007.

[72] Young-Woo Kwon and Eli Tilevich. Reducing the energy consumption of mobile applications behind the scenes. In *Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM 2013)*, 2013.

[73] Patricia Lago and Paris Avgeriou. First workshop on sharing and reusing architectural knowledge. *SIGSOFT Software Engineering Notes*, 31(5):32–36, September 2006.

[74] Byoung-Dai Lee. A framework for seamless execution of mobile applications in the cloud. In *Recent Advances in Computer Science and Information Engineering*, pages 145–153. Springer, 2012.

[75] William Lehr and Lee W. McKnight. Wireless internet access: 3G vs. WiFi? Technical Report 166, MIT Sloan School of Management, 2002.

[76] Grace Lewis, Sebastian Echeverria, Soumya Simanta, James Root, and Ben Bradshaw. Cloudlet-based cyber-foraging in resource-limited environments. *Emerging Research in Cloud Distributed Computing Systems*, pages 92–121, 2015.

[77] Grace Lewis, Patricia Lago, and Giuseppe Procaccianti. Architecture strategies for cyber-foraging: Preliminary results from a systematic literature review. In Paris Avgeriou and Uwe Zdun, editors, *Proceedings of the 8th European Conference on Software Architecture (ECSA 2014)*, volume 8627 of *Lecture Notes in Computer Science*, pages 154–169. Springer International Publishing, 2014.

[78] Grace A Lewis, Sebastian Echeverría, Soumya Simanta, Ben Bradshaw, and James Root. Cloudlet-based cyber-foraging for mobile systems in resource-constrained edge environments. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 412–415. ACM, 2014.

[79] Yue-Hsun Lin, Ahren Studer, Yao-Hsin Chen, Hsu-Chun Hsiao, Li-Hsiang Kuo, Jason Lee, Jonathan M. McCune, King-Hang Wang, Maxwell Krohn, Phen-Lan Lin, Adrian Perrig, Hung-Min Sun, and Bo-Yin Yang. SPATE: Small-group PKI-less authenticated trust establishment. *IEEE Transactions on Mobile Computing*, 9:1666–1681, 2010.

[80] Richard K Lomotey and Ralph Deters. Architectural designs from mobile cloud computing to ubiquitous cloud computing-survey. In *2014 IEEE World Congress on Services (SERVICES)*, pages 418–425. IEEE, 2014.

[81] Allan MacLean, Richard M Young, Victoria ME Bellotti, and Thomas P Moran. Questions, options, and criteria: Elements of design space analysis. *Human–Computer Interaction*, 6(3-4):201–250, 1991.

[82] Jerrid Matthews, Max Chang, ZhiNan Feng, Ravi Srinivas, and Mario Gerla. PowerSense: power aware dengue diagnosis on mobile phones. In *Proceedings of the First ACM Workshop on Mobile Systems, Applications, and Services for Healthcare*, page 6. ACM, 2011.

[83] Alan Messer, Ira Greenberg, Philippe Bernadat, Dejan Milojicic, Deqing Chen, TJ Giuli, and Xiaohui Gu. Towards a distributed platform for resource-constrained devices. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 43–51. IEEE, 2002.

[84] Dominik Messinger and Grace A Lewis. Application virtualizaton as a strategy for cyber foraging in resource-constrained environments. Technical report, Carnegie Mellon Software Engineering Institute, 2013.

[85] mobiForge. Mobile Hardware Statistics 2015, 2015. URL: `https://mobiforge.com/research-analysis/mobile-hardware-statistics-2015`.

[86] mobiForge. Mobile Networks Statistics 2015, 2015. URL: `https://mobiforge.com/research-analysis/mobile-networks-statistics-2015-0`.

[87] mobiForge. Mobile User Behavior Statistics 2015, 2015. URL: `https://mobiforge.com/research-analysis/mobile-user-behaviour-statistics-2015`.

[88] Shivajit Mohapatra and Nalini Venkatasubramanian. Optimizing power using a reconfigurable middleware. Technical report, UC Irvine, 2003.

[89] Mozilla Developer Network. Firefox OS, 2015. URL: `https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS`.

[90] Mozilla Developer Network. IndexedDB, 2016. URL: `https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API`.

[91] Object Refinery Limited. JFreeChart, 2014. URL: `http://www.jfree.org/jfreechart/`.

[92] MinHwan Ok, Ja-Won Seo, and Myong-soon Park. A distributed resource furnishing to offload resource-constrained devices in cyber foraging toward pervasive computing. In *Network-Based Information Systems*, pages 416–425. Springer, 2007.

[93] Michael J O'Sullivan and Dan Grigoras. The cloud personal assistant for providing services to mobile clients. In *2013 IEEE 7th International Symposium on Service Oriented System Engineering (SOSE)*, pages 478–485, 2013.

[94] OWM Inc. OpenWeatherMap, 2015. URL: `http://openweathermap.org/`.

[95] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: A research roadmap. *International Journal of Cooperative Information Systems*, 17(02):223–255, 2008.

[96] Sehoon Park, Youngil Choi, Qichen Chen, and H.Y. Yeom. SOME: Selective offloading for a mobile computing environment. In *2012 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 588–591, 2012.

[97] Alex Sandy Pentland, Richard Fletcher, and Amir Hasson. Daknet: Rethinking connectivity in developing nations. *Computer*, 37(1):78–83, 2004.

[98] Theophilos Phokas, Hariton Efstathiades, George Pallis, and MariosD. Dikaiakos. Feel the world: A mobile framework for participatory sensing. In Florian Daniel, George A. Papadopoulos, and Philippe Thiran, editors, *Mobile Web Information Systems*, volume 8093 of *Lecture Notes in Computer Science*, pages 143–156. Springer Berlin Heidelberg, 2013.

[99] Giuseppe Procaccianti, Hector Fernandez, and Patricia Lago. Empirical evaluation of two best practices for energy-efficient software development. *Journal of Systems and Software*, 2016.

[100] Lingjun Pu, Jingdong Xu, Xing Jin, and Jianzhong Zhang. SmartVirtCloud: virtual cloud assisted application offloading execution at mobile devices' discretion. In *2013 IEEE Wireless Communications and Networking Conference (WCNC): SERVICES and APPLICATIONS*, 2013.

[101] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. Odessa: enabling interactive perception applications on mobile devices. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 43–56, New York, NY, USA, 2011. ACM.

[102] Kiran K Rachuri. *Smartphones based Social Sensing: Adaptive Sampling, Sensing and Computation Offloading.* PhD thesis, University of Cambridge, 2012.

[103] M Reza Rahimi, Nalini Venkatasubramanian, Sharad Mehrotra, and Athanasios V Vasilakos. MAPCloud: mobile applications on an elastic and scalable 2-tier cloud architecture. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, pages 83–90. IEEE Computer Society, 2012.

[104] Raspberry Pi Foundation. Raspberry Pi 2 Model B, 2015. URL: `https://www.raspberrypi.org/products/raspberry-pi-2-model-b/`.

[105] Raspbian. Welcome to Raspbian, 2015. URL: `https://www.raspbian.org/`.

[106] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[107] M. Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10–17, Aug 2001.

[108] Mahadev Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.

[109] Mahadev Satyanarayanan, Rolf Schuster, Maria Ebling, Gerhard Fettweis, Hannu Flinck, Kaustubh Joshi, and Krishan Sabnani. An open ecosystem for mobile-cloud convergence. *IEEE Communications Magazine*, 53(3):63–70, 2015.

[110] Mahadev Satyanarayanan, Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, Wenlu Hu, and Brandon Amos. Edge analytics in the internet of things. *IEEE Pervasive Computing*, (2):24–31, 2015.

[111] Cong Shi, Pranesh Pandurangan, Kangqi Ni, Juyuan Yang, Mostafa Ammar, Mayur Naik, and Ellen Zegura. IC-Cloud: Computation offloading to an intermittently-connected cloud. Technical report, Georgia Institute of Technology, 2013.

[112] Joao Nuno Silva, Luis Veiga, and Paulo Ferreira. SPADE: scheduler for parallel and distributed execution from mobile devices. In *Proceedings of the 6th International Workshop on Middleware for Pervasive and Ad-hoc Computing*, pages 25–30. ACM, 2008.

[113] Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, and Mahadev Satyanarayanan. Scalable crowd-sourcing of video from mobile devices. In *Proceedings of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 139–152, New York, NY, USA, 2013. ACM.

[114] Ya-Yunn Su and Jason Flinn. Slingshot: deploying stateful services in wireless hotspots. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, MobiSys '05, pages 79–92, New York, NY, USA, 2005. ACM.

[115] The Apache Software Foundation. Apache Commons Codec, 2014. URL: `https://commons.apache.org/proper/commons-codec/`.

[116] The jQuery Foundation. jQuery, 2015. URL: `https://jquery.com/`.

[117] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. AIOLOS: Middleware for improving mobile application performance through cyber foraging. *Journal of Systems and Software*, 85(11):2629–2639, 2012.

[118] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer Science & Business Media, 2012.

[119] Yu Xiao, Pieter Simoens, Padmanabhan Pillai, Kiryong Ha, and Mahadev Satyanarayanan. Lowering the barriers to large-scale mobile crowdsensing. In *Mobile Computing Systems and Applications*, 2013.

[120] Fan Yang, Zhengping Qian, Xiuwei Chen, Ivan Beschastnikh, Li Zhuang, Lidong Zhou, and Jacky Shen. Sonora: A platform for continuous mobile-cloud computing. Technical report, Technical Report. Microsoft Research Asia, 2012.

[121] Kun Yang, Shumao Ou, and Hsiao-Hwa Chen. On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *IEEE Communications Magazine*, 46(1):56–63, 2008.

[122] Lei Yang, Jiannong Cao, Yin Yuan, Tao Li, Andy Han, and Alvin Chan. A framework for partitioning and execution of data stream applications in mobile cloud computing. *ACM SIGMETRICS Performance Evaluation Review*, 40(4):23–32, 2013.

[123] Ping Yu, Xiaoxing Ma, Jiannong Cao, and Jian Lu. Application mobility in pervasive computing: A survey. *Pervasive and Mobile Computing*, 9:2–17, 2012.

[124] Uwe Zdun. Systematic pattern selection using pattern language grammars and design space analysis. *Software: Practice and Experience*, 37(9):983–1016, 2007.

[125] Gu Zhang. From feature phones to smartphones, the road ahead. GSMA Intelligence, 2015. URL: `https://gsmaintelligence.com/research/2015/01/from-feature-phones-to-smartphones-the-road-ahead/456/`.

[126] Xinwen Zhang, Won Jeon, Simon Gibbs, and Anugeetha Kunjithapatham. Elastic HTML5: Workload offloading using cloud-based web workers and storages for mobile devices. In *Mobile Computing, Applications, and Services*, pages 373–381. Springer, 2012.

[127] Xinwen Zhang, Anugeetha Kunjithapatham, Sangoh Jeong, and Simon Gibbs. Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mobile Networks and Applications*, 16(3):270–284, 2011.

[128] Yang Zhang, Xue-tao Guan, Tao Huang, and Xu Cheng. A heterogeneous auto-offloading framework based on web browser for resource-constrained devices. In *Fourth International Conference on Internet and Web Applications and Services (ICIW'09)*, pages 193–199. IEEE, 2009.

[129] Ying Zhang, Gang Huang, Wei Zhang, Xuanzhe Liu, and Hong Mei. Towards module-based automatic partitioning of Java applications. *Frontiers of Computer Science*, 6(6):725–740, 2012.

[130] Olaf Zimmermann, Lukas Wegmann, Heiko Koziolek, and Thomas Goldschmidt. Architectural decision guidance across projects-problem space modeling, decision backlog management and cloud computing knowledge. In *Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on*, pages 85–94. IEEE, 2015.

[131] ZTE Corporation. ZTE Open C, 2015. URL: `http://www.ztedevice.com/product/b8fcc021-3e5b-4a96-8161-47ee8d4f42f1.html`.

SIKS Dissertation Series

2009

1 Rasa Jurgelenaite (RUN) *Symmetric Causal Independence Models*

2 Willem Robert van Hage (VUA) *Evaluating Ontology-Alignment Techniques*

3 Hans Stol (UvT) *A Framework for Evidence-based Policy Making Using IT*

4 Josephine Nabukenya (RUN) *Improving the Quality of Organisational Policy Making using Collaboration Engineering*

5 Sietse Overbeek (RUN) *Bridging Supply and Demand for Knowledge Intensive Tasks: Based on Knowledge, Cognition, and Quality*

6 Muhammad Subianto (UU) *Understanding Classification*

7 Ronald Poppe (UT) *Discriminative Vision-Based Recovery and Recognition of Human Motion*

8 Volker Nannen (VUA) *Evolutionary Agent-Based Policy Analysis in Dynamic Environments*

9 Benjamin Kanagwa (RUN) *Design, Discovery and Construction of Service-oriented Systems*

10 Jan Wielemaker (UvA) *Logic programming for knowledge-intensive interactive applications*

11 Alexander Boer (UvA) *Legal Theory, Sources of Law & the Semantic Web*

12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin) *Operating Guidelines for Services*

13 Steven de Jong (UM) *Fairness in Multi-Agent Systems*

14 Maksym Korotkiy (VUA) *From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)*

15 Rinke Hoekstra (UvA) *Ontology Representation: Design Patterns and Ontologies that Make Sense*

16 Fritz Reul (UvT) *New Architectures in Computer Chess*

17 Laurens van der Maaten (UvT) *Feature Extraction from Visual Data*

18 Fabian Groffen (CWI) *Armada, An Evolving Database System*

19 Valentin Robu (CWI) *Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets*

20 Bob van der Vecht (UU) *Adjustable Autonomy: Controling Influences on Decision Making*

21 Stijn Vanderlooy (UM) *Ranking and Reliable Classification*

22 Pavel Serdyukov (UT) *Search For Expertise: Going beyond direct evidence*

23 Peter Hofgesang (VUA) *Modelling Web Usage in a Changing Environment*

24 Annerieke Heuvelink (VUA) *Cognitive Models for Training Simulations*

25 Alex van Ballegooij (CWI) *RAM: Array Database Management through Relational Mapping*

26 Fernando Koch (UU) *An Agent-Based Model for the Development of Intelligent Mobile Services*

27 Christian Glahn (OU) *Contextual Support of social Engagement and Reflection on the Web*

28 Sander Evers (UT) *Sensor Data Management with Probabilistic Models*

29 Stanislav Pokraev (UT) *Model-Driven Semantic Integration of Service-Oriented Applications*

30 Marcin Zukowski (CWI) *Balancing vectorized query execution with bandwidth-optimized storage*

31 Sofiya Katrenko (UvA) *A Closer Look at Learning Relations from Text*

32 Rik Farenhorst (VUA) *Architectural Knowledge Management: Supporting Architects and Auditors*

33 Khiet Truong (UT) *How Does Real Affect Affect Affect Recognition In Speech?*

34 Inge van de Weerd (UU) *Advancing in Software Product Management: An Incremental Method Engineering Approach*

35 Wouter Koelewijn (UL) *Privacy en Politiegegevens: Over geautomatiseerde normatieve informatie-uitwisseling*

36 Marco Kalz (OUN) *Placement Support for Learners in Learning Networks*

37 Hendrik Drachsler (OUN) *Navigation Support for Learners in Informal Learning Networks*

38 Riina Vuorikari (OU) *Tags and self-organisation: a metadata ecology for learning resources in a multilingual context*

39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin) *Service Substitution: A Behavioral Approach Based on Petri Nets*

40 Stephan Raaijmakers (UvT) *Multinomial Language Learning: Investigations into the Geometry of Language*

41 Igor Berezhnyy (UvT) *Digital Analysis of Paintings*

42 Toine Bogers (UvT) *Recommender Systems for Social Bookmarking*

43 Virginia Nunes Leal Franqueira (UT) *Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients*

44 Roberto Santana Tapia (UT) *Assessing Business-IT Alignment in Networked Organizations*

45 Jilles Vreeken (UU) *Making Pattern Mining Useful*

46 Loredana Afanasiev (UvA) *Querying XML: Benchmarks and Recursion*

2010

1 Matthijs van Leeuwen (UU) *Patterns that Matter*

2 Ingo Wassink (UT) *Work flows in Life Science*

3 Joost Geurts (CWI) *A Document Engineering Model and Processing Framework for Multimedia documents*

4 Olga Kulyk (UT) *Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments*

5 Claudia Hauff (UT) *Predicting the Effectiveness of Queries and Retrieval Systems*

6 Sander Bakkes (UvT) *Rapid Adaptation of Video Game AI*

7 Wim Fikkert (UT) *Gesture interaction at a Distance*

8 Krzysztof Siewicz (UL) *Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments*

9 Hugo Kielman (UL) *A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging*

10 Rebecca Ong (UL) *Mobile Communication and Protection of Children*

11 Adriaan Ter Mors (TUD) *The world according to MARP: Multi-Agent Route Planning*

12 Susan van den Braak (UU) *Sensemaking software for crime analysis*

13 Gianluigi Folino (RUN) *High Performance Data Mining using Bio-inspired techniques*

14 Sander van Splunter (VUA) *Automated Web Service Reconfiguration*

15 Lianne Bodenstaff (UT) *Managing Dependency Relations in Inter-Organizational Models*

16 Sicco Verwer (TUD) *Efficient Identification of Timed Automata, theory and practice*

17 Spyros Kotoulas (VUA) *Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications*

18 Charlotte Gerritsen (VUA) *Caught in the Act: Investigating Crime by Agent-Based Simulation*

19 Henriette Cramer (UvA) *People's Responses to Autonomous and Adaptive Systems*

20 Ivo Swartjes (UT) *Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative*

21 Harold van Heerde (UT) *Privacy-aware data management by means of data degradation*

22 Michiel Hildebrand (CWI) *End-user Support for Access to Heterogeneous Linked Data*
23 Bas Steunebrink (UU) *The Logical Structure of Emotions*
24 Zulfiqar Ali Memon (VUA) *Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective*
25 Ying Zhang (CWI) *XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines*
26 Marten Voulon (UL) *Automatisch contracteren*
27 Arne Koopman (UU) *Characteristic Relational Patterns*
28 Stratos Idreos (CWI) *Database Cracking: Towards Auto-tuning Database Kernels*
29 Marieke van Erp (UvT) *Accessing Natural History: Discoveries in data cleaning, structuring, and retrieval*
30 Victor de Boer (UvA) *Ontology Enrichment from Heterogeneous Sources on the Web*
31 Marcel Hiel (UvT) *An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems*
32 Robin Aly (UT) *Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval*
33 Teduh Dirgahayu (UT) *Interaction Design in Service Compositions*
34 Dolf Trieschnigg (UT) *Proof of Concept: Concept-based Biomedical Information Retrieval*
35 Jose Janssen (OU) *Paving the Way for Lifelong Learning: Facilitating competence development through a learning path specification*
36 Niels Lohmann (TUe) *Correctness of services and their composition*
37 Dirk Fahland (TUe) *From Scenarios to components*
38 Ghazanfar Farooq Siddiqui (VUA) *Integrative modeling of emotions in virtual agents*
39 Mark van Assem (VUA) *Converting and Integrating Vocabularies for the Semantic Web*
40 Guillaume Chaslot (UM) *Monte-Carlo Tree Search*
41 Sybren de Kinderen (VUA) *Needs-driven service bundling in a multi-supplier setting: the computational e3-service approach*
42 Peter van Kranenburg (UU) *A Computational Approach to Content-Based Retrieval of Folk Song Melodies*
43 Pieter Bellekens (TUe) *An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain*
44 Vasilios Andrikopoulos (UvT) *A theory and model for the evolution of software services*
45 Vincent Pijpers (VUA) *e3alignment: Exploring Inter-Organizational Business-ICT Alignment*
46 Chen Li (UT) *Mining Process Model Variants: Challenges, Techniques, Examples*
47 Jahn-Takeshi Saito (UM) *Solving difficult game positions*
48 Bouke Huurnink (UvA) *Search in Audiovisual Broadcast Archives*
49 Alia Khairia Amin (CWI) *Understanding and supporting information seeking tasks in multiple sources*
50 Peter-Paul van Maanen (VUA) *Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention*
51 Edgar Meij (UvA) *Combining Concepts and Language Models for Information Access*

2011

1 Botond Cseke (RUN) *Variational Algorithms for Bayesian Inference in Latent Gaussian Models*
2 Nick Tinnemeier (UU) *Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language*
3 Jan Martijn van der Werf (TUe) *Compositional Design and Verification of Component-Based Information Systems*

4 Hado van Hasselt (UU) *Insights in Reinforcement Learning: Formal analysis and empirical evaluation of temporal-difference*

5 Base van der Raadt (VUA) *Enterprise Architecture Coming of Age: Increasing the Performance of an Emerging Discipline*

6 Yiwen Wang (TUe) *Semantically-Enhanced Recommendations in Cultural Heritage*

7 Yujia Cao (UT) *Multimodal Information Presentation for High Load Human Computer Interaction*

8 Nieske Vergunst (UU) *BDI-based Generation of Robust Task-Oriented Dialogues*

9 Tim de Jong (OU) *Contextualised Mobile Media for Learning*

10 Bart Bogaert (UvT) *Cloud Content Contention*

11 Dhaval Vyas (UT) *Designing for Awareness: An Experience-focused HCI Perspective*

12 Carmen Bratosin (TUe) *Grid Architecture for Distributed Process Mining*

13 Xiaoyu Mao (UvT) *Airport under Control. Multiagent Scheduling for Airport Ground Handling*

14 Milan Lovric (EUR) *Behavioral Finance and Agent-Based Artificial Markets*

15 Marijn Koolen (UvA) *The Meaning of Structure: the Value of Link Evidence for Information Retrieval*

16 Maarten Schadd (UM) *Selective Search in Games of Different Complexity*

17 Jiyin He (UvA) *Exploring Topic Structure: Coherence, Diversity and Relatedness*

18 Mark Ponsen (UM) *Strategic Decision-Making in complex games*

19 Ellen Rusman (OU) *The Mind ' s Eye on Personal Profiles*

20 Qing Gu (VUA) *Guiding service-oriented software engineering: A view-based approach*

21 Linda Terlouw (TUD) *Modularization and Specification of Service-Oriented Systems*

22 Junte Zhang (UvA) *System Evaluation of Archival Description and Access*

23 Wouter Weerkamp (UvA) *Finding People and their Utterances in Social Media*

24 Herwin van Welbergen (UT) *Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior*

25 Syed Waqar ul Qounain Jaffry (VUA) *Analysis and Validation of Models for Trust Dynamics*

26 Matthijs Aart Pontier (VUA) *Virtual Agents for Human Communication: Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots*

27 Aniel Bhulai (VUA) *Dynamic website optimization through autonomous management of design patterns*

28 Rianne Kaptein (UvA) *Effective Focused Retrieval by Exploiting Query Context and Document Structure*

29 Faisal Kamiran (TUe) *Discrimination-aware Classification*

30 Egon van den Broek (UT) *Affective Signal Processing (ASP): Unraveling the mystery of emotions*

31 Ludo Waltman (EUR) *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality*

32 Nees-Jan van Eck (EUR) *Methodological Advances in Bibliometric Mapping of Science*

33 Tom van der Weide (UU) *Arguing to Motivate Decisions*

34 Paolo Turrini (UU) *Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations*

35 Maaike Harbers (UU) *Explaining Agent Behavior in Virtual Training*

36 Erik van der Spek (UU) *Experiments in serious game design: a cognitive approach*

37 Adriana Burlutiu (RUN) *Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference*

38 Nyree Lemmens (UM) *Bee-inspired Distributed Optimization*

39 Joost Westra (UU) *Organizing Adaptation using Agents in Serious Games*
40 Viktor Clerc (VUA) *Architectural Knowledge Management in Global Software Development*
41 Luan Ibraimi (UT) *Cryptographically Enforced Distributed Data Access Control*
42 Michal Sindlar (UU) *Explaining Behavior through Mental State Attribution*
43 Henk van der Schuur (UU) *Process Improvement through Software Operation Knowledge*
44 Boris Reuderink (UT) *Robust Brain-Computer Interfaces*
45 Herman Stehouwer (UvT) *Statistical Language Models for Alternative Sequence Selection*
46 Beibei Hu (TUD) *Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work*
47 Azizi Bin Ab Aziz (VUA) *Exploring Computational Models for Intelligent Support of Persons with Depression*
48 Mark Ter Maat (UT) *Response Selection and Turn-taking for a Sensitive Artificial Listening Agent*
49 Andreea Niculescu (UT) *Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality*

2012

1 Terry Kakeeto (UvT) *Relationship Marketing for SMEs in Uganda*
2 Muhammad Umair (VUA) *Adaptivity, emotion, and Rationality in Human and Ambient Agent Models*
3 Adam Vanya (VUA) *Supporting Architecture Evolution by Mining Software Repositories*
4 Jurriaan Souer (UU) *Development of Content Management System-based Web Applications*
5 Marijn Plomp (UU) *Maturing Interorganisational Information Systems*
6 Wolfgang Reinhardt (OU) *Awareness Support for Knowledge Workers in Research Networks*
7 Rianne van Lambalgen (VUA) *When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions*
8 Gerben de Vries (UvA) *Kernel Methods for Vessel Trajectories*
9 Ricardo Neisse (UT) *Trust and Privacy Management Support for Context-Aware Service Platforms*
10 David Smits (TUe) *Towards a Generic Distributed Adaptive Hypermedia Environment*
11 J. C. B. Rantham Prabhakara (TUe) *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics*
12 Kees van der Sluijs (TUe) *Model Driven Design and Data Integration in Semantic Web Information Systems*
13 Suleman Shahid (UvT) *Fun and Face: Exploring non-verbal expressions of emotion during playful interactions*
14 Evgeny Knutov (TUe) *Generic Adaptation Framework for Unifying Adaptive Web-based Systems*
15 Natalie van der Wal (VUA) *Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes*
16 Fiemke Both (VUA) *Helping people by understanding them: Ambient Agents supporting task execution and depression treatment*
17 Amal Elgammal (UvT) *Towards a Comprehensive Framework for Business Process Compliance*
18 Eltjo Poort (VUA) *Improving Solution Architecting Practices*
19 Helen Schonenberg (TUe) *What's Next? Operational Support for Business Process Execution*
20 Ali Bahramisharif (RUN) *Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing*
21 Roberto Cornacchia (TUD) *Querying Sparse Matrices for Information Retrieval*

22 Thijs Vis (UvT) *Intelligence, politie en veiligheidsdienst: verenigbare grootheden?*
23 Christian Muehl (UT) *Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction*
24 Laurens van der Werff (UT) *Evaluation of Noisy Transcripts for Spoken Document Retrieval*
25 Silja Eckartz (UT) *Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application*
26 Emile de Maat (UvA) *Making Sense of Legal Text*
27 Hayrettin Gurkok (UT) *Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games*
28 Nancy Pascall (UvT) *Engendering Technology Empowering Women*
29 Almer Tigelaar (UT) *Peer-to-Peer Information Retrieval*
30 Alina Pommeranz (TUD) *Designing Human-Centered Systems for Reflective Decision Making*
31 Emily Bagarukayo (RUN) *A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure*
32 Wietske Visser (TUD) *Qualitative multi-criteria preference representation and reasoning*
33 Rory Sie (OUN) *Coalitions in Cooperation Networks (COCOON)*
34 Pavol Jancura (RUN) *Evolutionary analysis in PPI networks and applications*
35 Evert Haasdijk (VUA) *Never Too Old To Learn: On-line Evolution of Controllers in Swarm- and Modular Robotics*
36 Denis Ssebugwawo (RUN) *Analysis and Evaluation of Collaborative Modeling Processes*
37 Agnes Nakakawa (RUN) *A Collaboration Process for Enterprise Architecture Creation*
38 Selmar Smit (VUA) *Parameter Tuning and Scientific Testing in Evolutionary Algorithms*
39 Hassan Fatemi (UT) *Risk-aware design of value and coordination networks*
40 Agus Gunawan (UvT) *Information Access for SMEs in Indonesia*
41 Sebastian Kelle (OU) *Game Design Patterns for Learning*
42 Dominique Verpoorten (OU) *Reflection Amplifiers in self-regulated Learning*
43 Anna Tordai (VUA) *On Combining Alignment Techniques*
44 Benedikt Kratz (UvT) *A Model and Language for Business-aware Transactions*
45 Simon Carter (UvA) *Exploration and Exploitation of Multilingual Data for Statistical Machine Translation*
46 Manos Tsagkias (UvA) *Mining Social Media: Tracking Content and Predicting Behavior*
47 Jorn Bakker (TUe) *Handling Abrupt Changes in Evolving Time-series Data*
48 Michael Kaisers (UM) *Learning against Learning: Evolutionary dynamics of reinforcement learning algorithms in strategic interactions*
49 Steven van Kervel (TUD) *Ontologogy driven Enterprise Information Systems Engineering*
50 Jeroen de Jong (TUD) *Heuristics in Dynamic Sceduling: a practical framework with a case study in elevator dispatching*

2013

1 Viorel Milea (EUR) *News Analytics for Financial Decision Support*
2 Erietta Liarou (CWI) *MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing*
3 Szymon Klarman (VUA) *Reasoning with Contexts in Description Logics*
4 Chetan Yadati (TUD) *Coordinating autonomous planning and scheduling*
5 Dulce Pumareja (UT) *Groupware Requirements Evolutions Patterns*
6 Romulo Goncalves (CWI) *The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience*
7 Giel van Lankveld (UvT) *Quantifying Individual Player Differences*

8 Robbert-Jan Merk (VUA) *Making enemies: cognitive modeling for opponent agents in fighter pilot simulators*

9 Fabio Gori (RUN) *Metagenomic Data Analysis: Computational Methods and Applications*

10 Jeewanie Jayasinghe Arachchige (UvT) *A Unified Modeling Framework for Service Design*

11 Evangelos Pournaras (TUD) *Multi-level Reconfigurable Self-organization in Overlay Services*

12 Marian Razavian (VUA) *Knowledge-driven Migration to Services*

13 Mohammad Safiri (UT) *Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly*

14 Jafar Tanha (UvA) *Ensemble Approaches to Semi-Supervised Learning Learning*

15 Daniel Hennes (UM) *Multiagent Learning: Dynamic Games and Applications*

16 Eric Kok (UU) *Exploring the practical benefits of argumentation in multi-agent deliberation*

17 Koen Kok (VUA) *The PowerMatcher: Smart Coordination for the Smart Electricity Grid*

18 Jeroen Janssens (UvT) *Outlier Selection and One-Class Classification*

19 Renze Steenhuizen (TUD) *Coordinated Multi-Agent Planning and Scheduling*

20 Katja Hofmann (UvA) *Fast and Reliable Online Learning to Rank for Information Retrieval*

21 Sander Wubben (UvT) *Text-to-text generation by monolingual machine translation*

22 Tom Claassen (RUN) *Causal Discovery and Logic*

23 Patricio de Alencar Silva (UvT) *Value Activity Monitoring*

24 Haitham Bou Ammar (UM) *Automated Transfer in Reinforcement Learning*

25 Agnieszka Anna Latoszek-Berendsen (UM) *Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System*

26 Alireza Zarghami (UT) *Architectural Support for Dynamic Homecare Service Provisioning*

27 Mohammad Huq (UT) *Inference-based Framework Managing Data Provenance*

28 Frans van der Sluis (UT) *When Complexity becomes Interesting: An Inquiry into the Information eXperience*

29 Iwan de Kok (UT) *Listening Heads*

30 Joyce Nakatumba (TUe) *Resource-Aware Business Process Management: Analysis and Support*

31 Dinh Khoa Nguyen (UvT) *Blueprint Model and Language for Engineering Cloud Applications*

32 Kamakshi Rajagopal (OUN) *Networking For Learning: The role of Networking in a Lifelong Learner's Professional Development*

33 Qi Gao (TUD) *User Modeling and Personalization in the Microblogging Sphere*

34 Kien Tjin-Kam-Jet (UT) *Distributed Deep Web Search*

35 Abdallah El Ali (UvA) *Minimal Mobile Human Computer Interaction*

36 Than Lam Hoang (TUe) *Pattern Mining in Data Streams*

37 Dirk Börner (OUN) *Ambient Learning Displays*

38 Eelco den Heijer (VUA) *Autonomous Evolutionary Art*

39 Joop de Jong (TUD) *A Method for Enterprise Ontology based Design of Enterprise Information Systems*

40 Pim Nijssen (UM) *Monte-Carlo Tree Search for Multi-Player Games*

41 Jochem Liem (UvA) *Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning*

42 Léon Planken (TUD) *Algorithms for Simple Temporal Reasoning*

43 Marc Bron (UvA) *Exploration and Contextualization through Interaction and Concepts*

2014

1 Nicola Barile (UU) *Studies in Learning Monotone Models from Data*
2 Fiona Tuliyano (RUN) *Combining System Dynamics with a Domain Modeling Method*
3 Sergio Raul Duarte Torres (UT) *Information Retrieval for Children: Search Behavior and Solutions*
4 Hanna Jochmann-Mannak (UT) *Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation*
5 Jurriaan van Reijsen (UU) *Knowledge Perspectives on Advancing Dynamic Capability*
6 Damian Tamburri (VUA) *Supporting Networked Software Development*
7 Arya Adriansyah (TUe) *Aligning Observed and Modeled Behavior*
8 Samur Araujo (TUD) *Data Integration over Distributed and Heterogeneous Data Endpoints*
9 Philip Jackson (UvT) *Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language*
10 Ivan Salvador Razo Zapata (VUA) *Service Value Networks*
11 Janneke van der Zwaan (TUD) *An Empathic Virtual Buddy for Social Support*
12 Willem van Willigen (VUA) *Look Ma, No Hands: Aspects of Autonomous Vehicle Control*
13 Arlette van Wissen (VUA) *Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains*
14 Yangyang Shi (TUD) *Language Models With Meta-information*
15 Natalya Mogles (VUA) *Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare*
16 Krystyna Milian (VUA) *Supporting trial recruitment and design by automatically interpreting eligibility criteria*
17 Kathrin Dentler (VUA) *Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability*
18 Mattijs Ghijsen (VUA) *Methods and Models for the Design and Study of Dynamic Agent Organizations*
19 Vinicius Ramos (TUe) *Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support*
20 Mena Habib (UT) *Named Entity Extraction and Disambiguation for Informal Text: The Missing Link*
21 Kassidy Clark (TUD) *Negotiation and Monitoring in Open Environments*
22 Marieke Peeters (UU) *Personalized Educational Games: Developing agent-supported scenario-based training*
23 Eleftherios Sidirourgos (UvA/CWI) *Space Efficient Indexes for the Big Data Era*
24 Davide Ceolin (VUA) *Trusting Semi-structured Web Data*
25 Martijn Lappenschaar (RUN) *New network models for the analysis of disease interaction*
26 Tim Baarslag (TUD) *What to Bid and When to Stop*
27 Rui Jorge Almeida (EUR) *Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty*
28 Anna Chmielowiec (VUA) *Decentralized k-Clique Matching*
29 Jaap Kabbedijk (UU) *Variability in Multi-Tenant Enterprise Software*
30 Peter de Cock (UvT) *Anticipating Criminal Behaviour*
31 Leo van Moergestel (UU) *Agent Technology in Agile Multiparallel Manufacturing and Product Support*
32 Naser Ayat (UvA) *On Entity Resolution in Probabilistic Data*
33 Tesfa Tegegne (RUN) *Service Discovery in eHealth*
34 Christina Manteli (VUA) *The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems*

35 Joost van Ooijen (UU) *Cognitive Agents in Virtual Worlds: A Middleware Design Approach*

36 Joos Buijs (TUe) *Flexible Evolutionary Algorithms for Mining Structured Process Models*

37 Maral Dadvar (UT) *Experts and Machines United Against Cyberbullying*

38 Danny Plass-Oude Bos (UT) *Making brain-computer interfaces better: improving usability through post-processing*

39 Jasmina Maric (UvT) *Web Communities, Immigration, and Social Capital*

40 Walter Omona (RUN) *A Framework for Knowledge Management Using ICT in Higher Education*

41 Frederic Hogenboom (EUR) *Automated Detection of Financial Events in News Text*

42 Carsten Eijckhof (CWI/TUD) *Contextual Multidimensional Relevance Models*

43 Kevin Vlaanderen (UU) *Supporting Process Improvement using Method Increments*

44 Paulien Meesters (UvT) *Intelligent Blauw: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden*

45 Birgit Schmitz (OUN) *Mobile Games for Learning: A Pattern-Based Approach*

46 Ke Tao (TUD) *Social Web Data Analytics: Relevance, Redundancy, Diversity*

47 Shangsong Liang (UvA) *Fusion and Diversification in Information Retrieval*

2015

1 Niels Netten (UvA) *Machine Learning for Relevance of Information in Crisis Response*

2 Faiza Bukhsh (UvT) *Smart auditing: Innovative Compliance Checking in Customs Controls*

3 Twan van Laarhoven (RUN) *Machine learning for network data*

4 Howard Spoelstra (OUN) *Collaborations in Open Learning Environments*

5 Christoph Bösch (UT) *Cryptographically Enforced Search Pattern Hiding*

6 Farideh Heidari (TUD) *Business Process Quality Computation: Computing Non-Functional Requirements to Improve Business Processes*

7 Maria-Hendrike Peetz (UvA) *Time-Aware Online Reputation Analysis*

8 Jie Jiang (TUD) *Organizational Compliance: An agent-based model for designing and evaluating organizational interactions*

9 Randy Klaassen (UT) *HCI Perspectives on Behavior Change Support Systems*

10 Henry Hermans (OUN) *OpenU: design of an integrated system to support lifelong learning*

11 Yongming Luo (TUe) *Designing algorithms for big graph datasets: A study of computing bisimulation and joins*

12 Julie M. Birkholz (VUA) *Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks*

13 Giuseppe Procaccianti (VUA) *Energy-Efficient Software*

14 Bart van Straalen (UT) *A cognitive approach to modeling bad news conversations*

15 Klaas Andries de Graaf (VUA) *Ontology-based Software Architecture Documentation*

16 Changyun Wei (UT) *Cognitive Coordination for Cooperative Multi-Robot Teamwork*

17 André van Cleeff (UT) *Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs*

18 Holger Pirk (CWI) *Waste Not, Want Not!: Managing Relational Data in Asymmetric Memories*

19 Bernardo Tabuenca (OUN) *Ubiquitous Technology for Lifelong Learners*

20 Loïs Vanhée (UU) *Using Culture and Values to Support Flexible Coordination*

21 Sibren Fetter (OUN) *Using Peer-Support to Expand and Stabilize Online Learning*

22 Zhemin Zhu (UT) *Co-occurrence Rate Networks*

23 Luit Gazendam (VUA) *Cataloguer Support in Cultural Heritage*

24 Richard Berendsen (UvA) *Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation*
25 Steven Woudenberg (UU) *Bayesian Tools for Early Disease Detection*
26 Alexander Hogenboom (EUR) *Sentiment Analysis of Text Guided by Semantics and Structure*
27 Sándor Héman (CWI) *Updating Compressed Column Stores*
28 Janet Bagorogoza (TiU) *Knowledge Management and High Performance: The Uganda Financial Institutions Model for HPO*
29 Hendrik Baier (UM) *Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains*
30 Kiavash Bahreini (OU) *Real-time Multimodal Emotion Recognition in E-Learning*
31 Yakup Koç (TUD) *On the Robustness of Power Grids*
32 Jerome Gard (UL) *Corporate Venture Management in SMEs*
33 Frederik Schadd (TUD) *Ontology Mapping with Auxiliary Resources*
34 Victor de Graaf (UT) *Gesocial Recommender Systems*
35 Jungxao Xu (TUD) *Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction*

2016

1 Syed Saiden Abbas (RUN) *Recognition of Shapes by Humans and Machines*
2 Michiel Christiaan Meulendijk (UU) *Optimizing Medication Reviews through Decision Support: Prescribing a Better Pill to Swallow*
3 Maya Sappelli (RUN) *Knowledge Work in Context: User Centered Knowledge Worker Support*
4 Laurens Rietveld (VUA) *Publishing and Consuming Linked Data*
5 Evgeny Sherkhonov (UVA) *Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers*
6 Michel Wilson (TUD) *Robust Scheduling in an Uncertain Environment*
7 Jeroen de Man (VUA) *Measuring and Modeling Negative Emotions for Virtual Training*
8 Matje van de Camp (TiU) *A Link to the Past: Constructing Historical Social Networks from Unstructured Data*
9 Archana Nottamkandath (VUA) *Trusting Crowdsourced Information on Cultural Artefacts*
10 George Karafotias (VUA) *Parameter Control for Evolutionary Algorithms*
11 Anne Schuth (UVA) *Search Engines that Learn from Their Users*
12 Max Knobbout (UU) *Logics for Modelling and Verifying Normative Multi-Agent Systems*
13 Nana Baah Gyan (VUA) *The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach*
14 Ravi Khadka (UU) *Revisiting Legacy Software System Modernization*
15 Steffen Michels (RUN) *Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments*
16 Guangliang Li (UVA) *Socially Intelligent Autonomous Agents that Learn from Human Reward*
17 Berend Weel (VUA) *Towards Embodied Evolution of Robot Organisms*
18 Albert Meroño Peñuela (VUA) *Refining Statistical Data on the Web*
19 Julia Efremova (Tu/e) *Mining Social Structures from Genealogical Data*
20 Daan Odijk (UVA) *Context & Semantics in News & Web Search*
21 Alejandro Moreno Célleri (UT) *From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground*
22 Grace Lewis (VUA) *Software Architecture Strategies for Cyber-Foraging Systems*