ADAPTIVE HIGH LEVEL CONTEXT REASONING

IN PERVASIVE ENVIRONMENTS

by

BRIDGET B. BEAMON

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2011

To my parents Lynda and Earl, my husband Johnnell, and my daughter Havalynn.

ACKNOWLEDGEMENTS

This completed work is dedicated to my family. Their unwavering support and commitment to my success has made this possible. Life did not permit my sister Opal to see me complete my Ph. D. Still, she will forever be my biggest fan; one who believed in me more that I believed in myself. Knowing that completing this work forges path for my daughter to follow makes the sacrifices worthwhile. My husband, Johnnell, has supported every endeavor or mine without reservation. I am truly blessed to have the gift of such loving and supportive family. So, my success is not only for myself, but also for those I love.

I would never have been able to complete this work without the direction, patience and positive direction of my advisor Mohan Kumar. As a non-traditional PHD student, to say that finishing this work has been a challenge is an understatement. On several occasions, circumstances almost caused me to quit, but his words and practical suggestions kept me moving forward. Thanks to all of my committee members for taking the time to serve on my committee and provide valuable feedback: Vassallis Athitsos, Hao Cheand Yonghe Liu. Finally, I would like thank the CSE department and the National Science Foundation for providing me financial support. The work presented in this paper was partially supported under US National Science Foundation Grant ECCS-0824120.

-with sincerest gratitude to all of you

July 13, 2011

ABSTRACT

ADAPTIVE HIGH LEVEL CONTEXT REASONING

IN PERVASIVE ENVIRONMENTS

Bridget B. Beamon, PhD

The University of Texas at Arlington, 2011

Supervising Professor: Mohan Kumar

It is hard to believe that the Internet is now in its adolescent stage. This information age is replete with communication capable, intelligent, sensor equipped devices. Social networks, web services, and global information repositories make a wealth of information available instantly. There exist endless possibilities for creating useable knowledge. Much of what is considered useable knowledge is not directly observable from low level sensory devices. Abstract situations, relationships and activities must be inferred using a variety of techniques that fuse information from multivariate data sources. We refer to this useable knowledge as *high level context*. Social, physiological, environmental, computational, activity, location and situation are but a few

categories of high level context used today. In a general sense, context is any domain specific knowledge relevant to decision making. Low level contexts can be inferred after minimal manipulation and preprocessing of sensor data. High level context is intrinsically more complex. High level context involves many levels of data fusion for inferring high level concepts. The increased dimensionality of representing and reasoning on relationships among contextual components, factoring uncertainty and ignorance, makes it difficult to effectively reason.

A research problem in the area of context-aware computing is adaptive and effective high-level context reasoning. Effectiveness refers to the suitability of reasoning methodology for efficiently reasoning and representing the heterogeneous characteristics of context. Adaptive reasoning aides in maintaining context content and quality in the face of dynamic resource availability, degrading reasoning performance and evolving requirements. Context architects are at times challenged; constrained by the limited reasoning provided in the available platforms. Incorporating a generalized hierarchical hybrid reasoning engine, offering variety and optimization for reasoning across heterogeneous complex contexts would provide an effective alternative. Such architecture integrates a variety of configurable reasoning techniques, supporting the modularity of complex high level context. Ultimately, it promotes context reasoning framework reuse, knowledge sharing, and improved context aware application performance.

This research proposes novel enabling solutions for adaptive and effective reasoning in pervasive environments. The focus is on middleware solutions for deriving and sustaining high level context, with support for reasoning adaptation and quality maintenance in dynamic pervasive environments. These solutions provided can be used for initiating context inference applications or extending existing architectures for greater reusability. Reuse leads to rapid and innovative context aware application development, a necessary evolution for achieving the vision of ubiquitous computing and beyond.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

Context is a general term for *any* information in an application domain that is necessary for decision making. It is the *role* of the information that distinguishes it as context. In many cases, high level context is a by-product of multiple stages of reasoning and data transformation algorithms. Inferred high level context is the focus of this work. The complexity and heterogeneity of context is mirrored by consuming applications making context-aware decisions. This research was initiated with the purpose of deriving a thorough definition of *context* along with justifications and methods for context aware adaptation in pervasive environments. Context aware computing includes some elements of several traditional computer science areas of research including: data modeling and representation, data management, information fusion, middleware architecture, artificial intelligence, machine learning, and sensor networking. Research articles on context awareness have appeared in seemingly unrelated conference proceedings. However, this dissertation focuses on middleware solutions for deriving high level context, sustaining reasoning and maintaining quality in dynamic pervasive environments.

1.1 Pervasive Computing Evolution

The field of pervasive computing research was ignited by the vision of ubiquitous computing described in Mark Weiser's seminal article, "The Computer for the 21st Century" in 1991 [76]. He described a future where invisible computing components ubiquitously operate in almost every domain of human living; quietly, seamlessly and intelligently improving our quality of existence. Prototypes (proof concepts) involving context awareness proliferated in the early years of 2000-2005 [6],[12],[18],[20],[23],[24],[29],[35],[38],[44],[65],[79]. Context Toolkit is one of the first works to holistically capture the needs of context aware applications [28], [29]. In the early

research, there was a great need for models and representations capable of sufficiently capturing heterogeneous context characteristics and complex domain relationships. Numerous efforts have been made in this regard. Notably, these include works based on Object Role Modeling (ORM) and Semantic Web Ontologies [15],[19],[21],[38],[43],[63]. As a result, today we have sufficient models and tools to represent complex contextual concepts in a way that affords semantic understanding, supporting information sharing.

Early context reasoning and representations were largely monolithic. There was a tight coupling with the targeted application. So we observe that in the not so distant past, context middleware solutions offered little reuse in data modeling or framework functionality. Additionally, effective context reasoning was subjugated below other more traditional middleware concerns (i.e. data modeling, discovery and communication, event notification, knowledge management). Often, where a generalized context reasoning framework existed, acceptable performance did not scale beyond small contextual proof of concepts used to validate those architectures.

Today's context middleware frameworks are more pluggable, agile and scalable. There is a trend toward generalized and integrated reasoning methods. No doubt, this trend is in part due to the increasing ubiquity of multi-sensory smart devices. The information age is replete with communication capable, intelligent, sensor equipped devices. Social networks, web services, and global information repositories make a wealth of information available instantly. There exist endless possibilities for creating useable knowledge.

Much of what is considered useable knowledge is not directly observable from low level sensory devices. Abstract situations, relationships and activities must be inferred using a variety of techniques. Truly usable knowledge can only be derived from incoming heterogeneous data from disparate sources using an integration of reasoning methods. This knowledge is *high level context*.

Varied high level context reasoning approaches are used across a variety of applications in the pervasive computing domain including: health monitoring, intrusion detection, airport security and military target tracking. The same types of context are being inferred in diverse ways

across a number of platforms. For example, human activity has been inferred using a number of statistical, ontological, and logical approaches. A survey of context uses in existing literature makes it clear that varied reasoning approaches are needed to capture the heterogeneous high level contexts we find in pervasive applications along with the varying application performance and quality requirements. Social, physiological, environmental, computational, activity, location and situation are but a few categories of context used today. Within these categories we find varying complexity. Some contexts can be inferred after minimal manipulation and preprocessing of sensor data. High level context involves many levels of data fusion for inferring high level concepts.

Whether the ubiquitous vision was a virtual inevitability or pervasive computing community efforts created a self-fulfilling prophecy, we are now in the age of context aware computing. But, have we really realized the full vision of ubiquitous computing? Is there more needed?

1.1.1 *More Context Research Is Needed*

Inferring high level context adds complexity due to the increased dimensionality of relationships among contextual components. Also, when context spans categories, involves uncertainty and ignorance, it can be more difficult to effectively reason. To be sufficiently useful, context middleware must support diverse context and inference techniques. Though there is a progressive trend towards improving the scalability and extensibility of middleware, there exists no general purpose context aware reasoning framework that supports knowledge reuse. Stagnation of context middleware reuse results from the lack of *effective, adaptive, generalized* context reasoning and representation. Developers are challenged to find a comprehensive context framework solution.

Effectiveness refers to the suitability of reasoning methodology for efficiently reasoning and representing the heterogeneous characteristics of context. Adaptive reasoning aides maintaining context content in the face of dynamic resource availability, degrading reasoning performance and evolving requirements. Context aware adaptation was an early goal in context

aware computing and has been demonstrated by a number of works. In the past, context aware adaptation has been employed in i) adapting application behavior; ii) privacy preservation; and iii) conservation of resources (i.e. power, communication). We have not seen many works adapting interchangeable sources of context to support context quality preservation. Such adaptation is critical for reasoning in dynamic environments. Utilizing generalized reusable components for application development is another fundamental barrier to framework reuse. It is important to break the often applied monolithic relationship between reasoning and applications. Decoupling this relationship facilitates reasoning reuse, leading to rapid context aware application development.

The importance of information quality becomes more urgent as information quantity increases. Integrity of knowledge directly affects its value and usefulness. Imagine the deleterious effects of actions taken by a national figurehead based on quality poor information. The consuming public would be at the very least disappointed, but the most probable pervasive sentiment would be outrage. The same issue of quality exists with consumer commodities like wine, cheese, shoes and clothing. Inferior products often visually appear to offer the same value as those of much higher quality and constitution. We can only ascertain the true integrity by revealing the source and process of construction. In a computing sense, we find a parallel need for true information integrity perception. Deciphering integrity can be difficult without the appropriate middleware quality measures in place. So, another area in need of more extensive research is the derivation of quality of context. In this discussion, Quality of Context (QoC) can be defined as a collection of measures (indicators) reflecting the integrity and discriminative characteristics of information that is used as context. There can be many factors affecting context reasoning suitability. QoC is one means by which a context middleware accesses the suitability of context and its associated reasoning process for an application. Establishing context quality is not as simple as selecting the device with the best accuracy or other quality indicator. Increasingly, context-aware applications are interested in information that must be combined from multiple sources, using heterogeneous transformation and reasoning processes. Establishing

true context quality in such environments requires an integrated approach to acquiring heterogeneous quality measures and propagating them through reasoning and transformational processes to produce a composite high level context quality. Accurately reflecting the context construction process in the composite quality is a step towards improving information integrity or quality of context.

1.2 Dissertation Contributions

The focus on middleware solutions for deriving high level context, with support for maintaining quality in dynamic pervasive environments is a result of evaluating context needs in many existing context-aware applications and middleware frameworks. Specifically, in existing context modeling works, context is often obscured with concerns unrelated to inferencing (see section 2.5 for more details on related modeling works). We offer a clear identification of context as a by-product of reasoning; independent of other domain ontological elements. There is also the challenge of expressing heterogeneous context sufficiently. The HyCoRE model supports heterogeneous/complex context representations, quality of context and context provenance. There are numerous middleware frameworks for context reasoning (see section 2.2). However, none of them support all of the following features collectively: i) heterogeneous, adaptable context reasoning with extensibility; ii) integrated high level context quality measurement; and iii) consumer quality maintenance. HyCoRE provides these capabilities and more. Our architecture design and data-models could be realized in many ways. This work includes a prototypical java-based implementation with application demonstrations. We feel that the solutions provided here-in can be used independently or as an extension to existing middleware architectures. Contributions are summarized as follows:

1. *Context Modeling*
    a. HyCoRE context data models are a refreshingly clear and generalized approach to modeling context, quality and cost.
2. *Adaptive Context Reasoning*

a. Quality based middleware performance measures

b. Quality integration and validation scheme

c. Adaptive reasoning scheme called *context flows*

3. *Context Reasoning Middleware Architecture Implementation*

a. HyCoRE architecture design, prototypical implementation and application demonstrations for evaluating HyCoRE

Additionally, the reasoning suitability research in this dissertation may serve as a guide for reasoning planning in future context aware application design. The next section discusses dissertation contribution in greater detail. Each of the contributions mentioned offer unique contributions as discussed in the following sections.

1.2.1 *Generalized Hierarchical Hybrid Reasoning Engine: HyCoRE*

Existing context frameworks and toolkits offer limited reasoner re-usability and lack support for semantically decipherable data models necessary for context sharing. Also, we find that complex context reasoning is encumbered by immature functionality as well as limited vertical context applicability. Examples where these issues arise include: i) frameworks supporting only a single category of context(i.e. activity); ii) frameworks offering low level information fusion algorithms but lacking mechanisms for integrating these reasoning approaches; iii) frameworks lacking sufficient and semantically decipherable data models and iii) frameworks supporting knowledge sharing but only offering non-scalable and computationally inefficient reasoning techniques.

We have designed a generalized hierarchical hybrid reasoning engine (HyCoRE); a middleware for generalized context reasoning in pervasive environments. HyCoRE focuses on issues most neglected by existing frameworks and can be extended to use a number of reasoning techniques to infer diverse contexts from heterogeneous sources. The flexible design of HyCoRE reasoning components support quality aware reasoning adaptation. Context is the payload on context flows and the element of exchange between HyCoRE and its consumers and providers.

The context that a particular HyCoRE instance reasons about and publishes is driven by administrative configuration and application extensions of the HyCoRE data model. So, HyCoRE is not intrinsically limited to a particular type of context, but may be limited by available context providers. Multiple data models of context may exist in a single instance of HyCoRE.

1.2.1.1 Context Reasoning Application Demonstrations

HyCoRE is general purpose context reasoning engine, supporting a variety of applications. It is difficult to find a single application that requires everything HyCoRE supports. For this reason, we demonstrate two applications of HyCoRE reasoning to highlight the various features of its architecture. Refer to Chapter 7 for details on evaluation of HyCoRE in specific applications.

1.2.2 *HyCoRE Context Data Models*

There are many correct approaches to the modeling *context* [15],[19],[21],[43],[63]. What is most important with any approach is to sufficiently capture the targeted context characteristics, support efficient query, retrieval and maintenance. Characteristics include: attribute heterogeneity, dynamism, availability, temporality, constitution, source derivation, credibility, and uncertainty. A model that is coupled to a specific type of application may have inherent performance improvements over a general purpose model as shown here. We use this general context model approach since context is the basis of information exchange between HyCoRE internal and external high level reasoning components. So, we must capture heterogeneous low level data as well as complex inferred context. We have chosen to model concepts using UML to avoid any implementation specific association. Our context data model is distinct from other works in the following aspects:

- *Multi-Centricity*- Centricity is the target domain to which context information applies. Often, the centricity reflected in the data model and reasoning is singular and tightly coupled with a specific application. Our versatile model is distinct in supporting many centricities of context, including user, device and location centric contexts; thereby supporting varied application types.

- *Multi-Dimensional Quality Representation*- In the HyCoRE architecture, every component that has affected context derivation is associated with a quality model. Sensors, reasoners, general context providers, transformation functions all have quality indicators that affect resulting context. The quality model includes both declared and observed quality indicators appropriate for the type of component.
- *Machine Knowledge Representations*- Context elements have an *explicitl*y declared meta-data model which affords semantic interpretation. This model is structured with both physical and semantic descriptions which implicitly identify relevant context providers. Context providers using different internal knowledge representations may infer the same class of context resulting in the same external meta-data modeled values. Context meta-data with physical and semantic representation models enable generalization and reuse.
- *Context Provenance*- Context provenance identifies the sources, process of derivation and change history of context data values. The HyCoRE data model has attributes that reveal limited provenance. Refer to Section 3.3.5 for more details on provenance related attributes.

We use other data models to support quality maintenance and adaptation.

- *ContextIOSpecification*- In HyCoRE, the consumer service contract is an agreement between HyCoRE and a consumer application regarding context and quality. We allow the consuming application to specify context desired and quality required. It is only the application which can provide information that distinguishes what is relevant and valuable. To accomplish effective reasoning, consumer request for context is matched on weighted context attributes, categories, locations and targets irrespective of the knowledge models used for derivation.

  Context providers also publish their capability to share context using a service contract. It is this agreement along with periodic quality verification that gives HyCoRE knowledge of type and quality of context provided.
- *Context Flows*- HyCoRE reasoning is accomplished through the execution of instantiated reasoning plans. A Context Flow is a specific instance of a context reasoning plan. A

reasoning plan is a directed graph of components. The messages that travel along the edges contain context. Nodes are the work processes for context reasoning. Nodes are an abstraction of context providers and provide an I/O specification, describing input context requirements and context inferred. Edges imply a dependency of a destination node on contextual outputs of source node. In dynamic environments, HyCoRE is capable of adapting reasoning; replacing nodes with others that have comparable meta-data descriptions.

1.2.3 *Adaptive Context Reasoning*

The HyCoRE context reasoning framework adapts to sustain high level context inference in the face of dynamic device quality and availability. A derived composite measure of high level context quality is used as a basis for adaptation. Reasoning plans are flexible in that components may be replaced with current or future context providers with match generalized information description called context meta-data. Context consumers specify context and quality requirement using context meta-data. The middleware operation measures reflect success inferring high level context while meeting consumer requirements.

1.2.3.1 Quality Definitions and Measures

Herein, the term Quality of context middleware service (QoCS) is distinct from Quality of context (QoC) in that the former measures the performance and informational integrity of the system rather than the context data itself. A wealth of quality factors representing QoC have been studied, modeled and measured including: ‘precision, probability of correctness, trust-worthiness, accuracy, completeness, representation consistency, and access security, sensitivity, freshness, and temporal-spatial resolution’ [9],[13],[15],[48]. This work presents additional QoC and QoCS measures that reflect the success of the context middleware in meeting application requirements while minimizing system cost.

1.2.3.2 Quality Integration and Verification

High level context is derived from a combination of sources including: individual sensory devices, sensor networks, data repositories, and reasoning and transformation algorithms. When consuming high level context derived from heterogeneous sources, quality aggregation and propagation ensures that middleware & applications maintain a more accurate perception of the composite information integrity. Thus they are enabled to discriminate effectively. Behavior based on quality information improves application correctness and ultimately, value. The term quality *aggregation* refers to combining quality indicators from these sources to form a composite measure of data integrity. The term quality p*ropagation* refers to aggregation from raw data acquisition though all stages of reasoning. Both are used to produce composite quality indicators. These composite quality indicators provide a more realistic measure of high level context data integrity and serve as a basis for middleware context reasoning adaptation. *Verification* involves using feedback in converging reported provider quality to actual quality.

Since, context middleware separates applications from the concerns of quality enhanced context-sensing data and reasoning, it also bears responsibility for maintaining expected information quality. To accomplish this, the context middleware must aggregate quality factors as it senses raw data from heterogeneous sources and reasons to infer new knowledge. Additionally, middleware must periodically monitor its quality performance and adapt to meet requirements.

Several challenges exist with respect to integrating quality. Heterogeneous contexts are combined to form complex high level context inference. The challenge is to reflect the relative significance of each contribution to the resulting inferred context quality indicators. In dynamic environments, sources of context may be unavailable, stale or too costly to infer at an instant in time. The challenge is to deal with missing quality indicators. When aggregating homogeneous context, we must accurately represent the additive/diminutive value of additional evidences. Also, accurately reflecting the reasoning transformation process in resulting inferred context quality indicators can also be problematic. In this work we represent aggregation and propagation

functions as middleware policy dependent strategies. So to compute the integrated quality model, the problem we need to solve is finding useful functions/strategies for aggregation and propagation. We propose concrete strategies that HyCoRE uses to solve these. Our pluggable architecture facilitates dynamic configuration of these strategies.

We present results showing the effectiveness of our quality measures for middleware adaptation.

1.2.3.3 Adaptive Context Reasoning Based on Quality of Context

HyCoRE abstracts applications, sensors, reasoning and services as generalized context sources or providers. These are dynamic and are able to register in a uniform way. Context reasoning is accomplished by composing a hierarchical plan of context sources. These are called context flows. A context flow defines the low level and intermediate contexts as well as transformation and reasoning processes needed to produce high level contexts. Application quality preferences are considered in choosing context sources. HyCoRE supports context sustainability, which refers to the system's ability to continue to infer despite dynamic provider availability as well as maintaining the required quality of context. Reasoning adaptation helps the system run longer, sustaining context and quality over time. The context reasoning framework of HyCoRE interjects into the composition of the context and identifies the best adaptation for maintaining high level inferencing. As an example, to repair a reasoning plan, one or more participating providers might be replaced to mitigate flux. This best effort strategy for sustaining context is adaptive to underlying provider and resource limiting requirements. A second goal is resource efficiency. Every stage and component involved with high level context inference incurs cost. Devices use energy while sensing, platforms require memory and CPU cycles for processing context, and communication bandwidth is required for communicating with distributed reasoning components. Cost associated with inferring context increase with reasoning complexity. Reusing context inference for the benefit of multiple consumers saves on costs associated with context processing, communication and sensor actuation. In HyCoRE, only the minimal set of sensors is actively used to meet consumer or system requirements.

In summary, the sources used in context flows are adapted to: i) optimize quality; ii) reduce cost and iii) mask mobility or other environment conditions affecting context availability.

1.3 Organization of the Dissertation

This dissertation is organized into seven chapters. Chapter one provides an introduction with motivations for this work.. Chapter two presents background and work related to HyCoRE. Chapter three details the design and data model of the reasoning engine presented herein. Chapter four discusses quality definitions and quantification used as a basis for adaptation. Chapter five provides details on the reasoning adaptation process. Chapter six presents an evalutation of HyCoRE using a prototypical Java implementation. Chapter seven summarizes and concludes this dissertation.

CHAPTER 2

BACKGROUND

This chapter presents an overview of existing literature on context modeling, middleware architecture and reasoning techniques. Research contributions by contemporary researchers in the broad area of 'context-aware pervasive computing', are summarized and compared with the contribution of this dissertation.

2.1 Context

In this document, context refers to *any* information in the application domain that is necessary for decision making at a specific point in time. It is the *role* of that information that distinguishes it as context. Context is dynamic in structure and content; often exhibits multiplicity; and is distributed throughout the components of an application. In most cases, high level context is a by-product of multiple stages of reasoning and data transformation algorithms. We make this clarification, since context has been defined in many ways:

Schilit et al. observe, "Context encompasses more than just the user's location, because other things of interest are also mobile and changing. Context includes lighting, noise level, network connectivity, communication costs, communication bandwidth, and even the social situation; e.g., whether you are with your manager or with a co-worker." [66]

Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves [28]

Dourish argues that context relevance depends on setting, activity and players involved. The scope of contextual features is defined dynamically as context arises from activity. [30]

As can be seen from the definitions given, context is broad. No single definition will do. It depends on the application at hand. Thus, we refer to context in a broad sense in relation to reasoning.

Complex context is inherently modular and hierarchical. Some context can be used with little preprocessing of sensor data (i.e. low level context). Other contexts require one or two levels of reasoning and/or manipulation with other contexts (i.e. intermediate context). Complex high level context (See Figure 1) may involve many more levels of data fusion due to the increased dimensionality of relationships among contextual components. Often, context is dynamic in structure and content, exhibits multiplicity, and is distributed throughout the components of an application. Completely cataloging all types of context is difficult, since information availability and application capabilities are increasing. Table I summarizes some contexts and related sources as used across a wide variety of pervasive applications.

We use this table to highlight the effect of context category on reasoner suitability. The same inference techniques are often applied to derive contexts of the same category due to their sufficiency in capturing categorical characteristics. Performance affecting parameters may vary with implementations. There are instances where a single approach, such as rule based reasoning, may be sufficient for capturing all contexts of a domain. However, more than likely, this one-size-fits-all approach is ill suited for other reasons. Heterogeneous context source characteristics, system goals and limitations also affect the choice of context framework with associated modeling formalism and reasoners. Consider the following examples of system goals, limitations and context characteristics below:

- System goals: Inference accuracy, inference speed, data maintainability, scalability, extensible data models and reasoning, resource (CPU, RAM, energy, and bandwidth) limitations, and platform.
- Context & Source Characteristics: Size, Dynamism, Availability (varies with source and mobility concerns), Temporality/ Durability (TTL, Expiration), Constitution/Complexity, Derivation/Source (derived, sensed, profiled), Independence /Interdependence (affected by history or other contexts), Heterogeneity, Precision, Credibility, and Uncertainty.
-

Table 1 Context Categories

Context	Applicable Reasoning Approach
Social Context (Who is nearby? What's their availability? How can I reach them?)	Logic, Rules
Availability (Busy, Free)	Rules, Decision Tree
Proximity(close, near, far)	Fuzzy Logic
Activity(Driving, Exercising, Sitting, Walking, Running, Sleeping, Eating, Talking)	HMM, Decision Tree, Bayesian Network, DSET, Discriminant Analysis
Mobility(in moving vehicle) High Level Identity(family member, coworker, neighbor, friend, unidentified)	HMM, Dynamic Bayesian Network App specific Social Ontology, Rules
High Level Location(work, school, home, store, library, post office)	Bayesian network, Decision Tree, Logic, Rules
Low Level Identity (Tom, John, MID)	App specific DB Lookup
Low Level Location (GPS coordinates)	No reasoning/Direct from device

Table 2 Approaches to Reasoning

Context Category	Example(s)	Level	Example Derviation Source(s)
Social	"At party with Amy" "With Marty & Ruby" Meeting at Work"	H	Reasoning on location and other combined contexts
Physiological Biometric	ECG, Blood Glucose, Blood Oxygen Saturation, Pulse Symbolic Health (i.e. *well, sick*)	L,I,H	Direct from Bio Sensors (*heart, blood pressure, temperature, oximeter ..*) readings; Rules & reasoning on combined context
Environmental	temperature, humidity, hot, cold, raining, snowing, light brightness, noise level	L,I,H	Direct Sensor Reading (*Thermostat, decimeter*); Rules & reasoning on combined context
Resource	printers, display devices, cameras, recording devices	I	Device registry listings
Computational	Available cpu, memory, disk drive space, power	L	Direct device statistics
Temporal	Time, day, year, season	L,I	Direct from Time sources (*Calendar, Clock, Schedule*),; Rules & reasoning on combined context
Activity	walking, running, riding, sleeping, meeting, eating, speaking, listening to music	I,II	Rules and other forms of reasoning on combined context{ (*audio, accelerometer values, heart rate , vibration, photos*)
Identity	Name (*John, Mary*); Symbolic(*friend, foe*); Relational (*mother, family member*); Type (*truck, car*)	L,I,II	RFID Tag/Reader; User Profiles; Biometric Recognition Techniques; Rules & reasoning on combined context
Location	GPS (*21° 16' 674S 27° 30' 318E*); Postal (*123 Nedderman, Arlington, Tx 67656*); Symbolic (*Home, Work, at ATM, Train Station, Grocery Store*)	L,I,H	Direct from location sources (*maps, GPS, user feedback*}; Rules & reasoning on combined context

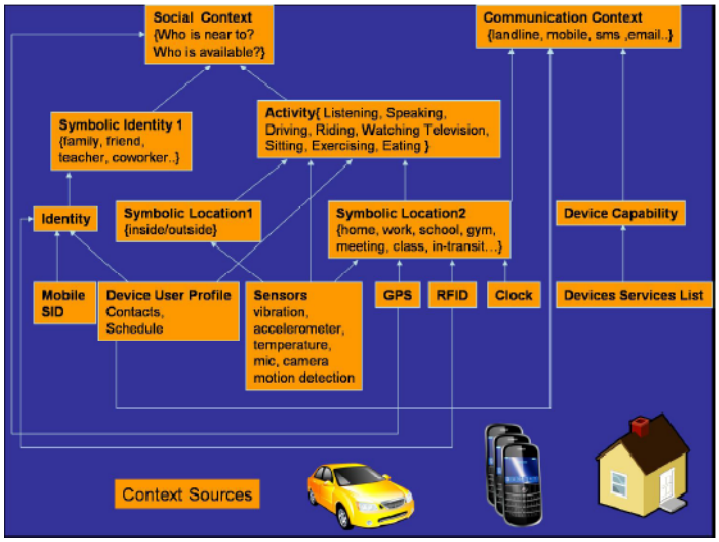

Figure 1 Hierarchical Complex Context

Cognizance of many heterogeneous contexts and hence situation-awareness is critical to composing appropriate services for deployment of collaborative applications in pervasive information environments. Sensors are the main sources of information in the environment. Context information gathered from sensors are processed, analyzed and inferred to determine high level situations and to facilitate application level services. Pervasive information environments face several challenges: i) dynamic availability due to failing or mobile sensors; ii) degradation in the quality of sensed context; and iii) selecting optimal sensors among homogeneous types. In such environments, subjected to constantly changing networks and content, the context middleware should be geared to acquire context information and execute support algorithms for context processing and reasoning and context adaptations. To this end, one trend in today's context middleware is the integration of reasoning approaches. Also, there is increasing work on defining and calculating context quality in support of application requirements. In this work, we demonstrate a way to derive composite high level context quality and show how middleware performance measures that are based on application context quality requirements may be used as the basis to context reasoning adaption enabling context sustainability and energy efficiency.

The following sections survey works related to context aware computing. As mentioned in the introduction, it is difficult to classify context aware computing, so related works that make comparative contributions to context middleware, reasoning and modeling, irrespective of their traditional computer science categorization have been selected. Many works make contributions to multiple context related categories (i.e. middleware, quality, reasoning and modeling). So, the following section titles are merely an organizational guide for reading.

2.2 Context Toolkit, Middleware and Architecture Related Work

Pervasive application framework developers have many functional concerns to address in their architecture. Functional concerns include: data modeling, sensing, component discovery and communication, event notification, knowledge management and context reasoning. Of these, reasoning is one of the most neglected concerns. In many instances, context reasoning is either insufficient or inefficient for all but a few types of contexts. Also, we seldom find frameworks supporting hybrid or integrated compositions of reasoning techniques. Yet rarer, are frameworks supporting reasoner optimizations that can be tailored to best suit targeted context characteristics or application goals. Several works make reference to hybrid context [1], [2], [18], [27], [38], [45], [53], [63]. Most of these are concerned with hybrid modeling and/or limited hybrid reasoning that combines the efficiency of rules with the expressiveness of ontologies. Though our approach is distinct, the concept is similar to the integration of shallow context representation and reasoning with ontological approaches, as detailed in the multilayer framework outlined by Bettini et al [7]. HyCoRE offers more reasoning options and greater flexibility in the way reasoning is applied. Similarly, to the notion of data flows in the Context Recognition Network Toolbox [3], HyCoRE context (i.e. inference) flows may be constructed by integrating parameterizable reasoning components in a hierarchical fashion. Both tiers of CoBrA's [18] centralized reasoning can be accomplished in HyCoRE. However, HyCoRE reasoning is distributed and is similar to the implementation of reasoning as a value added service in the federation layer of NEXUS [59]. NEXUS, UIF [11] and other frameworks supporting interoperability between context aware

systems provide a needed means for service, component and context sharing. HyCoRE also supports context sharing by its distributed design and semantically decipherable data model. However, the most important contributions of HyCoRE to a single context aware system are: i) optimizable, interchangeable, reusable reasoning; and ii) hierarchical integration of reasoning for complex context inference. These features support the natural modularity of complex context and improve reasoning usability and applicability across context types. Existing applications and frameworks may use HyCoRE to enhance core reasoning capabilities. Many other hybrid architectures & applications [22],[24], [31], [33], [34], [38], [44], [47], [52], [72], [80] are discussed in existing pervasive context literature. These demonstrate a variety of reasoning techniques, including: Hidden Markov Models (HMMs), Shannon Entropy, Artificial Neural Networks (ANN), Decision Trees, KNearest Neighbor, Naïve Bayesian Classification, Dynamic Bayesian Networks (DBNs), Dempster Shaefer Evidence Theory (DSET), rules, and custom/proprietary algorithms. Few of these architectures focus on modularizing reasoning, tuning performance or generalizing to promote reuse. Further, these are limited by the type of contexts inferred, the types of reasoning applied, openness and extensibility of the reasoning architecture.

Following is an additional survey of related context frameworks.

2.2.1 *"A Scalable and Energy-Efficient Context Monitoring Framework for Mobile Personal Sensor Networks"*

SeeMon [41] is a context middleware framework that offers scalability and energy efficiency when continuously monitoring numerous sensors in personal area networks (PANS/BANS). In a manner similar to HyCoRE, reasoning is accomplished through a pipeline of processing. This processing involves sensor actuation, feature extraction and potentially several stages of computationally intensive reasoning algorithms. SeeMon is able to save on processing costs by circumventing parts of processing when context features do not change. A process whereby high level context queries are translated into low level sensor feature queries along with bidirectional sensor communication is used to identify feature changes before reasoning algorithms are initiated. Arriving context queries are sorted into feature queries that are used to determine if

reasoning is necessary. As sensor updates arrive, only those necessary to serve active queries are selected for reasoning. Their approach for essential sensor set selection and context query handling provides 4 times better throughput and reduces wireless data transmission 50 to 90 percent when compared with the traditional unidirectional context reasoning approach. SeeMon is a novel work and shares similar concerns about efficiency. However, the issues of context quality and sustainability are not addressed in SeeMon. HyCoRE supports sustaining context in dynamic environments along with efficient processing of context. HyCoRE only actuates the minimal set of sensors necessary to support current consumer queries, but additionally offers reuse of that same sensor set by multiple applications.

2.2.2 *"Rapid Prototyping of Activity Recognition Applications"*

A context recognition toolkit (CRN) for the construction of activity reasoning plans is presented in [3]. Reasoning plans are called *data flows* and are constructed as chain reusable components/tasks. Each task is an encapsulation of reasoning algorithms and or data transformations. Tasks parameters control its operation. The specific IO requirements are declared by the task for generalized reuse. Optimization may also be provided as inputs. Tasks available in the toolkit include: Average Signal Energy, FFT, Distance2Poisiton, Hexomite2D, HMMs, KNN, PCFG parser, RangeChecker, SequenceDetector, and Simple HexSensClassification. The toolbox is extensible and provides APIs for further task development. This work illustrates that heterogeneous activity recognition can be accomplished using a small set of parameterizible algorithms. HyCoRE shares the goal of rapid development, but for heterogeneous context. We do not discount the significance of this work, but we must point out its limited applicability to the user-centric activity context. It is a solid prototyping framework for user contexts. HyCoRE uses a similar chain of reusable reasoning components (i.e. *context flows*) which may contain more than one processing chain. It is possible that the tasks of CRN can be exposed as context providers to HyCoRE. HyCoRE's physical and semantic modelsfor high

level context along with quantification and representation of integrated quality measure have no parallel in this work.

2.2.3 *"Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments"*

Middleware mediated context management for dynamic environments is espoused by the authors of the Orchestrator framework [42]. Orchestrator performs *active* context mediation. Applications do not make decisions regarding resource allocation. Rather, the middleware proactively keeps context plans mapped to the best set of dynamic resources. Their solution advocates a separation of logical context processing needs from physical resource allocation. This separation is realized as logical and physical context processing plans. Logical plans can be defined and added to the system administratively. Since sensors and associated resources change continuously, the translation of logical plan to physical plan occurs at run time. The efficacy of a physical plan is determined by several factors, including: required resource availability and cost of executing plan. This work illustrates how a single high level context can be derived in variety of ways. For example many logical plans could be declared for deriving *activity* context from accelerometer readings. However, several varying factors realized by different physical plans could offer different performance: i) varying the part of body on which the accelerometer is worn; ii) varying the features (*frequency vs. statistical*) extracted and transformations performed; and iii) varying inference model (*Decision tree or Naive Bayes*). The notion of both logical and physical reasoning plans is very similar to our thoughts on context templates and flows. Orchestrator plan adaptation is triggered as sensors join and leave, resources status changes or context requests change. This results in reconfiguration of processing plan set. The adaptation triggers are similar in HyCoRE. However, HyCoRE is able to adapt a single plan by removing and replacing nodes or by reconstructing a new plan, as opposed to reconfiguring the entire context flow set. Our approach is also distinct in offering application specified quality requirements along with context requests. Orchestrator does not consider quality of context as an integrated measure of integrity;

whereas HyCoRE is focused on measuring the middleware performance in adapting to meet multiple application quality requirements while minimizing system cost.

2.2.4 *"Toolkit to support intelligibility in context-aware applications"*

An extension to the "Context Toolkit"[29] which adds inference explanations to the end user is presented in [47]. This extension work addresses the challenge of making context aware applications intelligible by providing explanations of reasoning behaviors that are independent of the decision model. Namely the following explanations regarding an inference are provided: inputs, outputs, what, what if, why, why not, how to, and certainty. Context research has shown that these explanations add value to the context consumer. We observe that it is important to the context consumer to understand how an inference is made. Our approach tracks all participants of context inference and provides a serialized inference trail for provenance support. As certainty is provided as a type of explanation in this toolkit extension, HyCoRE supports certainty as a quality indicator. For probabilistic inferences certainty may be in the range 0 to 1, or may simply be 1 or 0 for rule based inferences. We also go further in computing a high level context quality vector which is a composite quality indication, reflecting the quality of all participating providers. To sustain context effectively we have developed an adaptable context reasoning framework along with measures for discriminating context providers.

2.2.5 *"An Extensible Sensor based Inference Framework for Context Aware Applications"*

This work [14] highlights the importance of breaking the often applied monolithic relationship between reasoning and applications. Decoupling these relationships facilitates reasoning reuse, leading to rapid context aware application development. This work is a realization and extension of an earlier Intel context framework design. The primary goal of this work and its demonstration is to achieve rapid context aware application development by removing the burden of sensing and reasoning as an application developer concern. To this end, Intel has created C++ based generalized context reasoning framework which performs inference based on XML DAG reasoning plans. Context providers of the systems are sensors or traditional pervasive devices

such as accelerometer, gyroscope, microphone etc. So then, the low level input is raw data with varied formats and sampling rates. Mills are generalized reasoning work components. The inputs, associated training models and algorithmic details are abstracted into a generalized API. Using generalized APIs, context application developers compose reusable DAG nodes, called *mills* to describe the high level context to be inferred. The context framework performs necessary sensing, buffering, normalization, feature extraction, transformation and reasoning algorithms. Additionally, the reasoning may be easily extended by creating new mills, which extend a common parent type. Multiple client applications are enabled to use the framework. Approaches to using the framework for three horizontal applications: gesture recognition, audio classification and physical activity reasoning are described.

Rapid context aware application development is also a primary motivation of HyCoRE. The concepts used for reasoning is very similar. Context reasoning is accomplished using DAGS, which we refer to as context flows. There are a few differences that make HyCoRE reasoning distinct. Since HyCoRE is designed with the added goal of context source tracing, participatory reasoning and adaptation. To this end, context source tracing and quality integration is performed in-line with reasoning to maintain an accurate view of context derivation and integrity. Context flows are associated with abstract reasoning plans or patterns. These abstract patterns support reasoning adaptation. In a dynamic system where multiple choices of context providers exist, a flow may be repaired by replacing a failed or sub-optimal reasoning node. Also, an alternate flow deriving the same context may be constructed using an alternate pattern. The Intel framework can be classified as a context toolkit. HyCoRE has considerations for middleware performance, and application quality requirements.

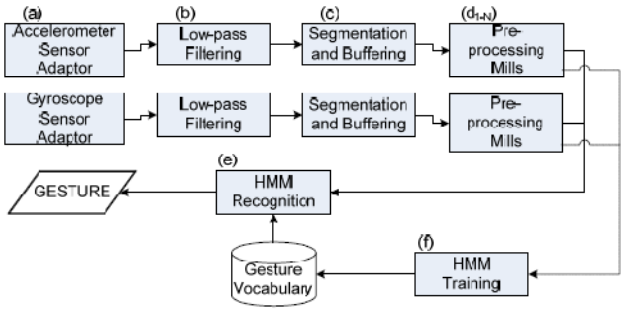

Figure 2 Intel Framework Gesture Recognition and Physical Activity Reasoning DAGs

2.2.6 *"INFERD and Entropy for Situational Awareness"*

INFERD [71] is an information fusion engine created as an aide to decision making in cyber security applications. However, this probabilistic graphical framework is flexible and can be used to model and reason across diverse applications. Airport Security, Cyber Network Attack alerting are two applications specifically mentioned. The implementation presented addresses levels 0, 1 and 2 of the US Joint Directors of Laboratories (JDL) 5-level data fusion model.

One of the examples is a simple airport security scenario. This is only used to explain the concepts of INFERD. The objective is to determine if a passenger is a threat to other passengers. Details of the example are as follows:

1. The probabilistic value of a passenger's identity being valid is some algorithmic combination of Risk Assessment, Photo ID Verification, and Biometric scans. See below:

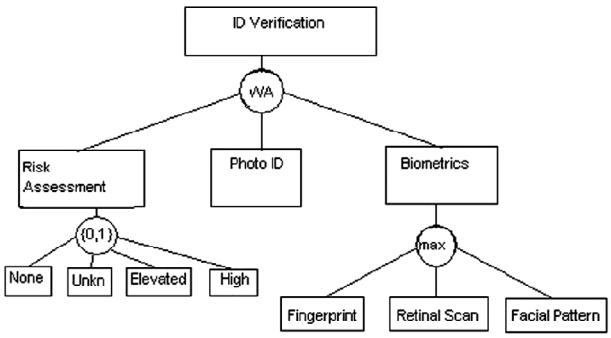

Figure 3 INFERD – Passenger Identification

- Risk is accessed by booking agents based on answers to questions
- The attendant at the airport verifies that information on ID matches that in system
- The CAPPS II (Computer Assisted Passenger Prescreening System) available at most airports is used verify biometrics.
- In all cases 0 indicates no problem and 1 indicates a security concern

2. Other interrelated context models are used: suspicious behavior, prior history, forbidden words

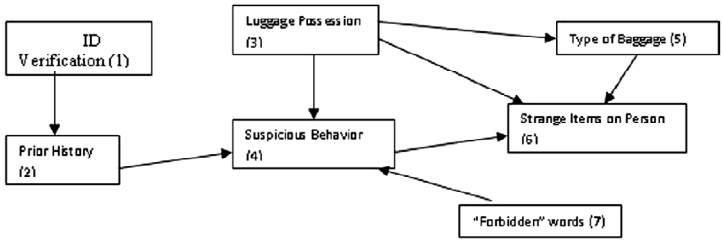

Figure 4 INFERD- Passenger Threat Template

3. All of the above mentioned contextual elements form a high level template graph that models passenger threat
4. Gibbs Boltzman and Shannon Entropy functions are demonstrated for determining template graph credibility (probability of the overall event occurrence).

5. The result from each approach was vastly different. The Gibbs Boltzmann parametric approach yielded 54% based on the given probabilities.
   The Shannon Entropy approach yielded a 26% probability.

IO is separated from fusion in this architecture. Application contexts are modeled in as in interconnected hierarchy {low level (0) to higher level situation reasoning (level 2 and above)} Multiple approaches to reasoning on the credibility of a situation are applied based the fusion level: i)At level 0, constraints and rules/thresholds are used at a low level to extract applications features from raw sensor data; ii) At level 1, various functions/rules can be used to aggregate features into composite events (*called template nodes*). Max/Min, and Yager's Generalized Ordered Weighted Average are discussed as some sample functions for an Airport Security Scenario and iii) at level 2, Shannon Entropy and Gibbs-Boltzmann equations are demonstrated as techniques for fusing probabilistic values of composite events into composition situations(*called template graphs*). Gibbs-Boltzmann parametric approach is compared with Shannon Entropy.

Accuracy and speed of inference are two characteristics that vary depending on the approach. This paper demonstrated a 54 % probability of passenger threat using Gibbs-Boltzmann, but only 26% using Shannon Entropy. In another application, the authors noted the ability to process and generate hypothesis about 86.4 million alerts in a 24 hour period. Further, the paper demonstrates that context can be viewed as a multiplicity of graphs (hierarchies of data). In a single context graph, multiple algorithms may be used to combine that information (i.e. different approaches to low and high level data fusion). There are a few additional considerations that HyCoRE addresses which could be used to improve the INFERD architecture: i) Context may begin and end any level of the JDL model. A reasoning engine must provide as little or as much as needed for the application and ii) Sensors are only one kind of contextual source. Feature extraction needs to be performed when receiving context from other sources. As we move towards distributed context awareness and reasoning, context may originate from sensors, users

and applications. Though data that is queried (pulled) is inherently streamlined for a particular aspect of reasoning, heterogeneous data that may be pushed onto a system must be filtered for applicability and value to the reasoning task at hand. Thus, feature extraction/data transformation functions must be considered for the range of contextual sources.

2.2.7 *"AI Techniques in a Context-Aware Ubiquitous Environment"*

Mobile being [26] is a generalized inferential framework concept for mobile devices. The adaptation use case is: automatically loading and unloading applications to a mobile device based on user and device context. Like a chameleon, the mobile device assumes the role mostly useful to the user. Similar to HyCoRE, it proposes AI approaches to reasoning on physical data to infer higher level abstract data. HyCoRE is a more general purpose framework that automatically adapts the middleware reasoning rather than the application.

2.2.8 *"Context-aware adaptation in an ecology of applications"*

A context middleware framework that enables structural application adaptation is presented in [59]. Application behavior is defined in a description of self-organizing components. Their approach couples application and middleware context reasoning. The HyCoRE concept of context flows offers a similar notion of service composition. However, reasoning adaptation in the middle is separated from application adaptation. The middleware of HyCoRE adapts to meet quality goals of many applications. It is not concerned with specific application behavior adaptation.

2.2.9 *"LoCa: Towards a context-aware infrastructure for e-health applications"*

LoCa [33] is a generic software infrastructure for adapting work flow based applications to user context. Functionally, its goal is to gather, process, analyze, visualize and store physiological data in an electronic health record. Workflows and visualization are adapted based on context {i.e.*procedures may change dynamically depending on age of patient, the Doctor. may be sent an SMS if biometrics reach a critical threshold*} As with most frameworks we've noted, reasoning is a secondary concern and is discussed only lightly. Using rules and logical connectors is mentioned,

but reasoning is not the focus of this work. However, the brief discussion on reasoning does make this point: '*Raw data has to be coarsened and analyzed in relation to one another*.' Automating this process through reasoning saves time, money and ultimately lives. A telemedicine scenario is presented where a doctor can remotely monitor a 65 year old heart patient. Instead of sending a nurse, to record the daily ECG data and other measurements, sensors and the LoCa framework are used to automatically interpret the raw sensor data in a particular order, comparing values with the patient's medical history.

The primary focus of LoCa is application infrastructure. Our focus is a reasoning engine. LoCa is an example of a context framework that could be improved with an reasoning engine like HyCoRE. LoCa also offers a base data model. We agree that some foundation data model should be used along with reasoning to capture context. We offer our context data models with the unique features that described herein.

2.2.10 *"Context Broker Architecture: CorBrA"*

CoBrA [18] is broker-centric architecture for supporting context awareness. An intelligent agent collects contexts from varied sources into a centralized location. OWL[1] is used to model and reason on context. OWL is based on P-SHIN, so it is essentially FOL reasoning. OWL does not support uncertainty or ignorance. OWL description logic reasoners are used for inference and consistency checking of knowledge model. The known tractability issues with reasoning using OWL apply with CoBrA. OWL reasoning does not scale well for applications with dynamic context models due to the overhead of re-classification. HyCoRE is more than a broker or central repository. It is an orchestrator of interchangeable reasoning elements and context source which dynamically adapts based on quality indications.

[1] SOUPA Ontologies are used

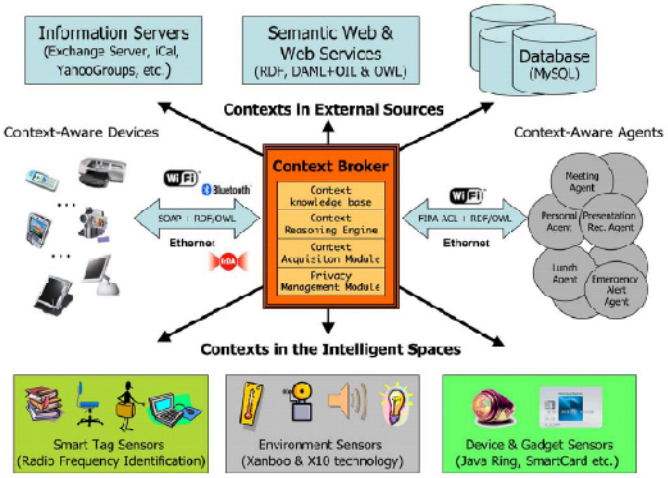

Figure 5 CoBrA Architecture

2.3 Quality of Context Related Work

There exist many examples of solutions for quality aware adaptation [9], [13], [15], [48]. The authors in [48] show how quality can be used to support adaptation that: i) alters application behavior; ii) improves middleware efficiency and iii) enforces privacy policy. In [15], the authors have observed that adaptation is expensive. They offer a quality measure (i.e. 'probability of correctness') which aids in increasing the utility of the decision to change/adapt context source. Several other examples are established on the domain model where multiple sensors collect similar information. These works adapt to minimize communication and power consumption costs associated with sensing and retrieving low level context data. The following sections review some works related to context quality.

2.3.1 *"Modeling and Measuring Quality of Context Information in Pervasive Environments"*

Our work shares many ideas found in [13]. We both recognize the importance of: i) a semantically decipherable context model ; ii) a flexible approach to context reasoning; iii) identifying the quality of *inferred* information though quality aggregation; understanding that QoC is composed of many indicators and iv) allowing applications to declare the relative importance of quality indicators. In support of context representation, this work presents a user centric data and QoC model. We recognize that there are many correct ways to model context and have chosen a representation

that is not centric to user, device, location or other context category. Rather, we present a general model sufficient to capture heterogeneous context types, support quality integration, source tracing, and the types of queries and adaptable reasoning intended with HyCoRE. The context reasoner of the architecture [13] supports description logic/semantic web rule language (DL/SWRL) reasoning. Also, there are abstract context reasoning components (CRC) serving as black box brokers of context. Abstract HyCoRE context providers are similar to CRCs. In HyCoRE a context provider is abstraction for algorithms, services or devices providing context. A context provider may support any type of reasoning as long as the resulting context and associated quality can be clearly described. Our distinction is in the way providers are integrated into a reasoning plan for deriving high level context. HyCoRE middleware understands how to construct reasoning plans in part based on the context metadata included in context provider service contracts. HyCoRE offers similar semantic definitions for quality indicators as those in [13]. The definition of informational resolution provided herein (see Chapter 4) has relevance to disclosure level, sensitiveness and resolution in [13]. Informational resolution applies to many types of context and simply reflects the amount information a context value reveals relative to some maximum. Completeness also has semantic similarity to HyCoRE's interpretation; reflecting the totality of the information used for deriving a specific context value. Missing and expired information reduces completeness. Additionally, HyCoRE offers other quality indicators and middleware performance measures that are unique. As research on context quality grow, so too does identification of useful measures. In this dissertation only a few select quality measures are highlighted. Our selection of quality measures serves to build support for a reasoning engine that adapts based on environmental context provider availability and quality along with application context quality requirements. Finally, though there are some differences in the way HyCoRE integrates quality indicators through reasoning plans, both [13] and HyCoRE employ a pluggable approach in aggregation algorithms applied to each quality measure. In [13], algorithms for quality aggregation (*i.e. pessimistic, optimistic, and average*) are system specified and may be distinct

for each quality measure. HyCoRE proposes a similar approach to quality integration (see Section 4.3).

2.3.2 *"A letter soup for the quality of information in sensor networks"*

The authors of [9] make a compelling argument for maintaining QoI in wireless sensor networks (WSNs). The approach to information sharing is similar to HyCoRE in that: i) Applications advertise needs; ii) Context providers advertise information available, and iii) applications are bound to selected providers. However in HyCoRE, the context middleware performs negotiations with providers; adapting for improved performance; shielding applications from environmental changes. The authors suggest a similar modification to the Sentire framework [12]. Novelty can be seen in the use of six common primitives (5WH: *why, when, where, what, who, how*) to capture application needs and information provider capabilities. The HyCoRE data models support describing and determining 4WH. Currently, we cannot see how the application's motivation (i.e. '*why*?') is relevant to the middleware. HyCoRE simply performs to consumer specification. A distinct feature of HyCoRE is projecting integrated quality calculation onto the context reasoning process. This is referred to as quality aggregation and propagation (see Section 4.3).

2.3.3 *"QoI-Aware Wireless Sensor Network Management for Dynamic Multi-Task Operations"*

A WSN framework supporting multiple prioritized tasks with heterogeneous QoI requirements is presented in [48]. . QoI is represented as a vector of attributes that may be extended over time. Also, each task specifies its own requirements on the values of attributes in the QoI vector. A *satisfaction index* or measure of the WSN's ability to meet task quality requirements is presented. Also, the authors derive a measure of the network's ability to accommodate new tasks, called *network capacity*. Network capacity is based on a composite satisfaction index function. A task admission control scheme is implemented using network capacity as a threshold. When attempting to admit a new task, if network capacity isn't sufficient, an attempt at quality re-negotiation of existing tasks is made. Also, as tasks complete, WSNs resources are reallocated

for optimal use. It is demonstrated by simulation that such an approach meets task requirements while reducing resource consumption when compared with traditional WSN task execution strategies. There is a parallel with our approach to quality based context reasoning. HyCoRE middleware receiving a consumer context request containing a prioritized context list with prioritized qualities indicators is similar to multiple tasks arriving at a WSN sink. Each task would correspond to a single context item requested. QoI and HyCoRE QoC vectors are practically synonymous concepts. The middleware context providers available at a time instant would correspond to the fixed WSN. The goal of HyCoRE is to distribute context reasoning functions in a way that meets consumer requirements while minimizing system cost. The *satisfaction index* is similar in meaning to our application effectiveness measure. One difference in our work is that QoC measurement involves the integrating quality from all components involved in the context reasoning process. This is referred to as aggregation and propagation. Further, HyCoRE employs a feedback mechanism as a check and balance to declared quality. Consumer requirements are mapped to system performance measures which are used as a basis for adaptable reasoning. HyCoRE method for measuring the middleware reasoning performance is distinct from [48] and supports the goals of its architecture as discussed in Section 3.1.

2.3.4 *"Middleware support for quality of context in pervasive context-aware systems"*

This work makes an argument for context-aware middleware which decouples applications from heterogeneous sensors and supports rapid application development [69]. The motivating reasons presented for Quality of Context are: i)QoC based application adaptations, ii)Middleware Efficiency, iii)User's Privacy Enforcement (*The middleware must therefore provide users with the means to limit the QoC information provided to different requesters*). They identify and define five Quality of Context QoC indicators:

- **Precision**- 'granularity with which context information describes a real world situation'.
- **Freshness**- 'the time that elapses between the determination of context information and its delivery to a requester'.

- **Spatial resolution**- 'the precision with which the physical area, to which an instance of context information is applicable, is expressed'.
- **Temporal resolution**- 'the period of time to which a single instance of context information is applicable
- **Probability of correctness**- 'the probability that an instance of context accurately represents the corresponding real world situation, as assessed by the context source, at the time it was determined'.

Other QoC from other works not included in their list are : trustworthiness, *coverage, resolution, accuracy,* repeatability, frequency and timeliness. The authors give examples and justifications for these 5 quality indicators. They believe they are the first to offer some quantification for Quality of Context. This work leaves ample room for defining many more quality indicators. Additionally, techniques for integrating QoC when inferring high level Context is needed.

2.3.5 *"An Effective Quality Measure for Prediction of Context Information"*

A technique for comparative selection of high-level context inference algorithms is presented in [73]. It defines a metric, C, by which to discriminate high-level context inference algorithms. The metric C measures the certainty for each value inferred. C is then used in a weighted error rate computation for the learning algorithm. C is also used in a formula correlating it to the probability of correctness. C is only appropriate for probabilistic classification algorithms. The correctness of learning algorithms has been measured in various works, however this work quantifies the probability of correctness. Since, pervasive devices are resource constrained and adaptation can be costly, they propose selecting a reasoning algorithm based on the greatest probability of correctness. In a similar vein, our work allows various inference algorithms to be compared using multiple quality indicators that may be observed and learned through verification. In HyCoRE, reasoning plans are selected based on ability to meet application requirements expressed as a context vector with associated quality indicators. Another observation is that this

work does not deal with the interaction of context elements, reasoning, transformation and the corresponding effect on composite Quality of Context as we do herein.

2.3.6 *"Building Principles for a Quality of Information Specification for Sensor Information"*

An application agnostic Quality of Information Specification is presented in [8]. There are many works deriving quality measures. Varied device types use specific models to describe quality. Often a different name is given in one approach to describe a semantically equivalent concept in another. The calculations used to compute the quality measures are as disparate as the implementing applications. This work is an effort at providing a common description representation for QoI measures. Common for all sensor/data source types.

2.3.7 *"Quality of context: What it is and why we need it?"*

Rationale for context aware middleware which decouples applications from heterogeneous sensors and supports rapid application development is provided in [15]. The motivating reasons presented for Quality of Context are: i) QoC based application adaptations; ii) Middleware Efficiency; and iii) User's Privacy Enforcement. The authors focus on five QoC indicators. They believe they are the first to offer some quantification for Quality of Context. This work leaves ample room for defining many more quality indicators. Additionally, techniques for integrating QoC when inferring high level Context is not discussed. HyCoRE offers additional quality measures and an approach to quality integration.

2.4 Deriving High Level Context Using Integrated Reasoning Related Work

Increasingly, context aware applications require information derived from heterogeneous sources. To this end, one trend in today's context middleware is the integration of reasoning approaches. In this work, we demonstrate a way to derive composite high level context using a hybrid approach. In this section we review a few related works and discuss their relation to our work herein.

2.4.1 *"Detection of Daily Activities and Sports with Wearable Sensor in Controlled and Uncontrolled Conditions"*

This work [31] makes comparison of supervised and unsupervised learning approaches to context reasoning. Artificial neural networks(ANN) and decision trees(DT) are used to identify the physical activities including: lying down, sitting, standing, general walking, Nordic walking, running, cycling on stationary bike, cycling on moving bike rowing, playing football. Identifying 'playing football' is an example of a complex high level context that reasons on lower level context: walking, running, standing, kicking the ball. They compare the inference accuracy of several context classification approaches: custom decision tree (*supervised data*), automatic decision tree(*unsupervised data*), artificial neural networks and a hybrid approach with both ANN and DT. Similar to HyCoRE, high level context is derived by a process of hierarchical reasoning on lower level context. However, this is an application of context where reasoning is coupled with the application. There is no framework module or data model that can be shared with HyCoRE. HyCoRE decouples the application from the reasoning engine, supporting reusability.

2.4.2 *"Context-aware activity recognition through a combination of ontological and statistical reasoning"*

[59] provides a demonstration of improving statistical activity recognition performance by considering context. A hybrid of statistical and ontological reasoning is used. A voting algorithm for resolving activity context inconsistencies is shown. The algorithm filters user activity by ensure it is consistent both statistically and ontologically for a given time window. The ontological modeled (TBox) is setup such that only limited activities are possible at described locations. If an activity inference is made that is inconsistent with the model, it is removed. The authors have a similar thought on generalized reasoning. We feel it is possible that many algorithms and associated training sets may be generalized. Consider these examples: sound can be generically classified as human voice or music; activity as sitting, standing, walking, or running; and weather as cold, wet, dry, or damp. The authors herein make the statement: 'Ideally, an out-of-box activity recognition system should be able to recognize one person's activities without the need of being trained on that person.'

2.4.3 *"MEBN: A Logic for Open-World Probabilistic Reasoning"*

Multi Entity Bayesian Networks is a logic for probabilistic reasoning [46]. It augments Bayesian Network Theory with First Order Logic Model Theory[2]. This is one of many approaches to probabilistic logics. FOL is the primary approach to reasoning in logical systems as Propositional Logic(PL) lacks the expressive power to define models with many items concisely. PL is very context dependent{ *car1 is blue, car2 is red*}. First Order Knowledge allows us to model objects and relations in a context independent way{ car(color)}. It is possible to express facts about some or all object in the universe (i.e. *Exists (car (color =blue)?,For All ((car (color =blue), Owner=female*)). FOL can determine a query to be T,F or indeterminate. It does not support reasoning under uncertainty as is found in many real world applications such as knowledge interchange. Probability is the most well understood approach to reasoning under uncertainty. It provides a coherent calculus for combining prior knowledge with observed data. Bayesian networks are an efficient probabilistic inference approach. However, the following issues have hindered Bayesian approaches to probabilistic reasoning: i) Lack of modularity; ii) Intractability of worst-case inference; iii) Difficulty in verifying unique and well-defined probability distributions and iv) Complexity of specifying local distributions{exponential to number of parents}. MEBN resolves these issues. It is modular and compositional. Like Bayesian networks uncertain hypothesis are represented as nodes (random variables) of a directed acyclic graph. The arcs represent probabilistic dependencies. Related random variables are logically separated into collections called MFrags. MFrags are partial Bayesian graphs used to derive posterior probabilities of their resident random variables (generative knowledge). MFrags also contain a general knowledge referred to "findings" that may be added based on observations. Findings are similar to T-Box assertions in ontological reasoning. Generative knowledge resembles ABox assertions. Both contribute to inference.

[2] Also commonly referred to as First Order Predicate Calculus

Traditional Bayesian networks are generally context specific and insufficiently expressive. MEBN provides a framework for generalized reasoning with Bayesian networks in the same way the FOL extends propositional logic with Universal and Existential semantics. Random variables in MFrags take arguments that refer to instances of entities in the domain application. Sets of MFrags are organized into collections referred to as MTheories. MTheories imply a joint probability distribution that can be used to answer application complex queries. Sequences of MTheories are created as new axioms(knowledge sentences) are added that do not contradict previous assertions. MEBN supports recursive MFrags. An instance of a random variable may depend directly or indirectly on other instance of the same random variable. This is similar to that offered in dynamic Bayesian networks. Additionally, MEBN logic comes equipped with a set of built-in MFrags representing logical operations, function composition, and quantification. An inference algorithm called Situation Specific Bayesian Network(SSBN) construction is provided. An SSBN is the minimal Bayesian network sufficient to compute the response to a query The SSBN can be approximated by pruning random variables and arcs that are irrelevant to the query. Also, specialized reasoners may be used for parts of the SSBN. These may include: i) Constraint satisfaction systems; ii) Deductive theorem provers; iii) Differential equation solvers; iv) Heuristic Search and optimization algorithms; v) Markov chain Monte Carlo algorithms and vi) Particle filters. Such approximation is part of *hypothesis management* which follows from *execution management* where accuracy is balanced against computational resources constraints. Interestingly, MEBN may be used to reason about which approximation to apply. The authors make the following observations which support representing knowledge a probabilistic FOL format: i) FOL is the *de facto* standard logic for formalizing both individual assertions and knowledge structures and ii) Probability theory provides a principled approach to knowledge interchange among different reasoning. The following diagrams show how a sample diagnostic task can be modeled from BN to MEBN.

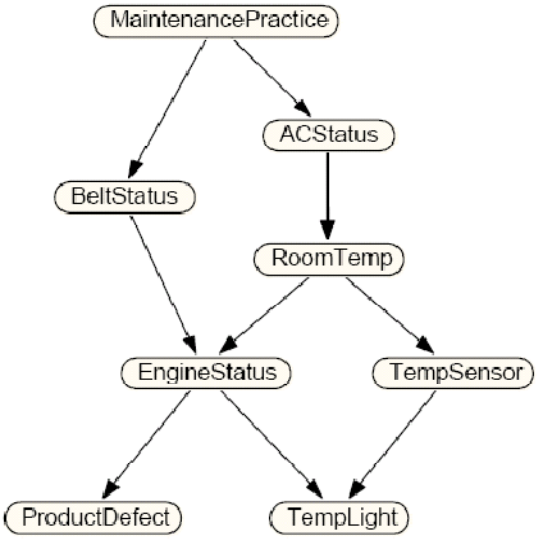

Figure 6 Bayesian Network For Equipment Diagnostic Task

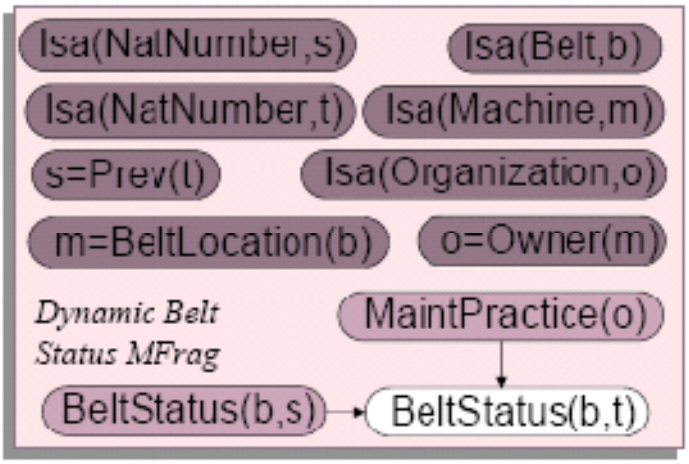

Figure 7 Recursive Mfrag

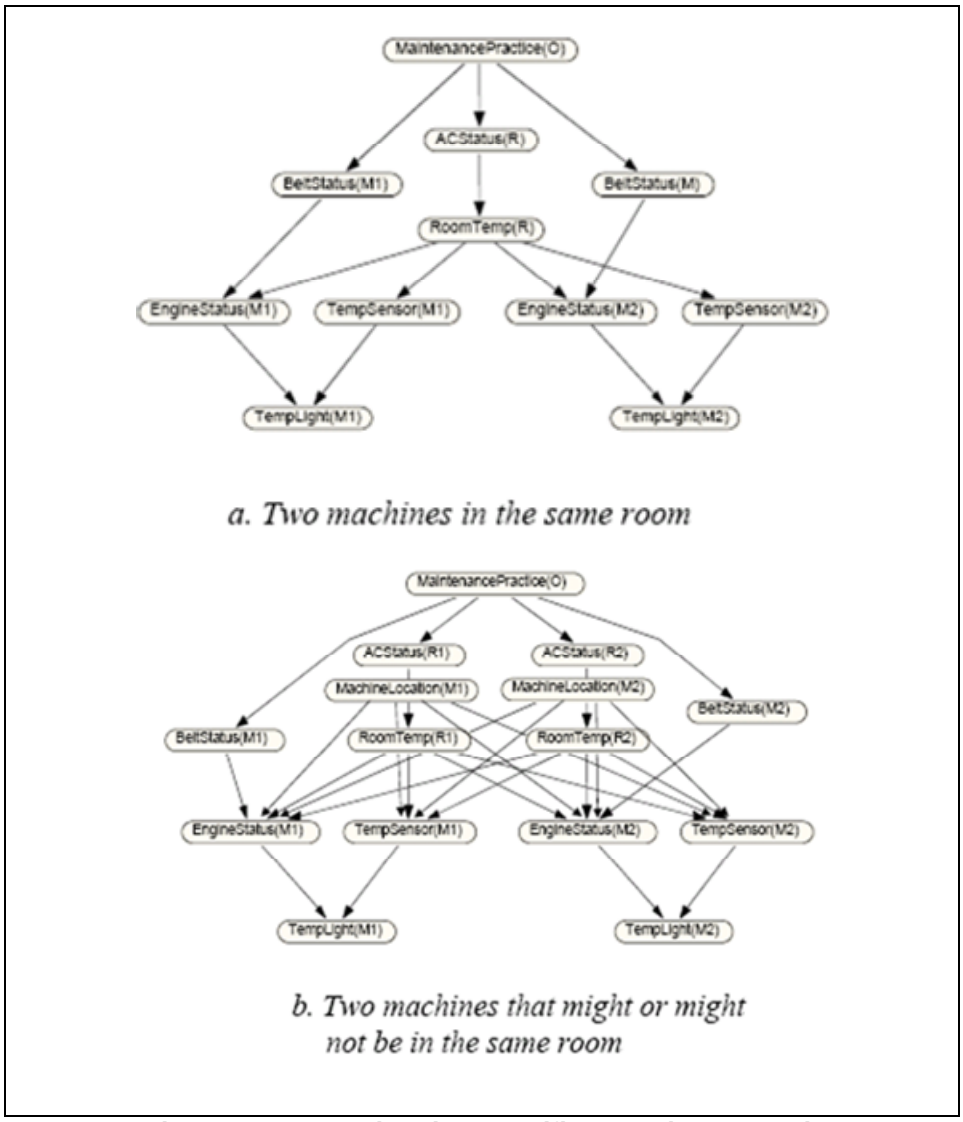

Figure 8 MEBN - Situation Specific Bayesian Network

MEBN is not a competing idea, but provides the basis for tools that may complement hybrid modeling and reasoning. The information presented underscores the need for open world reasoning under uncertainty. Approaches that do not inherently support uncertainty or those constrained to specific domain instances are of limited usefulness. The expressiveness & theoretical soundness of combining First Order Logic model semantics with probability theory was explained. The idea of execution management is also discussed. Reasoning may be adjusted to

balance accuracy against computational resource constraints. Further there may exist system polices or other justifications for selecting one algorithm over another. This execution management idea is similar to my approach of hybrid reasoning by context flow construction and adaptation.

2.4.4 *"Towards the adaptive integration of multiple context reasoners in pervasive computing environments"*

The effectiveness of integrating multiple context reasoners is demonstrated in [61]. A middleware solution for handling dynamic sensor environments is proposed. There is a demonstration of aggregation where the same context is multiply provided, by averaging confidence vectors after parallel execution of classification reasoners. The focus of HyCoRE is different. HyCoRE reflects a composite quality though many aggregation and propagation of quality indicators. Adaptable context reasoning, including the process and techniques for integrating quality, is one of HyCoRE's chief contributions.

2.5 Context Modeling and Representation Related Work

2.5.1 *"Evaluation and Analysis of a Common Model for Ubiquitous Systems Interoperability"*

One barrier to context sharing between pervasive systems is lack of a common semantic representation. Middleware frameworks tend to use proprietary data model. The authors of this work evaluate UIF/UCM used in the NEXUS framework [11]. UIF/UCM has the single purpose of providing a unifying model for context data. The UIF is a composite knowledge store, containing inputs from applications, reasoners or any context supplier that uses the Ubicomp system adaptor. The reasoning engine is based on Jena and supports SWRL rules. The context model used in HyCoRE can to support interoperability. UIF/UCM does not support quality measurements, source tracing and generalized reasoning as our model. HyCoRE is not limited rule based knowledge representations. While rules are expressive, large rule sets are cumbersome and inefficient when compared to statistical approaches.

2.5.2 *"Standard Ontology for Ubiquitous and Pervasive Architecture SOUPA"*

This work attempts to define generic OWL vocabularies that can be shared by all pervasive computing applications [21]. SOUPA also provides an extension to define additional vocabularies for supporting specific types of applications. The authors have recognized that existing pervasive systems were weak in supporting knowledge sharing and reasoning and lacked adequate mechanisms to control how information about individuals is used and shared with others. SOUPA provides historical perspective on modeling efforts. SOUPA consists of several sub ontologies:

- *Friend-Of-A-Friend ontology* (FOAF)- Allows the expression of personal information and relationships
- *DAMLTime*- Designed for expressing temporal concepts
- *OpenCyc Spatial Ontology*- Define a comprehensive set of vocabularies for symbolic representation of space
- *Regional Connection Calculus (RCC*)- Consists of vocabularies for expressing spatial relations for qualitative spatial reasoning;Describing and reasoning about location
- *COBRA-ONT*- Focuses on modeling contexts in smart meeting rooms
- *MoGATU BDI ontology*- Focuses on modeling the belief, desire, and intention of human users and software agents
- *Rei policy ontology*- Defines a set of concepts for specifying and reasoning about security access control rule

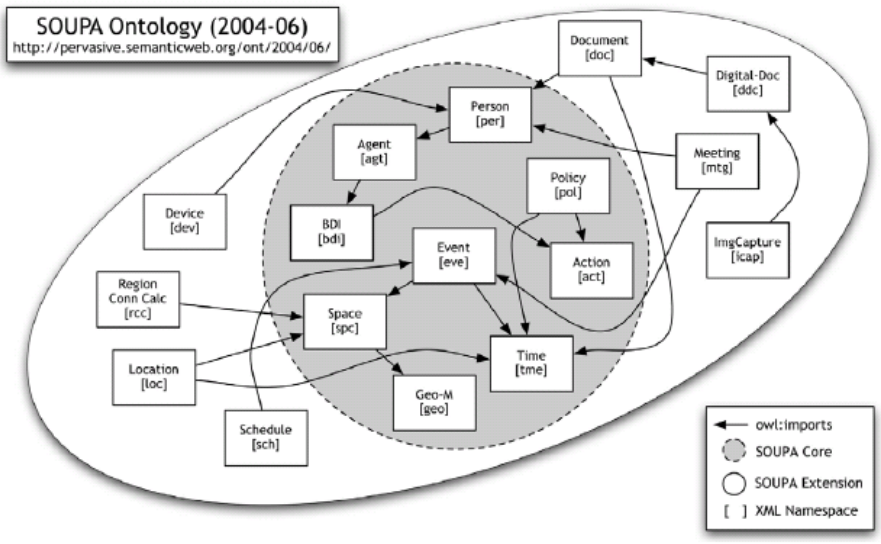

Figure 9 SOUPA

2.6 Survey of Knowledge Representations used for Context Modeling and Reasoning

The most widely used techniques for inferencing high level context are rule-based and decision tree algorithms, naïve Bayesian and hidden Markov models [47]. Applying these techniques to low level context data often requires a combination of inputs and layers of transformation. Choosing appropriate reasoning methodologies for context is critical to application performance as well as utilization of resources. Context architects must address the many suitability factors before choosing a methodology for inference. In the following, a sampling of considerations for choosing reasoning methodologies is discussed. The considerations are mainly based on: nature of context data & their sources; inferencing mechanism characteristics; and resource considerations. Description Logics (DL) reasoning on ontologies is a natural way to guarantee such requirements as rules, constraints or consistency levied across the entire context/knowledge set. However, the expected size of the context data at maturity and the sampling frequency must be considered since DL reasoning does not scale well in terms of meeting interference timelines for large knowledge sets [7],[43],[53],[74]. Uncertainty in context data acquisition determines the usage of probabilistic approaches requiring network structures and prior probability distributions. Bayesian networks [*55*] are mathematically well founded and efficient, but as noted by Liu and Zhang [*48*] suffer from the following weaknesses: i)mutual exclusivity required by the computing hypotheses; and ii) inability to account for the general uncertainty. Dempster Shaefer Evidence Theory(DSET)allows accountability for epistemic uncertainty in the data model and may prove better for reasoning on information when independent heterogeneous sources are reporting the same context [27],[57],[64]. Fuzzy Membership functions can be used to map low level feature values into application specific concepts. DBNs and HMMs can capture the effect of prior context values on current context probabilities if the context in question is affected by historical values [*27],*[64] Most context aware applications require constitution of higher level contexts by combining other lower level contexts. The challenge here is to determine how to combine low level contexts. Researchers have employed Bayesian Networks, Decision Trees, First Order Logic, and Rules [10],[47],[64] to perform similar tasks. In cases where future maintainability is a

concern, the use of rules should be limited and an unsupervised learning approach can be used for automatic adaptation. Artificial Neural Network or Genetic Algorithms [64] may be considered when the application involves predicting contexts over numerous possibilities. These are particularly good for context models exhibiting a high degree of interconnectivity; where there is sufficient time and training data for developing an accurate inference model. On the other hand, if the context can be reduced to some function of its inputs (such as recognizing human voice from a sound sample), a discriminant analysis [10], [52] approach may be appropriate.

Many, many approaches have been to model and reason about context. There are approaches that are clearly best suited for low level information fusion, while other approaches are better for high level context. This illustrated in Figure 10 below.

Context	Applicable Reasoning Approach
Social Context (Who is nearby? What's their availability? How can I reach them?)	Logic, Rules
Availability (Busy, Free)	Rules, Decision Tree
Proximity(close, near, far)	Fuzzy Logic
Activity(Driving, Exercising, Sitting, Walking, Running, Sleeping, Eating, Talking)	HMM, Decision Tree, Bayesian Network, DSET, Discriminant Analysis
Mobility(in moving vehicle)	HMM, Dynamic Bayesian Network
High Level Identity(family member, coworker, neighbor, friend, unidentified)	App specific Social Ontology, Rules
High Level Location(work, school, home, store, library, post office)	Bayesian network, Decision Tree, Logic, Rules
Low Level Identity (Tom, John, MID)	App specific DB Lookup
Low Level Location (GPS coordinates)	No reasoning/Direct from device

Figure 10 Reasoning Techniques applied to context

The following sections review some benefits and disadvantages of a few reasoning techniques used in existing context aware applications.

2.6.1 *Bayesian Networks*

Bayesian Networks (BN) as well as ontological reasoning has been applied to pervasive context inference [7],[64],[70],[77]. BNs provide a solid theoretical foundation for representing uncertainty. Hindrances to the use of BNs for inference in pervasive domains include: i) lack of expressiveness; ii) inflexible instance specific models; iii) inability to represent objects and relationships that cannot be specified in advance; iv) inability to express generalized recursions of objects; and v) intractable worst case inference.

2.6.1.1 First Order Bayesian Networks

Multi-Entity Bayesian Networks (MEBN) resolves many limitations of traditional Bayesian networks and offers great potential for pervasive context inference.

MEBN [46] is a knowledge representation formalism that augments the expressiveness of First Order Logic (FOL) with the sound theoretical foundations of Bayesian Networks (BN). Application domain concepts are segmented into groups of BN fragments called MFrags. Groups of MFrags collectively represent a joint probability distribution, referred to as an MTheory. Nodes in MFrags can be parameterized. Thus, with MEBN, classes/types of random variables can be defined. Generalized recursion is also possible within Mfrags. An instance of a random variable may depend directly or indirectly on another instance of the same random variable. As queries are posed to the MEBN system, Situation Specific Bayesian networks (SSBNs) are dynamically constructed to answer those queries. SSBNs are minimal BNs needed to compute posterior probabilities on targeted random variables in light of provided evidences. MEBN claims to be the first language having all the following properties: i) the ability to express a globally consistent joint distribution over models of any consistent FOL theory; ii) a “proof theory capable of identifying inconsistent theories in finitely many steps and converging to correct responses to probabilistic queries”; and iii) a built-in mechanism for extending and refining theories in the light of observations [46]. Additionally, MEBN provides representations for quantifiers, function composition, and logical connectives. It should be possible to translate any knowledge represented in FOL to a set of MEBN theories.

2.6.2 *OWL Ontologies and Descriptive Logic Reasoning*

Ontologies offer the expressiveness of first order logic and a formal specification of data semantics. They are a desirable representation of knowledge since semantics are implicit in the representation. Objects with their attributes, relationships and constraints can be defined. This makes ontologies suitable for representing complex relationships and knowledge sharing.

OWL reasoning can be used to reason about:

- Knowledge Consistency (discovers if the models or knowledge is contradictory)
- Existence of data instances, property values
- Subsumption relations
- Derive implicit relationships
- Knowledge Equivalency

Ontologies are good for answering queries that involve: search on attributes, search for class/types instances, inheritance/subsets, relationships derived indirectly. Some examples of information that can be derived with ontologies include:

- Implicit knowledge/relationships -*Tweety is a bird, however tweety is not a flighted bird. So, Tweety cannot fly*
- Knowledge base consistency - *a person cannot be present in two locations*
- Find all people at location
- Is Sheryl at location
- Is X attribute of Object X equal to Y
- Is Lynda an ancestor of John?
- Which printers are available? However, availability may be better modeled with a belief network, fuzzy logic or rules

When we refer to ontological reasoning we are primarily making reference to Description Logics (DL). Description Logic reasoning is the decidable subset of first order logic. SROIQ,

SHION and SHIF are examples of DL languages. Today, there are probabilistic variances to support inference with uncertainty: P-SHIQ and/or P-SHION. Several DL implementations exist {*Pellet, Racer, Fact++, HerMit*}[3]. These support inference from ontological languages.

Though the differences in underlying DL languages implemented by DL reasoners affects performance characteristics of inferencing, in general DL reasoning can be used to: In general inference with existing ontological languages is computationally expensive, does not scale well and offers no direct way to represent uncertainty or logically combine concepts to model complex high level contexts.

Ontologies are used extensively in the semantic web for many domains {*medical, automotive, education, earth & space* } There are current efforts to convert existing knowledge bases to ontologies[4] for reasons including those above. OWL[5] 2 is the most recent version of the Web Ontology language recommended by W3C. It offers three dialects that constrain the language to improve inference performance, interoperability with databases and rule languagues. In order or decreasing representation power there exists: OWL Full, DL, EL and Lite. Reasoning based on OWL-DL is decidable. This is accomplished by constraining OWL full concepts that make reasoning unwieldy. DL reasoning has exponential time complexity. OWL-EL Limits OWL to expressions that can be decided in Polynomial Time. One existing EL reasoner is Pellet EL.

Wide acceptance of OWL has led to stable tools, apis, platforms and multiple concrete syntaxes for development. Reading the Use Cases that have guided the OWL 2 standard helps in understanding how useful it can be for knowledge representation.

The tradeoff between the cumbersome; verbose OWL representation and its beneficial occurs when when the knowledge base(KB) is sufficiently large; where consistency maintenance and

[3] See http://www.cs.manchester.ac.uk/~sattler/reasoners.html for a list

[4] Stanford University is currently converting its Immune Epitope Relational Database to OWL

[5] http://www.w3.org/2007/OWL/wiki/OWL_Working_Group

inferencing would be too unwieldy using other techniques. For small KB and those that involve little relationship inferencing, other representation and reasoning techniques might apply. The paradox for pervasive environments is that fact the even DL reasoning performance is often unacceptable for small KBs. In one work, Pellet took 11 minutes to classify a relative small ontology. This is yet another reason for offline classification of large ontologies. In highly dynamic environments, DL reasoning must be used with care.

2.6.2.1 SWRL

SWRL is a proposal for a Semantic Web Rules Language, combining (OWL DL and Lite) with the Rule Markup Language. SWRL has the full power of OWL DL, but at the price of decidability and practical implementations. Rules are of the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. Figure 11 shows two SWRL rules. They specify i) whenever a building has an intrusion status of 'ALARM', the threat potential must be 'HIGH' and ii) If a device that is located in an area of the building has an intrusion status of 'ALARM', then the status of that area as well as the building must also 'ALARM'.

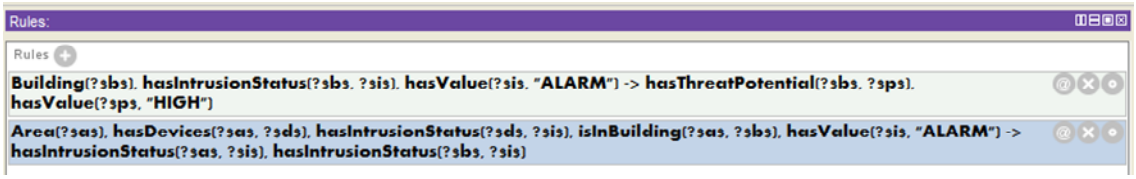

Figure 11 Sample SWRL Rules in Protege

SWRL and OWL can to be verbose, cumbersome and error prone. While rules are expressive, large rules sets are cumbersome and inefficient when compared to statistical approaches. The mature tools available for OWL mitigates some of this challenge.

2.7 Background Summary

The following figures and tables summarize our observations regarding the suitability of OWL, SWRL and traditional BNs for context modeling. Rules offer more espressiveness while tradtional BNs offer greater inference speed and scalability.

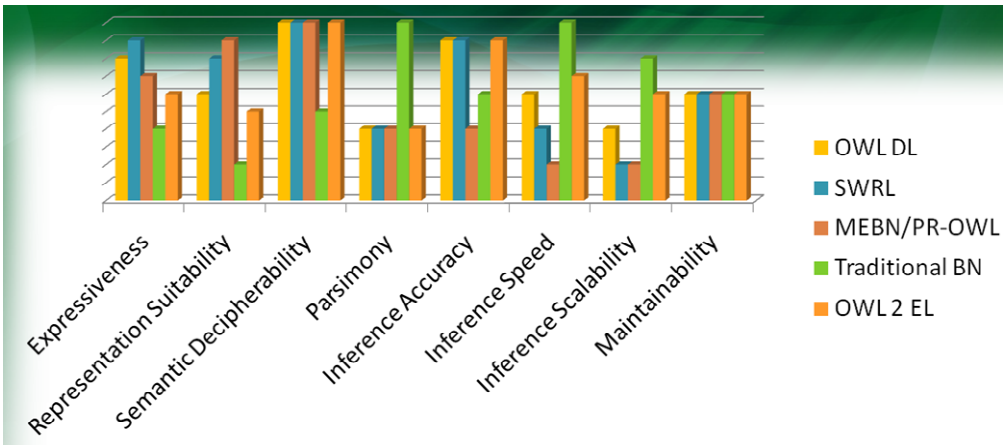

Figure 12 Comparison of OWL, SWRL and BNs for Knowledge Representation

CHAPTER 3

HYCORE DESIGN AND DATA MODEL

This research proposes enabling solutions for adaptive and effective reasoning in pervasive environments. Considerations include: shareable context data models, integrated reasoning, deriving integrated quality of context, and adaptive generalized context reasoning.

3.1 Requirements Analysis

Ideal requirements for a reasoning engine were derived after surveying many context aware applications and frameworks (see Chapter 2). These ideas have been selectively borrowed and combined from existing frameworks and software engineering concepts. The following subsections discuss these requirements.

3.1.1 *Requirement: Modular Components for Reasoning*

As can be seen across contemporary applications, many approaches are needed to meet the requirements of a complex context aware application. Variety in reasoning, modularization and optimization equips a reasoning framework to support a variety of reasoning techniques across heterogeneous context sources. This results in greater reuse, as application designers are able to map context elements to appropriate reasoning modules. Decoupling data, reasoning and knowledge management promotes context sharing among applications.

Such architecture also supports maintainability, extensibility and evolution of reasoning. Reasoning algorithms as well as derived contexts (i.e. knowledge) may be reused among applications. APIs for abstraction add greater flexibility in the types and sources of context that can be supported. Mechanisms for reasoner optimizations provide application designers with an additional tool for meeting system goals. Optimization can be accomplished by exposing algorithm parameters tuned using domain experience along with experimentation. Where there is

variety, interchangeable algorithms can be used in parallel to improve accuracy. Experimentation will ultimately validate the suitability of chosen reasoners for a given application.

3.1.2 *Requirement: Consistent and Dynamic Context Data model*

Greater context sharing among applications can be achieved with a consistent representation of data. At times, reasoning requires a unique set of models and formatting. Also, modeling formalisms evolve with time. Utilities for model translation reduce the burden on application developers; removing another obstacle to framework reuse. A dynamic model strengthens context maintainability. Along with modularization, it also supports incremental development of applications and reasoning techniques. Reasoning data models should reflect the current reality. As applications mature, new concepts may be augmented to the data model, while redundant and obsolete elements are pruned.

3.1.3 *Requirement: Mechanisms for Context Maintenance*

- Anomaly Reporting and Conflict Resolution: There are times when two sources of context report conflicting observations. The same could be true of independent algorithms making parallel inferences. The resulting inferences could conflict. Additionally, an observation or derived context could violate the data model. When issues cannot be automatically resolved, a sufficient context reasoning architecture has a means to handle anomalies (log, service call, JMS notification, email, etc.).
- Context integrity: Context integrity includes considerations for context expiration, provenance/traceability of context, and retention of pertinent context information at every level of the data model. To maintain an accurate view of current reality, every element of context should have an associated expiration. Expired context should be stored for reasoning based on history, but never used where reasoning requires fresh data. Context derivation includes information regarding time, quality, source and method of deriving context. Every contextual element can be traced back to its source. This is useful in evaluating credibility, quality or mitigating conflict. Where storage is not an issue, retaining pertinent levels of

context information may be useful in offline analysis, supporting future applications and error handling. Other context integrity issues to consider include: context purging and synchronization with other knowledge stores.

- Consistent Storage and Retrieval of Context: The usability of a reasoning framework is improved when knowledge can be stored and retrieved in a consistent manner. Besides supporting direct application queries, case-based reasoning techniques rely on past instances of events. Specifically, HMMs and DBNs use histories to infer current events. To support a wide variety of reasoning approaches as well as offline data analysis, storage of context history is needed.

3.1.4 *Requirement: Distributed Context Reasoning and Collection*

A centralized approach to context reasoning and collection is limiting. Pervasive devices are limited in resources, so it is wise to make use of backend or offline reasoning where possible. Also, if a complex application is to compute multiple heterogeneous pieces of context using disparate approaches, parallel execution in the distributed system could improve performance. Existing toolkits may already offer the best suited implementation of a reasoner. APIs for interfacing external reasoners assists in rapid development and innovation. Though many applications do not require or support remote communication, a sufficient reasoning framework would minimally offer the option of distributed reasoning.

3.1.5 *Requirement: Quality of Context (QoC) Support*

Along with variety in reasoning choice and parameterization, QoC based context reasoning further helps application developers to meet system requirements. Maintaining QoC requires retention of associated data and statistics through many components of the architecture. Feedback techniques are useful in validating and grading performance. A consumer of context may be concerned with the following QoC factors: accuracy, speed, trustworthiness, data freshness, resolution and class. Class refers to data type with associated semantics and representation. It is reasonable to expect classifiers and other types of reasoners to offer varying

classes and granularities on the same categories of context. To illustrate: there could exist multiple reasoners inferring 'activity' for a single sensor subsystem. Using the same sensory inputs, each Reasoner applies distinct algorithms producing distinct classes for use by different applications. Also, reasoners inferring contexts like location and time may offer greater resolution with corresponding performance cost.

3.2 HyCoRE Architecture

The architecture of HyCoRE shown Figure 13 conceptually meets the ideal goals discussed in the previous requirements section. The logical layers of processing are those found in existing context frameworks, namely: i) context acquisition (including sensing and preprocessing); ii) context consumption; iii) context reasoning; and iv) knowledge storage and retrieval.

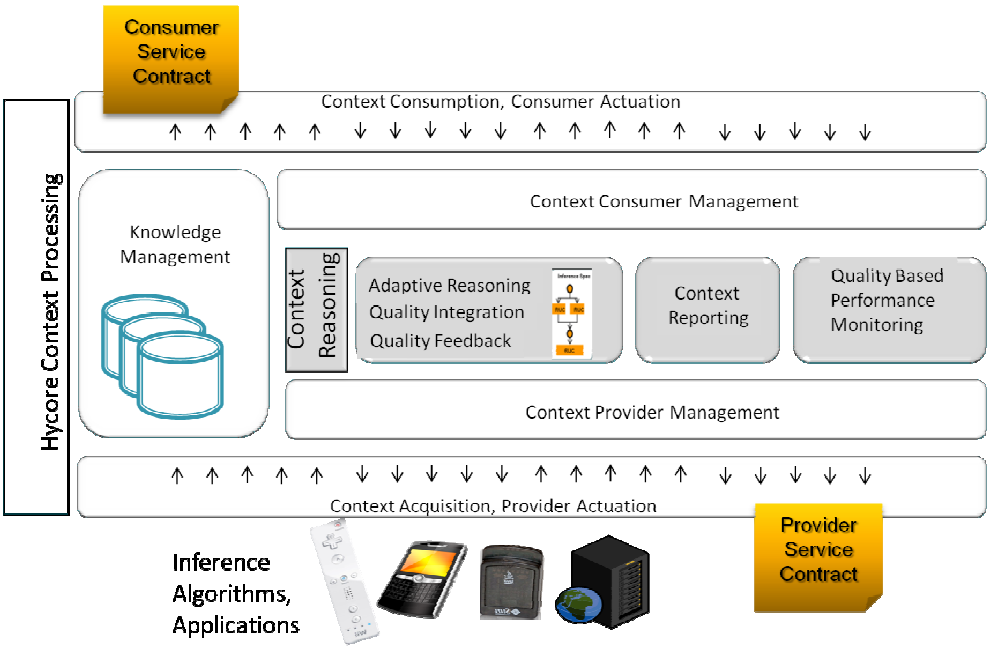

Figure 13 HyCoRE High Level Architecture

This design decouples context sensing and consumption from high level reasoning. Context providers are abstractions of devices, services, applications or reasoning algorithms providing context. The context consumer component generalizes subscribers of context. Provider service contracts contain descriptions of capabilities/contexts available, as well as acquisition

procedures. Consumer service contracts specify consumer context and quality requirements. Both consumer and provider components include actuation specific to their participating entity.

At the context acquisition layer, context providers offer quality annotated context data/ services to HyCoRE. Context sources are abstracted internally as ReUsable Reasoning Components. RUCs are generalized parameterizable modules which are used to abstract reasoning algorithms (including statistical, ontological, logic and rule based approaches), data transformers, and utilities for sensory data acquisition, knowledge query and maintenance. Low level concepts can be fused into complex high level concepts by linking RUCs in an application specific order. As needed, the context reasoning subsystem selects appropriate RUCs for participation in orchestrated reasoning plans to produce high level context. Provider actuators separate the context source from context reasoning using utilities and APIs to map source data into HyCoRE domain concepts. Actuators act as data collectors and may employ any buffering, preprocessing and feature extraction techniques as long as each conforms to HyCoRE actuator interface.

At the consumption layer context consumers request context information, providing quality requirements and response actuation details in the consumer service contract. As context requests are received, appropriate reasoning plans for inference are activated.

The knowledge layer contains stored context data. Selective elements of context (direct and inferred) are maintained along with pertinent derivation information. It is useful to maintain much of the data related to context derivation. Applications may need to verify pieces of low level context used in context derivation data to meet QoC, security policies or to mitigate conflict. New context is sanitized by performing data consistency checks against existing knowledge models, as well as system polices. Knowledge storage also includes services for context conflict resolution, synchronization, cleanup, expiration and error reporting.

The context reasoning subsystem is the heart of HyCoRE. It is responsible for context reasoning planning, execution and adaptation. At this layer, inferencing is accomplished by instantiating a reasoning plan best matching application requirements. Instantiated reasoning plans are called context flows. Context flows are independent inferencing tasks that can be run

in parallel. These are created by composing HyCoRE reusable reasoning components (RUCs). Context flow configuration includes considerations for asynchronous execution, periodicity and persistence. A context flow is represented by a directed acyclic graph of nodes which include references to selected context providers as RUCs. Reasoning quality is monitored and plans are adapted to sustain context inferencing and quality in the face of dynamic provider quality and availability. Context flows will be discussed at length in succeeding sections.

In summary, HyCoRE is more than a broker or central repository. A single instance of HyCoRE may employ distributed reasoning and KM components. As will be discussed, it is the source agnostic information descriptions that enable the orchestration of generalized information into usable knowledge; independent of data source type or location. HyCoRE orchestrates interchangeable reasoning elements and dynamically adapts based on quality indications. Adaptive context reasoning based on adaptable flows contributes to efficient use of resources. With regard to reasoning using context flows, the HyCoRE middleware manages:

- ✓ **Context Flow Creation**- runtime instantiation of reasoning plans based on available providers;
- ✓ **Quality Management**-deriving integrated high level context quality measures;
- ✓ **Reasoning Adaptation**-identifying triggers and adapting reasoning flows to support middleware policy as well as application requirements;
- ✓ **Asynchronicity**-supporting context push and pull both synchronously and asynchronously;
- ✓ **Provenance** –tracking context sources contributing to high level inference. Also, change history may be associated with every inference;
- ✓ **Reuse**- attempts minimal reasoning in support of multiple consumers, reusing underlying resources and reasoning plans.

A more detailed discussion of reasoning subsystem components is deferred until after the HyCoRE data model and context quality quantification are explained (See SectionsCHAPTER 4 3.3 and 4.2). Elements of the data model are used as internal and external components collaborate to accomplish context reasoning. Quality quantification is central to decision points in context reasoning, as will be discussed.

3.3 HyCoRE Context Data Model

As illustrated in Figure 14, context is the basis of information exchange between HyCoRE internal and external reasoning components. It is a general term for information identified, modeled and used for the purpose of context aware adaptation or higher level reasoning. Often, it is a knowledge by-product of stages of reasoning and transformation on low level data. The context that a particular HyCoRE instance reasons about and publishes is driven by administrative configuration and application extensions of the HyCoRE data model. So, HyCoRE is not intrinsically limited to a particular type of context, but may be limited by available context providers. Multiple knowledge models, supporting varied types of context, may exist in a single instance of HyCoRE. (*Note that the terms knowledge, information and context are used interchangeably in this document).*

As mentioned previously, there are many correct approaches to modeling *context* [7],[13].[19],[21],[28],[32],[36],[38],[40],[43],[63]. The important concern is for the model to sufficiently capture the targeted context characteristics, support efficient query, retrieval and maintenance. Context characteristics include: attribute heterogeneity, dynamism, availability, temporality, source derivation, credibility, and uncertainty. A model that is coupled to a specific type of application may have inherent performance improvements over a general purpose model. Despite this possible loss of efficiency, HyCoRE employs a general context model approach since heterogeneous low level data as well as complex inferred context characteristics must be captured. Further, since HyCoRE is designed for effective orchestration of local and distributed reasoning components into a reasoning plan executed to produce high level context, the essential information necessary to support this function is also provided in the HyCoRE data model.

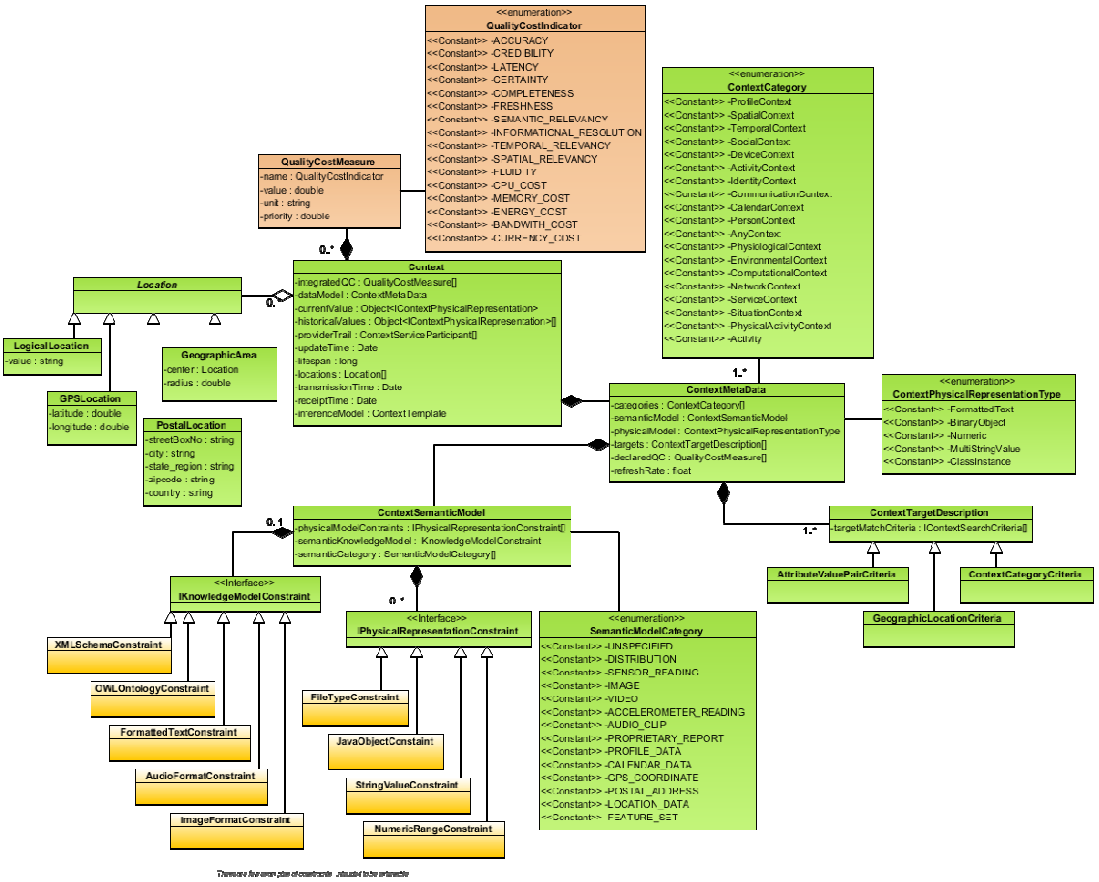

Figure 14 HyCoRE Context Data Model

Not using the popular sematic web ontology language (OWL) as a conceptual representation is justified. OWL is useful if the context expected involves inference on relationships between concepts. Many description logic (DL) languages/tools do this efficiently. The tradeoff between the cumbersome, verbose representation of OWL and its benefits is found when the knowledge base (KB) is sufficiently large; where consistency maintenance and inferencing would prove too unwieldy using other techniques. For small KB and those that involve little relationship inferencing, other representation and reasoning techniques are more appropriate. The paradox for pervasive environments is the fact that DL reasoning performance is often unacceptable for small KBs (see discussion in Section 2.6.2). So, we have chosen to represent concepts using UML, avoiding any implementation specific association. However, the multimodel support in

HyCoRE allows the architecture can be configured to support OWL or other XML schema based components. This is a conceptual data model that may be implemented any number of ways (XML, OWL, JSON, Object Serialization etc.).

The HyCoRE context data model offers value to context modeling in the following ways: i) Context Information is modeled as source agnostic meta-data; ii) Disparate types/categories of context are supported; iii) Multi-dimensional quality attributes are associated with all elements of context; iv) There is support for multiple context knowledge models with separation of physical representation and semantic interpretation concerns; and v) Finally, this model captures context provenance. These unique features of the HyCoRE context model are discussed in the next section.

3.3.1 *Source Agnostic Context Meta-data Model*

It is common practice to extract meta-data of large bodies of information to facilitate manageable information sharing. We use context meta-data to enable reasoning generalization and reuse; enabling machine interpretation of context. Meta-data is not specific to device, platform or reasoning algorithm. It simply describes the information in a machine interpretable manner. For this purpose we have identified information supplementary characteristics necessary in describing context in pervasive application domains. These characteristics include: i) context categories ii) quality cost indications iii) both a semantic and physical representation of data; and iv) a list of targets to which a context applies.

The context target specification describes domain objects on which context information applies and is considered valid. A target is a generalized description for an object of knowledge that previously exists and is understood by both the consumer and provider(s) associated with a particular context element. Targets of context include: person(s), location(s), device(s), time interval, situation .etc.

Context categories, quality indications, semantic and physical representational specifications of the meta-data model will be discussed in the following sections which highlight other unique features of this model.

3.3.2 *Multi-Centricity*

Centricity refers to the target domain in which context information applies. Often, the centricity reflected in the data model and reasoning is singular and tightly coupled with a specific application (see related work in Chapter 2). The HyCoRE model is distinct in supporting many centricities of context, including user, device and location centric contexts; thereby supporting varied application types. In the model illustrated by Figure 14, *context category* is a label describing the type of information; identifying its centricity. Since context is a general term for information produced as a reasoning by-product, it may be classified many ways. Section 2.1 discussed the many types of context existing in literature. Context category(ies) capture the types to which a piece of information is applicable. Examples include: activity, identity, situation social environmental and spatial contexts. Often informal applies to multiple categories, so this model of context supports specifying a list of applicable categories.

3.3.3 *Multi-Dimensional Quality Representation*

In the HyCoRE architecture, every component affecting context derivation is associated with a quality/cost model. Sensors, reasoners, general context providers, transformation functions all have quality indicators that affect resulting context. The quality model includes both declared and observed quality/cost indicators appropriate for the type of component. It is impossible to foresee all measures that could be determined to be quality related. Also, not all quality indicators are appropriate for all sources of context. The concept of context quality will be discussed more in Chapter 4; wherein the observations determining quality attributes/features provide application-specific added value, thus supporting declaration of different quality indications for each component.

All context providers have an associated QC indication as does the context that is inferred by them. Context consumers specify quality requirements. The HyCoRE system has both quality and cost policy settings. During the context reasoning process, an integrated high level context QC indication is computed and provided to the consumer in the context report. This high level QC indication is further used by the HyCoRE system for computing QoCS measures and evaluating reasoning adaptation needs. More details on deriving high level QC indications, quantifying QoCS and reasoning adaptation are discussed in later chapters. HyCoRE represents quality/cost of context as a generalized vector of attribute values pairs with associated priority and unit indications. For simplicity we assume common understanding of quality semantics implied by attribute names (i.e. accuracy, latency..). Refer to Chapter 4 for quality discussion.

3.3.4 *Support for Multiple Knowledge Representations*

Many inferencing toolkits support only a few types of context (see Section 2.2). By using context meta-models, HyCoRE supports heterogeneous representations of knowledge. Notice the context data model element in Figure 14. The data model contains an objective description of context information that is not specific to a device; also called context meta-data. Context meta-data specifies the physical and semantic criteria for context equivalency. Context consumers are paired with providers on this basis. Also, providers are interchangeable based on matching meta-model information.

The physical context model describes the format for physical representation of context values. A survey of context reasoning approaches reveals that numerous physical models are used for the same categories of context. Physical models include: number, name/value pairs, object, file reference, string and multi-string values. A context element's physical data model is one of many ways to realize that context value for a specific application. For example: A Tri axis accelerometer sample may logically contain multiple lines of X, Y, Z axis acceleration readings. Physically this may be represented as a string or Java object. Context provider developers

determine which representation is most suitable and publish the chosen model in provider service contracts.

The semantic model elements collectively clarify the syntax and semantic intention of the physical context model. The semantic model includes:

1. Semantic Category – specifies the logical class of data contained in the physical model which may include: image, video clip, audio sample, accelerometer sample, feature set, activity, location, GPS coordinate, and postal address. Note that the semantic category is distinct from the context type. An ‘activity‘ context type can have several physical and semantic models with associated semantic category. The semantic category is tightly coupled with the data representation. Whereas the type of context is a broader classification of information. For example: A low level ‘activity’ context represented by raw data may have a semantic category of ‘accelerometer sample’. A transformation algorithm which manipulates raw data produces ‘activity’ context with a semantic category of ‘feature set’. A follow on reasoning algorithm may transform the feature set in to a high level ‘activity’ context with a semantic category of ‘physical activity’.
2. Physical model constraints- impose syntax and value limitations on the physical model. For example: a number may be constrained to the range [1-10]. A string physical model may be constrained to a subset of values: walking, sitting or standing.
3. Knowledge model constraint- adds semantics to the physical model by specifying a schema, ontology, java class or other semantic limitation on the context representation. For example: an audio clip can be physically represented as a binary object with a semantic knowledge model specifying a wave format. A target location can be represented physically as a string, but the semantic knowledge model may be a class with field designations for GPS latitude, longitude and direction.

Context is always associated with a meta-data model which affords semantic interpretation. This model is structured such that possible relevant context providers are *implicit*. Though the context

provider reporting the context is a specific instance, any provider with matching input criteria could have provided it. This means that multiple knowledge representations are also implicit, since context providers using different internal knowledge representations may infer the same class of context resulting in same external meta-data modeled values. Context meta-data with physical and semantic representation models enable generalization and reuse. HyCoRE context reasoning adaptation logic, discussed in Chapter 5, also centers on matching of meta-data.

3.3.5 *Context Provenance*

Context provenance identifies the sources, process of derivation and change history of context data values. Limited provenance is revealed by context attributes shown in Figure 14. A context providerTrail attribute provides a way to trace back through serialized context flows. This source trail could be important in off-line provider analysis in sensitive applications (i.e. geo-political, military, criminal forensic applications). 4WH is an acronym for the descriptors: ***w**hen, **w**here, **w**hat, **w**ho, and **h**ow.* These descriptors are used in many domains for describing information. This data model supports declaring and determining these values for a given context. The work in [3] uses 5WH. That work also considers *why* as reflection of application targeted use of context. HyCoRE is not concerned with *why* context is required. It uses application requirements and system defined policies to guide behavior. The *when* descriptor reflects the time domain of context; revealed by timestamp attributes. *Where* is the spatial domain of context and is revealed by location attributes. *What* describes the content of context and is revealed by the value and data model elements. *Who* as the source of context is revealed by the *providerTrail* attribute. *Who* as the target of information is revealed in the target list included in the context-meta data. *How* reflects the method(s) of inference; revealed in both the *inferenceModel* and *providerTrail* attributes. The *inferenceModel* identifies the context template. Context templates are patterns of reasoning defined by domain experts; specifying how information is combined and transformed to derive high level context. Templates are realized in real-time as context flows in response to consumer request and provider update. Templates specify how information is combined and transformed to derive high level context.

3.4 HyCoRE Context IO Model

In HyCoRE, consumer and providers are discovered, actuated and managed using a common context I/O model. Figure 15 depicts the I/O model. Table 3, is a generalized consumer requirement specification to be discussed in the next section.

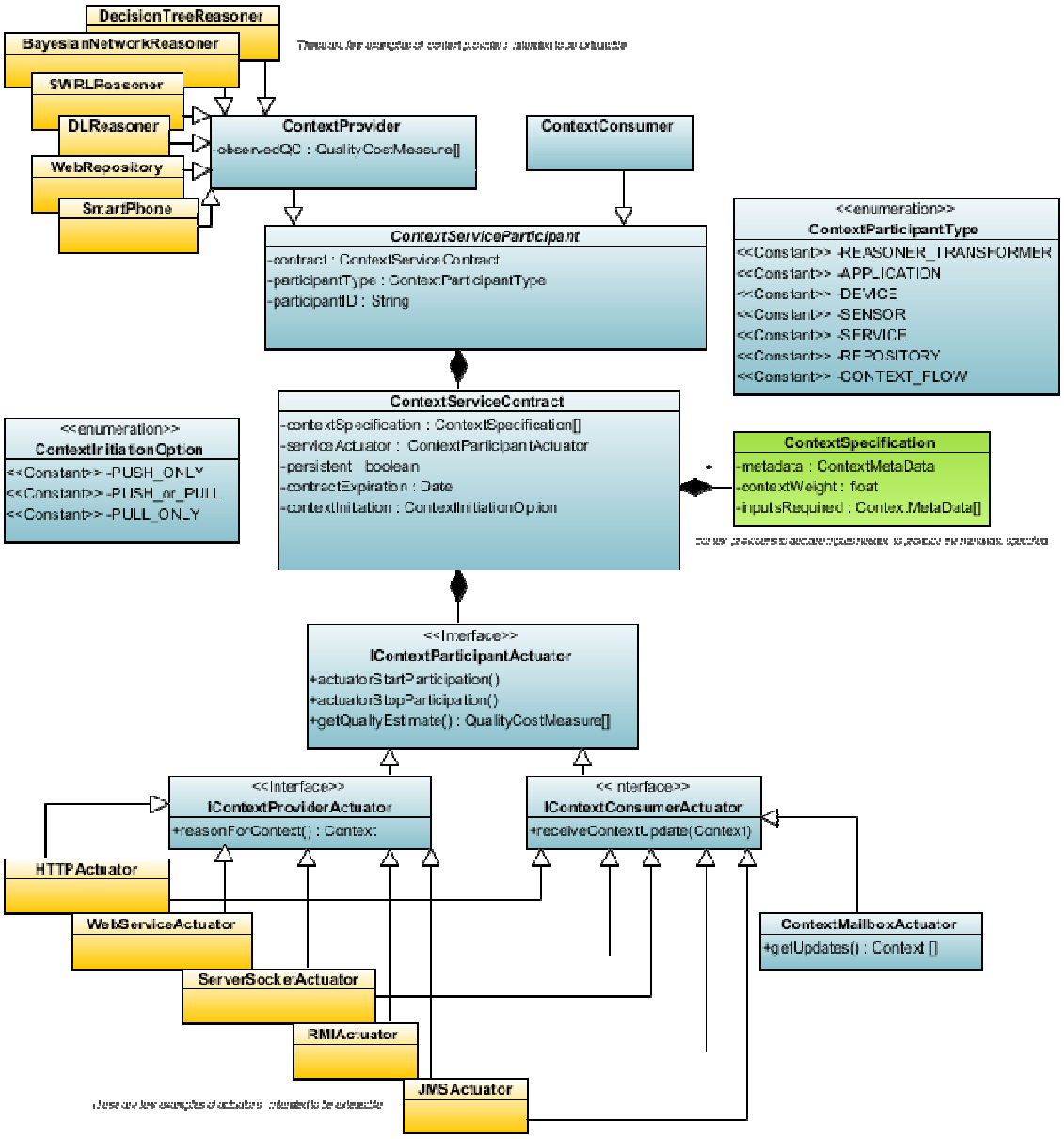

Figure 15 HyCoRE Context IO Specification

Table 3 Consumer Quality Requirement Specification

Specification Term	Definition/Syntax
Application Context Goal	*f*(Context Quality Goals) ∈ [0,1]
Context Quality Goal	*f*(Context Quality Indicators)∈ [0,1]
Application Context Goal	1..* W_{ci} (Context Quality Goal) , *where* $\sum_{ci=1}^{ci=n} Wci = 1$
Context Quality Goal	ContextID{0 |1..* W_{qi} (QualityExpr) where $\sum_{qi=1}^{qi=n} Wqi = 1$, 0 indicates that the context information is desired, but the application makes no quality constraints. This definition enables the application to simply set preferences on quality indicator values or to express unique formulas as criteria.
W_{ci}	context identifier weight, w ∈ [0,1] ; a weight of 0 indicates that the context information is desired when available, but is not critical
W_{qi}	quality indicator weight, w ∈ (0,1]
QualityExpr	1..*< *optional* RelativityOP> QualitySentence
QualitySentence	QualityID,OP,(,),QVal
ContextID	Unique alphanumeric identifier for a context value
QualityID	Unique alphanumeric identifier of a quality indicator
OP	MathOP, RelativityOP
MathOP	+,-,*
RelativityOP	=,<=,>=,<,>, AND, OR, NOT
QVal	*quality value;* any number <*optional* Unit>
Unit	Middleware specified unique alphanumeric qualifying such measures as time and temperature.

The figures above are intended as conceptual syntax. Actual implementation may vary.

3.4.1 *Context Consumers*

A context consumer is any formal subscriber of information in HyCoRE. The consumer service contract is an agreement between HyCoRE and a consumer application regarding context and quality. Consuming applications are allowed to specify context desired and quality required. The application is best placed to provide information that distinguishes what is relevant and valuable. The application context specification is a requirement by which the middleware system measures its success. It is a syntax that allows an application consumer to describe its quality goal as a

mathematical expression containing context identifiers and weighted quality indicator preferences. Weights reflect application priorities of context and quality. Figure 15 and Table 3 conceptually illustrate consumer requirements. To accomplish effective reasoning, consumer requests for context are matched to reasoning plans irrespective of the knowledge models or inference algorithm used for derivation. A consumer request describes context using the meta-data model which includes generalized physical and semantic descriptions of information desired. Figure 16 below is a prototypical example of a context consumer request containing meta-data. The consumer is requesting the location of three targets, specifying an accuracy of 92%. The negative priority indications in the example represent the consumer's indifference to those QC indicators. The expected format and context of the data to be returned in the context report is as specified. The meta-data requirement specification contains enough information for deriving a reasoning plan for inference.

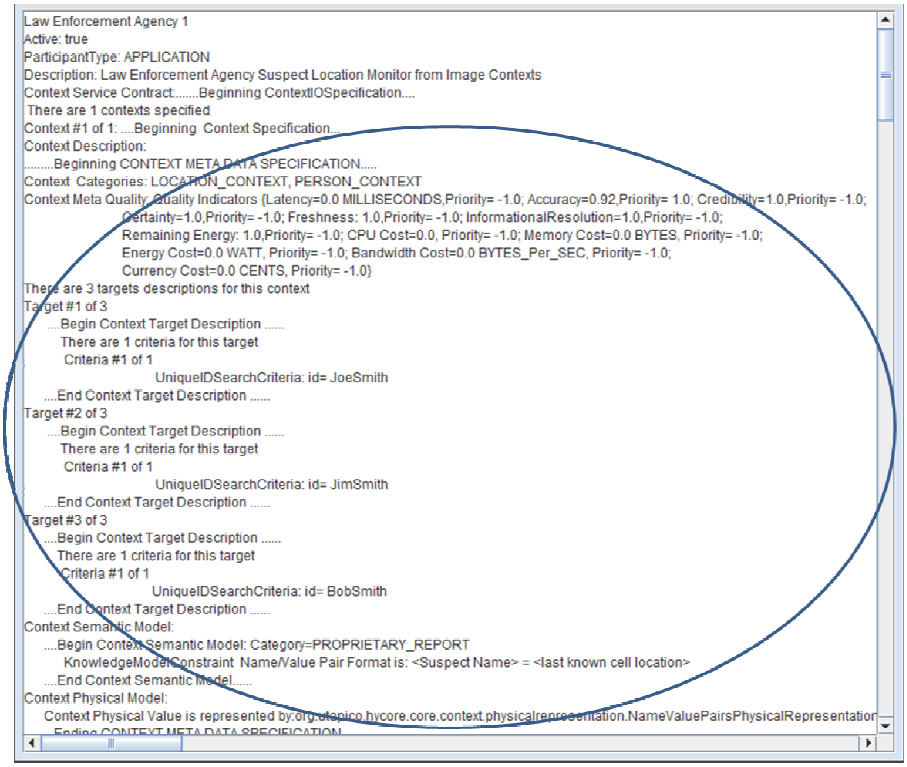

Figure 16 Example Context Consumer Requirements

3.4.2 *Context Providers*

A Context Provider is any device, repository, algorithm or service providing context. A single provider may be the source of several types of context. Providers may use varied knowledge representations as a basis for inference. The most popular models in pervasive computing are rule-based, decision tree, naïve Bayesian and hidden Markov models (see Section 2.6). However, the provider's internal model is irrelevant for meta-data matching. It is the source agnostic meta-data model elements that are important to HyCoRE. Providers include an actuation component which serves as a control for local or remote communication. Context providers publish their capability to share context using a service contract. The context provider service contract along with periodic quality verification, give HyCoRE knowledge of types and quality of context available. Section 3.1 discussed ideal architecture requirements and suggested that a desirable feature of a context provider definition would include optimization parameters or some means to control internal behavior. This design could be extended to support such attributes. Alternatively, context provider may publish multiple services contracts; where distinct configurations are actuated separately, resulting in different QC indications. In this case, HyCoRE would choose between configurations the same way as it would choose between different providers. Figure 17 below is a prototypical example of a context provider declaration containing meta-data.

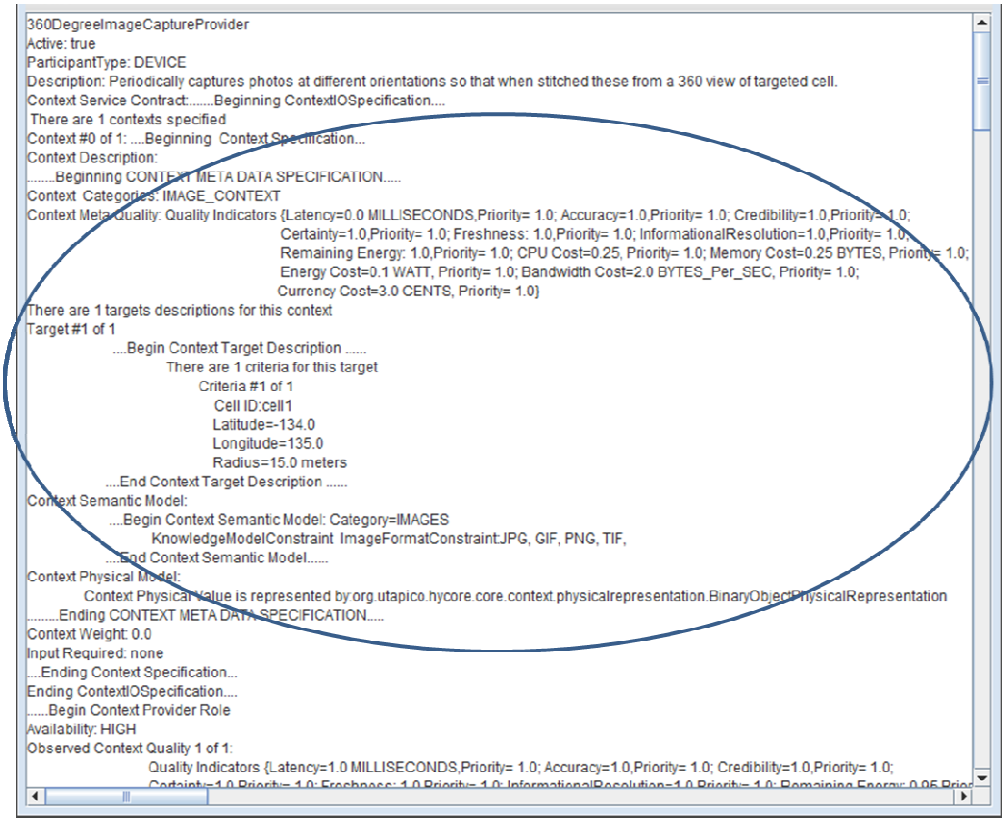

Figure 17 Example Context Provider Context Meta-data

3.4.3 *Reasoning Data Model*

HyCoRE's context reasoning begins with declaration of reasoning plans or templates. These are dynamically realized and bound to specific context providers as context flows. A flow contains a reasoning stream which contains a serialized list of reasoning nodes. Details on context reasoning nodes, templates and flows are given in the next section. Figure 18 is a UML representation of the HyCoRE's reasoning model.

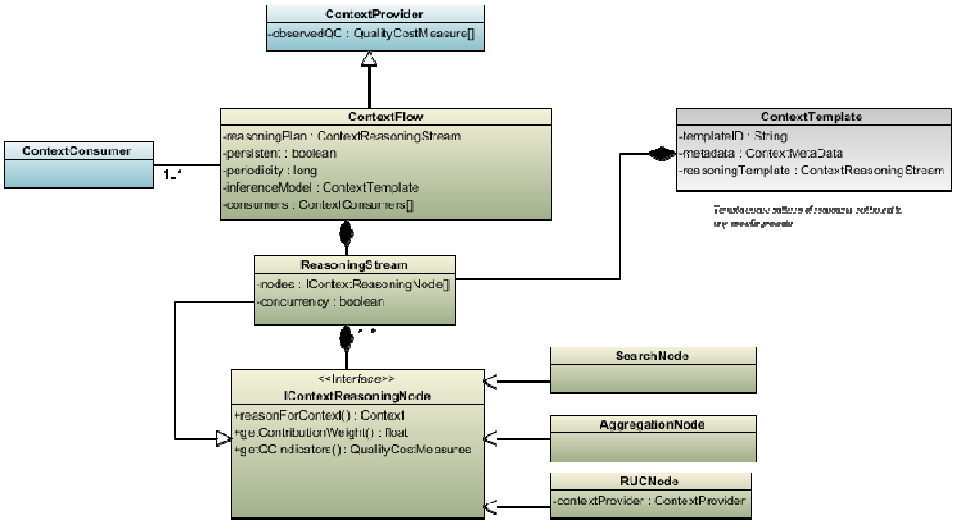

Figure 18 HyCoRE Reasoning Data Model

3.4.3.1 Context Templates

A context template is a generalized context inference network configuration, representing a formal description of a single high level context. The template does not specify context providers or reasoners. It only describes how published context specifications are used to derive higher level context. This generalization supports dynamic and adaptive context inference. HyCoRE context reasoning is accomplished through the execution of instantiated reasoning plans (*context flows*). A reasoning plan or template is represented as a directed graph of reasoning components, as illustrated by the context template depiction in Figure 19. Many works represent context reasoning processes with directed acyclic graphs (DAG) for their parsimonious representation and available graph theoretical solutions.

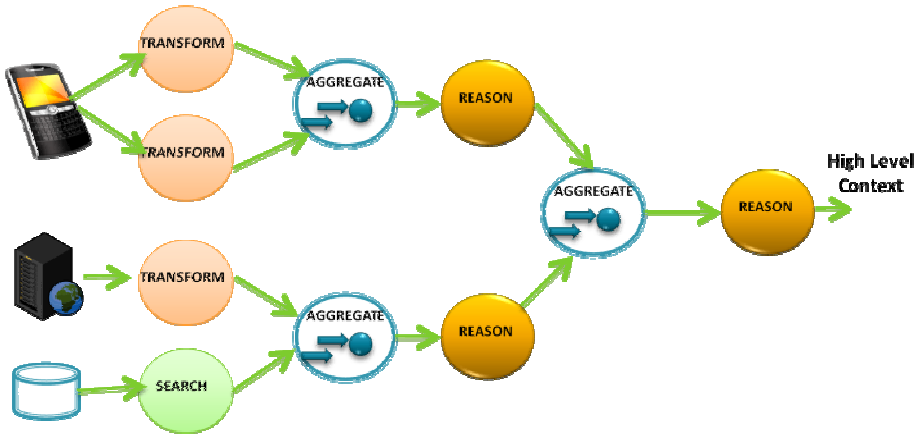

Figure 19 Unweighted Context Template

Context templates represent domain expert knowledge of transforming raw data into knowledge.

Templates may be administratively added to HyCoRE at any time. We envision a toolbox of templates made available as HyCoRE matures and allows for dynamic discovery of new templates. The inferenceModel attribute included in the context model of Figure 14 is a reference to a context template, which support provenance as discussed.

Templates are expressed as a graph of reasoning nodes. The message that travels along the edges contains context. Edges imply dependency of a destination node on contextual outputs of source node. Nodes are the work processes for context reasoning. Supported nodes include: search, aggregation, reusable component nodes.

A *search node* represents a local context query for profiled or stored context as distinct from initiating a reasoning process or executing a remote query. Search components have no need for a QM, since locally stored data includes a QM associated with its original derivation sources. The assumption is that no change in reported context quality occurs with the exception of adding edge latency associated with search time. Context Flow designers include search elements for context known to be available locally, and which cannot be inferred or trusted when derived from remote

sources. Search elements add minimal processing overhead when compared with RUCs. An *aggregation node* is used in the reasoning plan to indicate synchronization of context output and aggregation for input to the next node. Incoming edges of an aggregation node indicate possible parallel activity. The *reusable component (RUC)* node is an abstraction for generalized context providers participating in flow reasoning. Context providers may support many types of context. Each RUC is an abstraction for a particular type of context inferred by a reasoner, sensing device, service, or other generalized context provider. In the case of templates, RUC nodes contain on only meta-data and are not bound to a specific provider.

The implicit concurrency, serialization and participant context contribution weight (*not depicted*) are included in the template. A node's context contribution weight represents the percentage of inference effect that node's function has on the overall reasoning plan. Inference effect relates to the correctness and quality of the resulting high level context. Since templates are not bound to providers, the potential quality of context provided by a template depends on its realization as a context flow; where all bound context provider quality is integrated to produce high level context quality. Multiple context templates may exist for inferring the same high level context meta-data. Each offers features (i.e. quality, cost, performance etc.) that are used by HyCoRE's context building function in selecting the most effective template for an inferencing task. Context building will be discussed in Chapter 5. Following are definitions to clarify components used in HyCoRE reasoning as illustrated in Figure 19:

A *graph node* represents any RUC, search or aggregation element. Each node is weighted by its contribution to the resulting high level context. Aggregation nodes have 0 weight. Typically, transformation RUC nodes have little weight. It is the reasoning and source nodes that play the greatest role in the resulting high level context quality. Weights are defined administratively by domain experts when context templates are registered.

A *Reasoner* represents algorithms for reasoning. These may use trained models and/or apply machine learning. A *Transformation Function* is any function/algorithm that filters, extracts

features or transforms information. Both reasoners and transformation functions are abstracted as context providers and represented as RUC nodes in a context flow.

Edges of the context flow represent the transmission of context from a source to a destination node. These imply a serialization of inference events as well. Not seen on the context flow, are implicit incoming edges at start nodes. All latency is initially 0. Context nodes update edge latency values as the difference of arrival and transmission times. The latency on the edges between aggregation nodes and other node is implicitly 0. In a diagram, an aggregation node should never be joined to another aggregation node (it *is redundant*). An edge does not imply synchronous communication, only the order of events is implied. Context result may arrive asynchronously from local or remote sources.

3.4.3.2 Context Flows

Context flows are dynamic reasoning plans instantiated at runtime in response to consumer requests and provider changes. A context flow follows the pattern of a specific context template. The biggest difference between templates and flows is that flows are bound to specific context providers whereas templates are not. The RUC nodes in flows are abstractions of context providers which provide an I/O specification, describing input context requirements and inferred context output. As a concept clarifying supplement to Figure 19, Figure 20 offers an alternative conceptual specification for context templates/flows; where a specific ContextID is inferred using a the sequence of instructions provided there.

Flow Specification: ContextID{ 1..*{ INSTR}}

ContextID: Unique alphanumeric identifier for a context value

INSTR: AGGREGATE, TRANSFORM, COLLECT, SEARCH, CONNECT(parameter list)

AGGREGATE: Combine information essentially a logical AND

TRANSFORM: Apply function to output

COLLECT: Use RUC to sense or infer data

SEARCH: Execute data model or context query

CONNECT: Implies a sequential processing. X CONNECT Y would be graphically represented as two nodes with a directed arrow emanating from X towards Y. It specifies that X must occur before Y.

Figure 20 Context Flow Conceptual Specification

Adaptation is important in maintaining the inferencing plan. No sensor has infinite energy or availability. At some point any physical device will fail due to energy, malfunction, mobility etc. When device failure occurs, inferencing is stalled. We mitigate this and increase the uninterrupted running time for a context flow by repairing the flow using alternative RUC nodes. Also, there are quality requirements associated with context flows. Context providers include sensing devices as well as local / remote reasoning algorithms or services. Quality indications and cost associated with these providers contribute to the integrated high level context QC indicators. As an example, remote nodes have bandwidth-communication and latency costs often greater than local node which may offer low latency with minimal accuracy. The system policies and consumer requirements determine what is acceptable. Reasoning adaptation sustains context while meeting requirements. More details on reasoning adaptation are provided in Section 5.7. For now, this discussion turns to the importance of context quality. The next section discussion information and middleware quality measures as it relates to HyCoRE.

CHAPTER 4

QUALITY OF CONTEXT

Information quality is a general measurement term for information integrity. The term remains ambiguous until its constitution is revealed as a function of observable data measures. One work [32] uses the term ‘quality indicator’ to refer to individual measures that affect quality evaluation. Likewise, the terms *quality indicator* and *quality measure* will be used interchangeably throughout this document.

Establishing high level context quality is not as simple as selecting the device with the best accuracy or other singular quality indication. Increasingly, context-aware applications are interested in information that must be combined from multiple sources, using heterogeneous transformation and reasoning processes. We call this process hybrid high level context reasoning. Establishing true context quality in such environments requires an integrated approach to acquiring heterogeneous quality measures and propagating them through reasoning and transformational processes to produce a composite high level context quality. Many quality measures are only appropriate for describing low level sensor or other context sources. These have a cross effect on resulting high level quality indicators, but many do not translate directly to useful high level context measures. Figure 7 illustrates the dilemma.

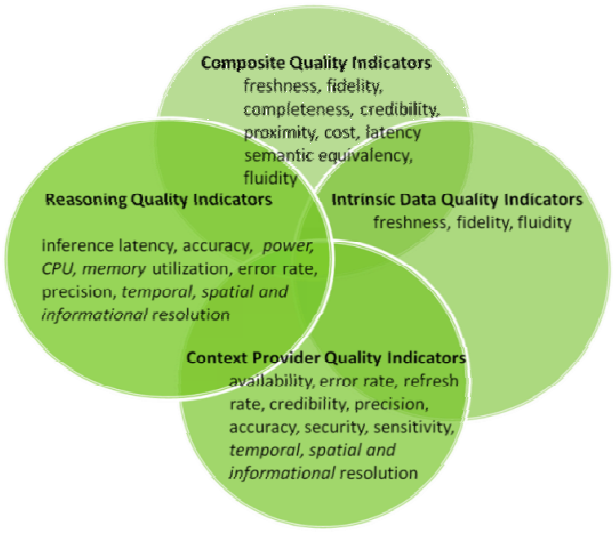

Figure 21 Quality Indicators

Quality of Context (QoC) has been defined as "*any information describing the quality of information that is used as context*."[15]. A related definition is provided for quality of information (QoI) as "*the collective effect of information characteristics (or attributes) that determine the degree by which the information is (or perceived to be) fit-to-use for a purpose*." [9] For this discussion, Quality of Context (QoC) can be defined as a *collection of measures (indicators) reflecting the integrity and discriminative characteristics of information that is used as context*. QoC is one means by which a context middleware accesses the suitability of context and its associated reasoning process for an application. When serving several heterogeneous applications, it is beneficial for the middleware to allow each application to specify quality requirements. An ideal middleware maintains application quality requirements by adapting to environmental changes. At the same time, the middleware may make the most efficient use of available resources. We identified the need for middleware quality measures in [4] and here redefine ***Quality of Context Middleware (QoCS)*** as a *collection of measures (indicators) which reflect the informational integrity and efficacy of a middleware system in meeting the collective context quality requirements of client applications*.

One goal of the HyCoRE framework is adaptation to meet application and system context quality requirements. Which quality measures may be controlled through middleware adaptation depends on the constitution of the context. For example, individual low level context source *credibility* cannot be controlled. However, given multiple sources, the high level context's credibility can be controlled by adapting the participating sources. For the remainder of the document the term *QC indicator* will refer to both context quality and cost measures. The identification of a specific measure as a quality indication is subjective and domain dependent. In this work, we focus on *objective indicators of high level context data integrity or value*.

The following subsections clarify this work's use of existing quality labels and define new quality indicators. We later offer insight for solving quality integration challenges. The terms for semantically equivalent concepts vary, so we offer our perspective. To the best of our knowledge the following quality indicator concepts are first identified in this work. These are: i) information semantic equivalency; ii) context fluidity; iii) inferencing resource efficiency; iv) middleware context effectiveness and v) context middleware operational efficiency. All of the indicators we have chosen can be represented by a numerical value for ease of computation in mathematical equations. Concrete individual representations of indicator values are implementation specific. These reflect the semantics of their associated applications. So herein, we discuss indicators at a conceptual level; only offering concrete examples for illustration. This is not an attempt to identify all quality indicators. No one can anticipate which factors will be important to every application. Any taxonomy of quality indicators, including that given herein, must be extensible.

4.1 Quality Definitions

4.1.1 *Intrinsic Context Data Quality Indicators*

Intrinsic quality indicators reflect the nature of the data. These characteristics are irrespective of the source device and transformation applied to achieve them.

Information Freshness- a relative measure of the recency of information. Creation and update timestamps along with expiration polices may be used in computing this value. Middleware may use this value as criteria for data maintenance, identification and removal of stale values.

Information Fidelity- measures the level of information detail. Similar to the way that numerical units (i.e. feet vs. centimeter, seconds vs. minutes) imply resolution, context fidelity measures the detail exposed by a given value relative to all possible values. In meeting consumer requests, context providers comply with security polices for sharing data. Such security policies may be expressed using context fidelity.

Information Fluidity- measures the rate at which a context value changes with time. Low level facts such as a person's age have low fluidity. Values for high level contexts like activity and location are much higher.

Information Semantic Relevancy/Equivalency- measures the closeness in meaning or semantic distance between two concepts. Disparate context providers use distinct terms for semantically equivalent concepts. This gives an indication of context exchangeability. Semantic equivalency can be used by applications to discriminate available context. Middleware context processing may use these values in adapting reasoning for high level context. In HyCoRE, a consumer request for context is matched on weighted context attributes, categories, locations and targets irrespective of the knowledge models used for derivation. We can infer semantic relevancy by a weighted ratio of matching identifiers values to those requested. If a semantic relevancy policy were configured, it could be used as an optimization, short circuiting exact context matching.

Informational Resolution is a measure of how much information is revealed by a value. This metric can subsume precision, sensitivity, and disclosure level as described in other quality works. The value gives an application insight into the details of the information provided. Also, informational resolution may be used for privacy protection. A middleware may define a system policy threshold on information resolution. Only context information with resolution lower than

that threshold may be shared with consumers. Context providers many further specify a per context informational threshold to limit information sharing.

4.1.2 *Context Source Quality Indicators*

Context sources include: i) devices providing raw data or ii) services that transform data and reason to infer context. Hardware, platform, and environmental fluctuations contribute to the integrity with which context is provided. Useful context source quality indicators include: *availability, error rate, refresh rate and credibility*. We do not redefine these measures, since our interpretation is consistent with existing uses [15],[32],[51],[73]

4.1.3 *Quality of Inference*

The expressiveness and complexity of reasoning techniques vary greatly [1],[2],[7],[26],[27],[31],[38],[53],[57],[63]. Such differences are part of the reason HyCoRE recommends integrating a variety of reasoning techniques to accomplish effective high level context reasoning. Quality of inference indicators are QoC measures reflecting the cost of inferring context. These may be used to discriminate between comparable inferencing methods/algorithms.

Inferencing Latency- a quality measure of timeliness in reasoning for context. Reflects the complexity of algorithms used.

Inferencing Accuracy- a quality measure of how correctly the reasoning process captures situational truth. It reflects the completeness of algorithms used as well as quality of data models (including statistical training data, rules, and ontology specifications)

Inferencing Resource Utilization(power, CPU, memory, network)**-** measures the resource usage of an inferencing technique. This measure reflects the ratio of resources used to resources available on the platform where the reasoning task is deployed. A middleware system may adapt by contracting reasoning from a resource rich remote source when the technique has high resource utilization locally but low utilization remotely. Note that network utilization reflects bandwidth usage in acquiring contributing data from remote stores or other sources.

Inferencing Resource Efficiency(power, CPU, memory, network)**-** relative measure of reasoner ability to minimize resource utilization.

4.1.4 *High Level Context Quality Indicators*

High level context quality indicators are the result of quality aggregation and propagation. These are of concern to context consumers and can be used to express context requirements.

Context Accuracy- a quality measure of how correctly a context value reflects the situational truth at a moment in time. It is affected by error rate of the raw sensing device, as well as the error potential of the inference process.

Context Certainty- a quality measure of confidence in inferred value.

Context Latency- a quality measure of the time required to retrieve and return context information to the requestor. Context latency may include times for sensing, transformation and reasoning, data store access and end- to-end communication.

Context Completeness- measures the totality of the information set used in deriving a specific context value. Missing and expired information would reduce completeness. It reaches maximum value when all relevant context contributory data is available with the required freshness. Direct information from profiles or sensors have an implicit completeness of 1. This value is not related to the number of queries satisfied for an application. See Section 4.2 for quantification discussion.

Context Freshness- *same as previous definition given in 4.1.1*. When data is used directly from the sensing source this value is the same as that defined for intrinsic data characteristics.

Context Credibility- a measure of trust in the correctness of information; a measure of belief as distinct from accuracy. High level context credibility can be affected by data freshness as well as contributing context source credibility. A context consumer may interpret this value as an indication of trust.

Context Proximity- an indicator of spatial relation to a specified location. This value is affected by location of data sources.

Context Cost- a multidimensional indicator of context liability. In general, it reflects inferencing resource utilization and end-to-end communication costs. CPU, memory, energy, and currency are all dimensions of cost. It is necessary to measure integrated cost since the middleware platform has physical limitations. To illustrate: Context flows using local sensors could incur significant energy costs (i.e. camera, mic, gps, accel, light etc.); ontological reasoning on large knowledge base would significantly impact CPU and memory cost; in a dynamic environment where providers offer context for a price, currency is a context cost consideration. See Section 4.2 for quantification discussion.

Context Fluidity- rate of information change over time with respect to all contributing factors. Fluidity provides an indication of data temporality, a measure of information stability over time. Low fluidity indicates a value that is not subject to change beyond set time and value ranges. Fluidity may drive how often a consumer needs to acquire new context values prior to decision making. Alternatively, different knowledge representations of the same context are distinguished by fluidity, so then middleware and applications may discriminate among choices base this indicator. Context provider quality measures such as refresh rate, sampling rate have a direct effect on fluidity. See Section 4.2 for quantification discussion.

Context Fidelity- *similar to previous definition given for information fidelity in 4.1.1*.

Context Informational Resolution- is measure of how much information is revealed by a value. This metric subsumes precision, sensitivity, and disclosure level as described in other quality works. The value gives insight into the detail of the information provided. Also, informational resolution may be used for privacy protection. For example, only context information with resolution lower than a specified threshold may be shared with consumers. Context providers many further specify a per context informational threshold to limit information sharing. See Section 4.2 for quantification discussion.

Context Semantic Equivalency- is the measure of the semantic distance between two concepts. See Section 4.2 for quantification discussion.

These indicators also apply to low level contexts. There are instances where context consumers use low level contexts direct from the sensing source. Examples are: user profile values or simple environment measurements such as temperature. Quantification for some of the high level context measures above is discussed Section 4.2.

4.1.5 *Context Middleware Quality Indicators (QoCS)*

We offer the following middleware quality indicators. A later section discusses how these can be quantified.

Middleware Context Effectiveness- a cumulative measure of the system's performance in meeting context requirements for all applications at an instant in time.

Middleware Context Cost- a measure of liability associated with context acquisition. Cost reflects the cumulative effect of processing, reasoning, and communication cost indicators. The concept of middleware context cost may directly correlate to resource utilization.

Middleware Cost Effectiveness- measures the platform's ability to stay within policy defined cost bounds for all cost indicators.

Middleware Quality Effectiveness- measures the system's performance in remaining within policy defined quality thresholds

Middleware Operational Efficiency- a measure of efficient context recognition. It reflects the ability of the middleware system to adapt in a way that meets requirements while minimizing cost.

Meantime Before Context Failure-- an average running time before failure or adaptation for all flows that have been running since system startup.

Reasoning Stability Measure- a cumulative measure of uninterrupted context reasoning.

Quantification for QoCS middleware measures given above is discussed in 4.4.

4.2 High Level Context Quality Quantification

The previous section offered definitions for context quality measures. We offer quantification details for some of those measures in this section.

Cost of Context as defined in HyCoRE is given by:

$$CCost_i = \sum_{j=1}^{n} (Node_j(\ Cost_i\ ) \tag{1}$$

where *i* ∈(Context Indicators(*cpu,memory,energy,bandwidth,currency*..) and *j* ∈(context reasoning node list of flow representing context *i*).

Context Completeness is given by:

$$Completeness = f(contextInputsNeeded, missingOldInputs)$$
$$Completeness_{approx} = \frac{validContextFlowInputs}{requiredContextFlowInputs} \tag{2}$$

Context Freshness measure of information age. Unexpired old and new information has value. It is up to the consumer to specify its preference of age.

$$Q_{freshness} \frac{ContextAge}{ContextLifeSpan} \tag{3}$$

Context Fluidity -Suppose that the context update rate *(ur*) forms a continuous curve over time(*t*). Let, function (*f*) be determined by interpolation , then fluidity is given by:

$$ur(t) = f(t), \qquad Fluidity = f'(t)$$

Alternatively fluidity can be approximated simply as the information update rate.

$$Fluidity\ (approx) = \frac{numberOfUpdates(t)}{t} \tag{4}$$

Again, we must presume there exists a function that would map fluidity to the range [0,1].

Context Informational Resolution is given by:

$$Information\ Resolution = \frac{currentInformationValue}{maxInformationValue} \tag{5}$$

Context Semantic Relevancy (Equivalency) is given by:

$$Semantic\ Relevancy = \frac{descriptionElementsMatched}{descriptionElementsRequired} \tag{6}$$

The succeeding sections will discuss techniques for deriving integrated QC indicators, linking QC indications to middleware performance, and adaptation based on evaluation of QC indications against requirements.

4.3 Quality Integration

Quality aggregation occurs when the quality indicators of multiple context inputs must be combined to produce composite indicators of data integrity. A search component may yield multiple evidences that need to be aggregated. Also, the nodes of a context flow may require several inputs. Aggregation of these inputs must be performed to achieve a combined qualitative effect. With reference to Figure 19, aggregation is a vertical integration of inputs at any node. Quality propagation refers to the process of creation and transformation of quality indicators from raw data acquisition though all stages of reasoning. Propagation produces composite quality indicators appropriate for measuring high level context data integrity. With reference to Figure 19, propagation is a horizontal integration of inputs at any node. Aggregation or propagation of context may increase or reduce resulting high level context quality. As the middleware system infers context, it incurs quality aggregation and propagation challenges. This is especially true for hybrid high level context reasoning.

Figure 22 Quality Aggregation

Figure 23 Quality Propagation

We have identified the following propagation challenges:

- *Accurately reflecting the reasoning transformation process in resulting inferred context quality indicator*-In the HyCoRE extension presented later, we partially address the issue by offering a way to declaratively reveal the inferencing transformational pattern. We refer to the patterns as context flows. Context flows are inference specifications of high level context. However, context flows only partially expose reasoning. Reusable reasoning components (RUC) in the context flow serve as black boxes for reasoning. Runtime computation of composite quality indicators does not present a problem in this design, since RUCs bear the responsibility for computation. Accurate prediction of resulting quality is what proves difficult. In order to fully capture the transformation process, heuristic functions of quality indicators would need to be expressed for every RUC. An inference specification containing context flows with quality indicator transformation heuristics can be used to predict context quality. Such predictions are useful to reasoning adaptation.
- *Combining heterogeneous context contributions*- Heterogeneous contexts are combined to form complex high level context inference. The challenge is to reflect the relative significance of each contribution to the resulting inferred context quality indicators. One solution is to declaratively specify this significance by weighting the contributions of context flow RUCs mentioned previously.

Context aggregation issues [51][69][73] have previously been identified. We offer suggestions for meeting these challenges:

- *Coping with missing context or quality indicators*- In dynamic environments, sources of context may be unavailable, stale or too costly to infer at an instant in time. In some cases,

it is appropriate to bypass unavailable values, using some representation interpreted as unknown. Otherwise missing values can be approximated using quality history or default policies. The context completeness quality indicator can be used as a measure of missing data.

- *Handling redundant context*- some works have applied simple approaches to data aggregation from homogenous sources. Examples include min, max, and average functions applied to the context set. Dempster Shaefer Evidence Theory (DSET)[57] could be applied for deriving composite quality indication. In this case, the resulting composite quality indicators would be potentially greater than any individual contribution. The important issue is to accurately represent the added value of additional evidences.
- *Mitigating Conflicting data*- in [51] the authors use quality indicators such as accuracy, credibility and freshness to prioritize data importance. Alternatively, DSET also allows for representing contradictory evidences in a manner consistent with the significance of the evidence.

4.3.1 *Aggregation and Propagation Strategies*

To compute an integrated quality model, the problem we need to solve is identifying the appropriate functions for aggregation and propagation. The correct aggregation and propagation function can be a matter of debate. The quality indicator in question, application and middleware goals all contribute to the appropriateness of an approach. Also, the technique used for aggregation may necessarily differ from propagation. For example, a SUM function is appropriate for propagating latency, but MAX is better for aggregation. AVERAGE function might be used for aggregating accuracy, but a PRODUCT function when propagating. In HyCoRE, these QC integration functions are implemented as middleware policy dependent interchangeable strategies. Several approaches to quality data integration have been used in other works. These are described below:

Product- computes the quality product of participating sources;

Sum- computes the quality sum of participating sources;

Average- computes the quality average of participating sources; and

Maximum, Minimum – determine the maximum or minimum quality from participating sources.

This pluggable architecture facilitates dynamic configuration of integration strategies. The reasoning process considers current system policy when calculating the integrated context quality model (AWM). For the aforementioned evaluation scenario, the QC indicator strategies shown in Table 4 are applied:

Table 4 Quality Integration Strategies

QC Indicator	**Aggregation**	**Propagation**
Accuracy	Average	Product
Certainty	Minimum	Product
Credibility	Average	Average
Freshness	Minimum	Minimum
Informational Resolution	Maximum	Maximum
Latency	Maximum	Sum
CPU Cost	Sum	Sum
Memory Cost	Sum	Sum
Energy Cost	Sum	Sum
Bandwidth Cost	Sum	Sum
Currency Cost	Sum	Sum

Figure 10 illustrates the quality integration process beginning with outputs from two transformation functions. In the first step, the QC indicators for inputs must be combined to form an aggregated quality model (*aqm*). Next, the learned or acquired QC indicators of a context provider used for reasoning must be integrated with the *aqm*, resulting in a propagated quality model (*pqm*). Only, the QC indicators that are directly affected by the quality of intermediate inferencing nodes participate in the *pqm* step. Of the indicators mentioned in table 1, only *accuracy*, *credibility*, and *certainty* are affected. *Latency* is measured in real-time. Finally, the

runtime actuator inference QC indicators are combined with the *pqm* to yield the composite high level context quality. When the strategies specified in table 1 are used, the resulting high level context QC indicator vector is: [*240, .88, .95,.78, .94,1, .1, .45, 40 , 1,7, 23.8 ,0], where QC labels are [Latency (ms), Accuracy, Credibility, Certainty, Freshness, InformationalResolution, RemainingEnergy, CpuCost, MemoryCost(bytes), EnergyCost, BandwidthCost(bytes), and CurrencyCost].*

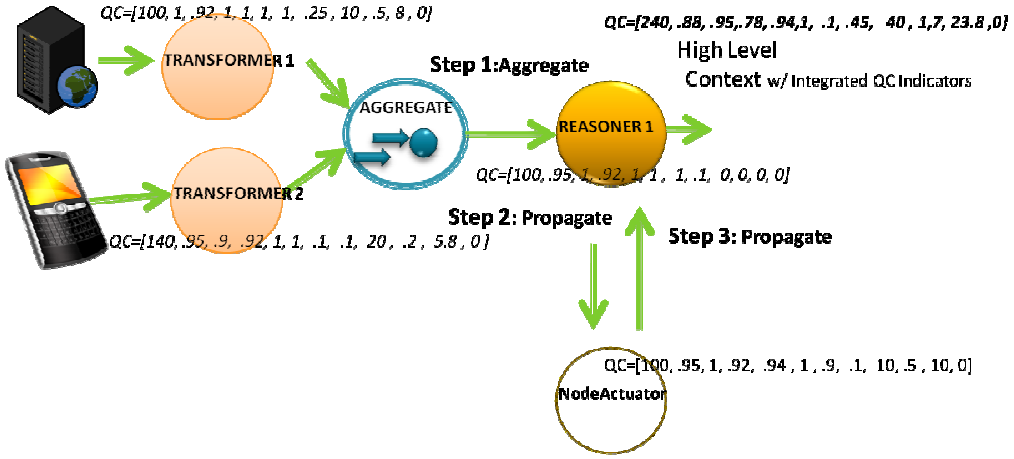

Figure 24 Quality Integration Process

4.4 Context Middleware Quality Quantification (QoCS)

Context middleware should rate itself primarily on its ability to meet consumer context requirements.

Context middleware effectiveness is achieved by adapting the use of context resources (especially reasoning) to maximize the cumulative context effectiveness for all applications while minimizing system cost. Here, we propose context middleware system quality measures that appropriately aggregate application context requirements, reasoning performance and system cost. Of course our pre-supposition is that middleware has its own admission control policy and only rates itself relative to admitted consumers. Additionally, the middleware platform has both soft and hard resource limitations. Efficient usage of platform resources while remaining below hard resource limits is important. Another issue involves efficient use of available providers of context. Depending on the application domain in which HyCoRE is deployed, quality indicator and context cost limits define what is considered efficient for the system domain. In summary, platform resource limitations, middleware context cost and quality policies can be used to efficiently adapt context reasoning.

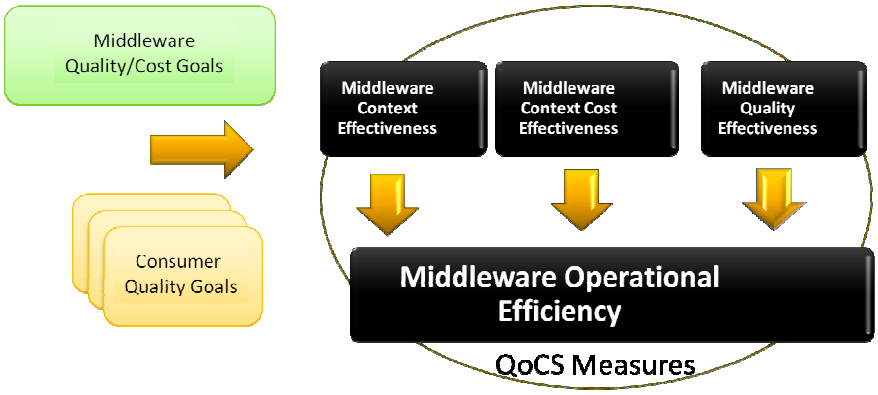

Figure 25 HyCoRE Performance Criteria

We offer the following methods to quantify for middleware quality indicators previously discussed.

4.4.1 *Middleware Context Effectiveness*

Context effectiveness (*Ceff*) is a cumulative measure of the system's performance in meeting context requirements for all applications at an instant in time. To derive Ceff, we must define a few intermediate measures.

Quality quotient (Qq) refers to the system's ability to meet individual context quality goals of an application. It is a measure of quality with regards to single context required by a consumer at an instant in time. Quality quotient is given by:

$$Qq_i(t) = \frac{Actual\ Quality\ Value_i}{Quality\ Value\ Goal_i} \tag{7}$$

$$Qq_i(t) = \frac{Quality\ Value\ Goal_i}{Actual\ Quality\ Value_i} \tag{7.1}$$

The actual context quality value is measured in real-time, while the quality goal is given as part of the consumer requirement specification[6]. Some quality indicator ratios such as latency should be interpreted differently from others like accuracy. For example: the resulting value of .885 /.96 for accuracy appropriately reflects good performance with a higher value. However, 3ms/10ms for latency would cause one to believe that the quality is too low. A small ratio is good for latency, but this is not the case with accuracy. So that the relative meaning of all ratios in the system is the same, we recommend equation 2.1 for certain measures, especially those with time units. A higher ratio indicates that the system is performing well, relative to 100%. So, in the case of latency, 10ms/3ms appropriately indicates good performance.

The individual context effectiveness (Ice(t)) for a given application/consumer represents the system performance in reasoning for a single context within the required quality thresholds. It is measured as a function of Qq for all QC indicators specified. We express this as:

[6] The consumer requirement specification allows the consumer to specify a weighted list of context requirements. Additionally, for each context, consumer specify QC indicator thresholds and priorities.

$$ICe(t) = \sum_{i=1}^{i=n} Wq_i * \min(1, Qq_i(t)) \quad (8)$$

Where 0 ≤ *ICe(t)* ≤ 1 , i represents a QC indicator index, and Wq_i is the weight/priority given to that QC indicator(i) in the consumer requirement specification for a particular context.

Application context effectiveness is a measure of the middleware ability to meet all of an application's context needs. We express the context effectiveness with respect to a single application/consumer as:

$$ACe(t) = \sum_{i=1}^{i=n} Wc_i * ICe_i(t) \quad (9)$$

Where 0 ≤ *ACe(t)* ≤ 1 and Wc_i is context weight as specified in the consumer requirement specification.

Now that we have all the intermediate measures, we express cumulative context effectiveness (Ceff) as:

$$CEff(t) = \frac{\sum_{i=1}^{i=n} ACe(t)}{n} \quad (10)$$

Where 0 ≤ *CEff(t)* ≤ 1, and n represents the number of context consumers currently being served. In the ideal case context effectiveness remains 1 to indicate that all consumer requirements are met.

4.4.2 *Middleware Quality Requirement Policy (Middleware Quality Priorities and Middleware Quality Effectiveness)*

The middleware quality policy specifies a weighted list of QC indicator concerns. The system adapts according to these priorities. As an example: If the *energyCost* QC indicator is specified as the only system quality concern; the system will always select the most energy efficient of qualified providers for inferencing, rather than the first available.

The middleware quality policy also specifies the minimum quality indicator value that a provider must meet to participate in inferencing. The middleware quality effectiveness (MQe) measures

the system's performance in remaining within policy defined quality thresholds. MQe policy is specified as a weighted list of quality indicator thresholds. MQe is given as a function of quality indicator effectiveness (QIE). QIE measures the systems performance in remaining within a quality indicator policy thresholds. MQe and QIE are given by:

$$QIE_i(t) = \frac{Worst\ Actual\ Quality\ Indicator\ Value_i}{Middleware\ Quality\ Indicator\ Threshold_i} \tag{11}$$

$$QIE_i(t) = \frac{Worst\ Actual\ Quality\ Indicator\ Value_i}{Middleware\ Quality\ Indicator\ Threshold_i} \tag{11.1}$$

$$MQe(t) = \sum_{i=1}^{i=n} WQ_i * \min(1, QIE_i(t)) \tag{12}$$

Where $0 \leq WQ_i \leq 1$, represents the relative priority of the QC indicator to the overall middleware quality effectiveness as set in current middleware policy. Similar to the way the quality quotient (Qqi) is defined, equation 6.1 is given as alternative representation of the QIE ratio. See the quality quotient discussion in section 3.8.1 for rationale. Also, $Middle\ Quality\ Indicator\ Threshold$ is a system policy setting.

If any provider is below the quality threshold for a given indicator that has been identified as a concern for the system, middleware quality effectiveness is reduced. Continued use of such providers failing to meet middleware quality thresholds, degrades the overall middleware quality, which further degrades middleware operational efficiency discussed in section 4.4.4. If there are no quality priories set, the *MQe* will always be 1.

4.4.3 *Middleware Context Cost*

Middleware system context cost is a measure of liability associated with context acquisition. At any instant of time, cost represents the cumulative effect of processing, reasoning, and communication cost indicators. The concept of middleware context cost may directly correlate to resource utilization. Middleware context cost is more efficient when context is shared. Notice in

the equation that the cost of context from a single provider is only counted once, even if many consumer flows are using it. Logically, middleware cost is calculated as follows:

$$MCost(Ci,t) = \sum_{j=1}^{m}\sum_{k=1}^{n}(Flow_j[Node_k(Cost_i)]) - \sum_{l=1}^{p}\sum_{m=1}^{r-1}(Provider_l(Context_m(Cost_i) * Context_m(numReasoningNodeRefs), \quad (13)$$

where *m* represents the number of active context flows, *n* represent the number of nodes in a flow; *p* represent the number of active providers; *r* represents the number of contexts each provider has; *Ci* ∈(*cpu,memory,energy,bandwidth,currency*…) and *numReasoningNodeRefs* reflect the number of nodes sharing $node_k(context_i)$. Refer Section 4.4.3 for discussion on deriving $CCost_i$.

Operationally, the bulk of middle context cost calculation is performed at run time in-line with context flow inferencing.

4.4.4 *Middleware Cost Effectiveness*

Middleware cost effectiveness (MCe) measures the platform's ability to stay within policy defined cost bounds for all cost indicators. The middleware cost policy is a weighted list of cost indicator thresholds. These reflect the hard or soft limitations imposed on the reasoning middleware. Cost Indicator effectiveness (CIE) is a sub-measure of MCe. It reflects the system's platform's ability to stay within policy defined cost bounds for a single cost indicator. CIE and MCe are given by:

$$CIE_i(t) = \frac{MCost_i}{Middleware\ Cost\ Indicator\ Threshold_i} \quad (14)$$

$$CIE_i(t) = \frac{Middleware\ Cost\ Indicator\ Threshold_i}{MCost_i} \quad (14.1)$$

$$MCe(t) = \sum_{i=1}^{i=n} WC_i * \min(1, CIE_i(t)) \quad (15)$$

Where $Middle\ Cost\ Indicator\ Threshold$ is a system policy setting.

4.4.5 *Middleware Operational Efficiency*

Operational efficiency(OEff) is a measure of efficient, effective context recognition. It reflects the ability of the middleware system to adapt in a way that meets both consumer and system quality/cost requirements. OEff is given by:

$$OEff(t) = Ceff(t) * MCe(t) * MQe(t) \quad (16)$$

4.4.6 *Mean Time before Context Failure*

The meantime before context failure is an average running time before failure or adaptation for all flows that have been running since system startup. We use this statistic to compare the effects of reasoning adaptation in different HyCoRE system configurations. MTBCF is given by:

$$MTBCF(t) = \frac{\sum_{i=1}^{f}(MTBCF_i(t))}{f} \quad (17)$$

$$MTBCF = \frac{\sum_{i=1}^{t}(MTBCF_t)}{t} \quad (17.1)$$

Where *f* represents the number of context flows and t represents the number of instantaneous MTBCF calculations. So, then MTBCF reflects the uninterrupted context flow inference time average over the *system* running time.

4.4.7 *Reasoning Stability Measure*

Reason stability is an average of uninterrupted context reasoning times for all contexts being actively inferred. The reasoning stability measure reflects the percentage of time inferencing is uninterrupted. Context flows (reasoning plans) are active when all components are ready, but temporarily set as inactive when a failure occurs. Periods of time between a flow becoming inactive and being repaired and reactivated reduce the reasoning stability. Context flow stability measure(FSM) reflects the percentage of uninterrupted reasoning for specific reasoning plan.

$$FSM(t) = min\left(1, \frac{TotalInactiveTime}{FlowLifeTime}\right) \tag{18}$$

$$RSM(t) = \frac{\sum_{i=1}^{n}\left(\mathrm{FSM}_i(\mathrm{t})\right)}{n} \tag{19}$$

If there are no flows in the system but there are active consumers, the reasoning stability measures is 0. If there are no flows in the system and there are no active consumers, the reasoning stability measure is 1.

4.4.8 *Summary System Measures and Symbols*

The QoCS measures that have been presented assume independent context provider behavior. Collaborative failure of context providers would cause operational efficiency to drop precipitously. The system would continuously attempt to adapt. Unfortunately in the case of collaborative value, the alternative devices have also failed, so that HyCoRE will not adapt successfully until after this anomalous failure period is over.

With regard to middleware cost effectiveness, we envision deriving a measure that truly reflects the cost efficiency of inferencing. However, we must first derive a way to estimate minimum cost for a context given the available providers. This minimum possible cost can be compared to current system cost as an indication of how well the system is keeping costs low. For now, the cost effectiveness calculation uses middleware cost indicator threshold policies.

Table 1 summarizes the QoCS performance measures defined previously.

Table 5 Middleware Quality of Context Measures (QoCS)

Equation Number	***QoCs Measure Name***		**Definition**
7	$Qq_i(t)$	Quality Quotient	Measures system's ability to meet individual context quality goals of an application.
8	*ICe(t)*	Individual Context Effectiveness	Measures the system performance in reasoning for a single context within the required quality thresholds.
9	$ACe_i(t)$	Application Context Effectiveness	Measure of the middleware ability to meet all of an application's context needs.
10	*Ceff(t)*	Middleware Context Effectiveness	Cumulative measure of the system's performance in meeting context requirements for all applications at an instant in time.
11	$QIE_i(t)$	Quality Indicator Effectiveness	QIE measures the systems performance in remaining within quality indicator policy threshold.
12	*MQe(t)*	Middleware Quality Effectiveness	Measures the system's performance in remaining within policy defined quality thresholds.
13	$MCost(C_i, t)$	Middleware Context Cost	Measures liability associated with context acquisition. Cost represents the cumulative effect of processing, reasoning, and communication cost indicators.
14	$CIE_i(t)$	Cost Indicator Effectiveness	Measures the platform's ability to stay within policy defined cost bounds for a single cost indicators.
15	*MCe(t)*	Middleware Cost Effectiveness	Measures the platform's ability to stay within policy defined cost bounds for all cost indicators.
16	*OEff(t)*	Middleware Operational Efficiency	Operational efficiency is a measure of efficient, effective context recognition
17	*MTBCF*	Mean Time Before Context Failure	An average running time before failure or adaptation for all flows that have been running since system startup.
18	$FSM_i(t)$	Flow Stability Measure	Reflects the percentage of uninterrupted reasoning for specific reasoning plan.
19	*RSM(t)*	Middleware Reasoning Stability Measure	Average of uninterrupted context reasoning time for all contexts being actively inferred.

CHAPTER 5

ADAPTABLE CONTEXT REASONING WITH HYCORE

The context reasoning data model was introduced in section 3.4.3. In this section we describe context reasoning as the process of inferring knowledge. The reasoning subsystem is at the heart of HyCoRE's adaptable context reasoning engine. Meta data matching is a function that occurs several times in HyCoRE processing. It involves matching on the five potential meta data criteria: i) context category, ii) targets, iii)semantic model, iv) physical model; and v)QC indications. Figure 26 highlights the collaborations of reasoning subsystem components.

The follow sections define each component and their interactions in details.

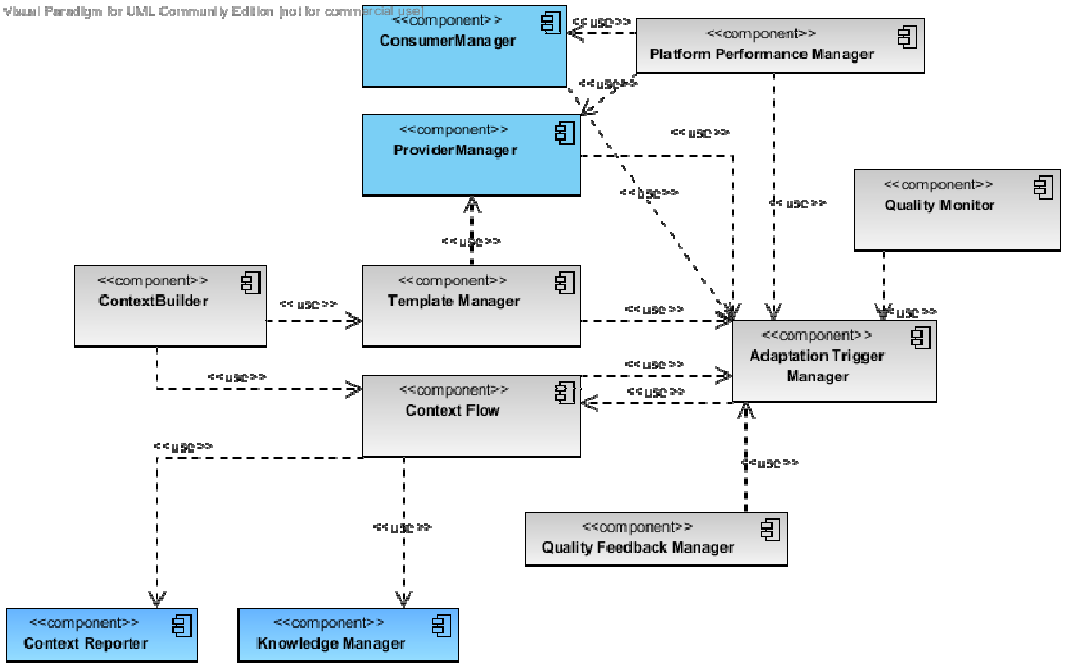

Figure 26 Context Reasoning Subsystem Components

5.1 Template Manager Functions

To initiate even the simplest reasoning, there must exist a patterns for inferring context. Domain knowledge of inferring context is capture in the form of context templates. As introduced in section 3.4.3, the context template is a generalized reasoning plan, not bound to a specific context provider. The context template manager component handles context template registrations and well as events that affect potential realizations of those patterns. A context template can have a number of possible instantiations. Each possible instantiation is a context projection. A context projection is a specific instance of a reasoning plan, using selected context providers; reflecting the composite quality of participating providers. The template manager configures and maintains projections of potential context flows. Projection finding is triggered by provider or template changes including: i) New Template Registration; ii) New provider registration and iii) Provider availability change or failure. Since the template manager maintains dynamic context projections, HyCoRE has alternative reasoning plans readily available for instantiation. When new projections are identified by the template manager, a context adaption trigger is created. The new projection trigger handler adapt reasoning by: i) attempting to initiate context reasoning for consumers not assigned to a context flow. Perhaps because there were no providers or projections existing at the time; and ii) Attempting to replace suspended flows experiencing temporary failure with an alternative projection. Note that a separate context quality manager component performs periodic optimization of existing context flows. Also, reasoning adaptation due to failure to meet requirements is handled at runtime. Run-time adaptation events are captured by the Adaptation Trigger Manager which is discussed later.

The process for deriving the possible projections of a template is as follows:

1. Receive new template
2. Create an array mapping of providers'→ references rences for each RUC node context meta-data description given in the template. Below is a sample provider mapping for a

template (c1→c2→c3→High Level Context A) that calls for 3 RUCs producing 3 distinct contexts :

Table 6 Template Manager Provider Mapping Example

Context1	Context 2	Context 3
Provider 4	Provider 1	Provider 2
Provider 5		Provider 3

a. In the above illustrations context providers are matched to the meta-data described in the template such that the possible template projections (omitting search and aggregation nodes) are:
b. RUC Node(provider 4) → RUC Node(provider 1) → RUC Node (Provider 2) → High Level Context A
c. RUC Node(provider 5) →RUC Node(provider 1) → RUC Node (Provider 2) → High Level Context A
d. RUC Node(provider 4) →RUC Node(provider 1) → RUC Node (Provider 3) → High Level Context A
e. RUC Node(provider 5) →RUC Node(provider 1) →RUC Node (Provider 3) → High Level Context A

So, there are four possible ways to realize the template in this example. The template manager is only responsible for maintaining the projections. The context builder, discussed in the next section, is responsible for projection selection.

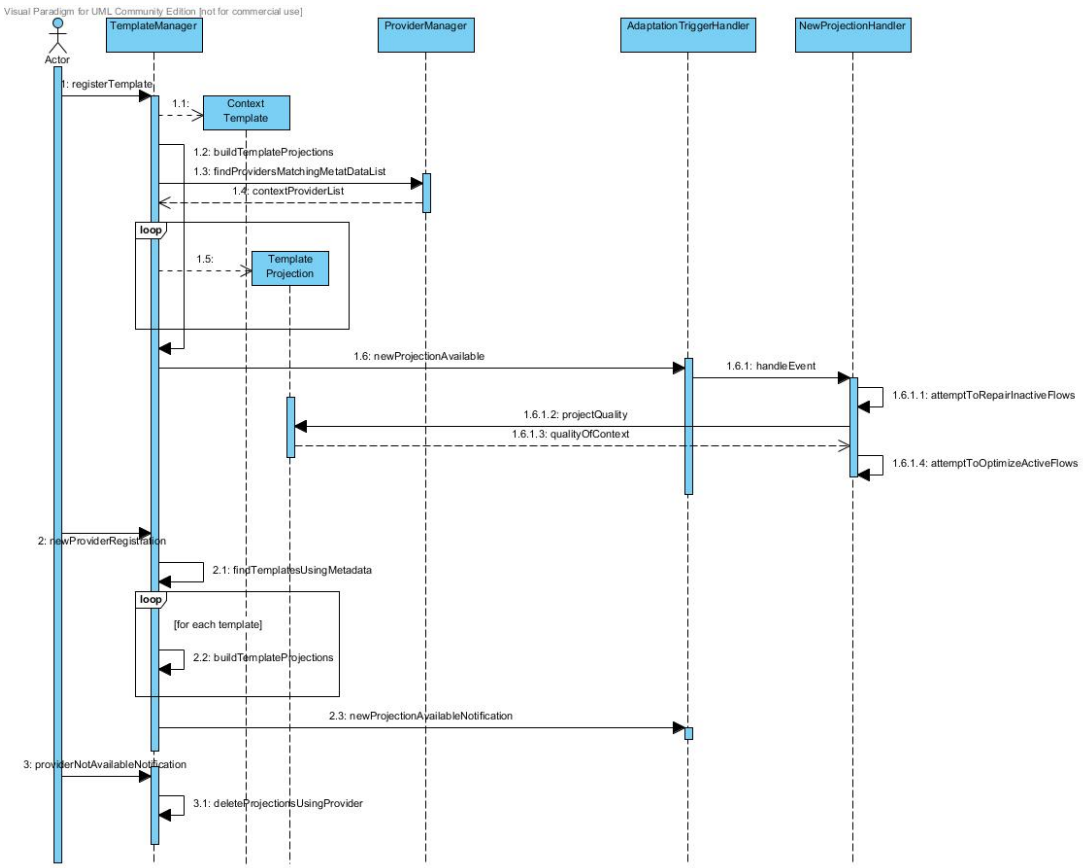

Figure 27 Template Manager Sequence Diagram

5.2 Context Builder Functions

The context builder instantiates context templates as context flows to serve consumer requests based on available template projections. The context builder picks the best fit available projection for current context inferencing tasks. Projections are selected based on their closeness to matching consumer requirements. By selecting a projection, the collective effect of all participating providers is considered. The logic for metadata matching is the domain of the context builder. So, in other situations where a flow needs be repaired by replacing a single node, context builder utilities are called upon. These run-time adaptation events are captured by the Adaptation Trigger Manager which is discussed later. The context building process is as follows:

1. Query Template Manager for templates with meta-data matching the consumer request.
2. Select set of projections based on consumer meta-data match and builder selection policy. Builder selection policies are provided as a way to optimize system performance. Policies include:
    a. *Best of first X* available projections policy, where X is a limited number of projections to consider. This can be beneficial where many projections are anticipated and real-time analysis of each would be unreasonable;
    b. *Shortest* projection policy causes the builder to select the meta-matched projection with the least number of nodes;
    c. *Maximum of X* nodes policy causes the builder to select meta-matched projections with no more than X nodes; and
    d. *All* projections policy causes the builder to select all meta-matched projections for consideration.
3. Score selected projections: If more than one meta-matched projection was selected, the best match is determined by computing a context effectiveness score. The projection with the highest effectiveness score is selected for instantiation. Scoring is based on

consumer requirements, projected provider QoC, and system requirements. Where projection scores are equal, a discrimination factor is used to distinguish them. The projection with the smaller DF has lower cost and is a more efficient choice.

4. Instantiate best projection. Even if none of the selected projections meet requirements, it is used since HyCoRE makes a best effort in meeting requirements. Run-time adaptation will fix the situation should new providers become available and /or provide QC indications improve. In HyCoRE, context flows represent the process of knowledge inference for a single context element with regard to a specific consumer. Consumers don't not share context flows, but they may share providers through the use of RUC reasoning node abstractions.
5. Register context flow with Flow Manager.
6. Start context flow for active inferencing.

In the cases where there are no available projections or where the context builder projection selection policy is configured to quickly instantiate a potentially sub-optimal reasoning plan, periodic quality evaluation will cause the reasoning to adapt to the best available.

In summary, the context builder is primarily invoked when a consumer request arrives and new context flow needs to be instantiated. Other uses include repairing a flow when a participant node fails or falls below quality thresholds and needs to be replaced and replacing a context flow when it falls below consumer or system quality requirements.

5.3 Context Provider Registration and Context Update Processing

As previously discussed, context providers are abstraction for any source of information. The context provider service contract details provider actuation. This design supports synchronous and asynchronous pull receipt of provider information. HyCoRE maintains a context buffer which allows for data staging at context updates are received.

5.4 Context Consumer Registration with Context Mailbox

When registering with HyCoRE, context consumers specify how context is to be returned. Mobile users are challenged to maintain server connections or listening ports for receiving context. So, HyCoRE offers the context mailbox as a server-side storage of context reports. Consumers are able to check their context mail at earliest availability. Context consumer specify context mail in their service contract at the method by which context is to be returned. Below we illustrate the consumer registration process along with context mailbox:

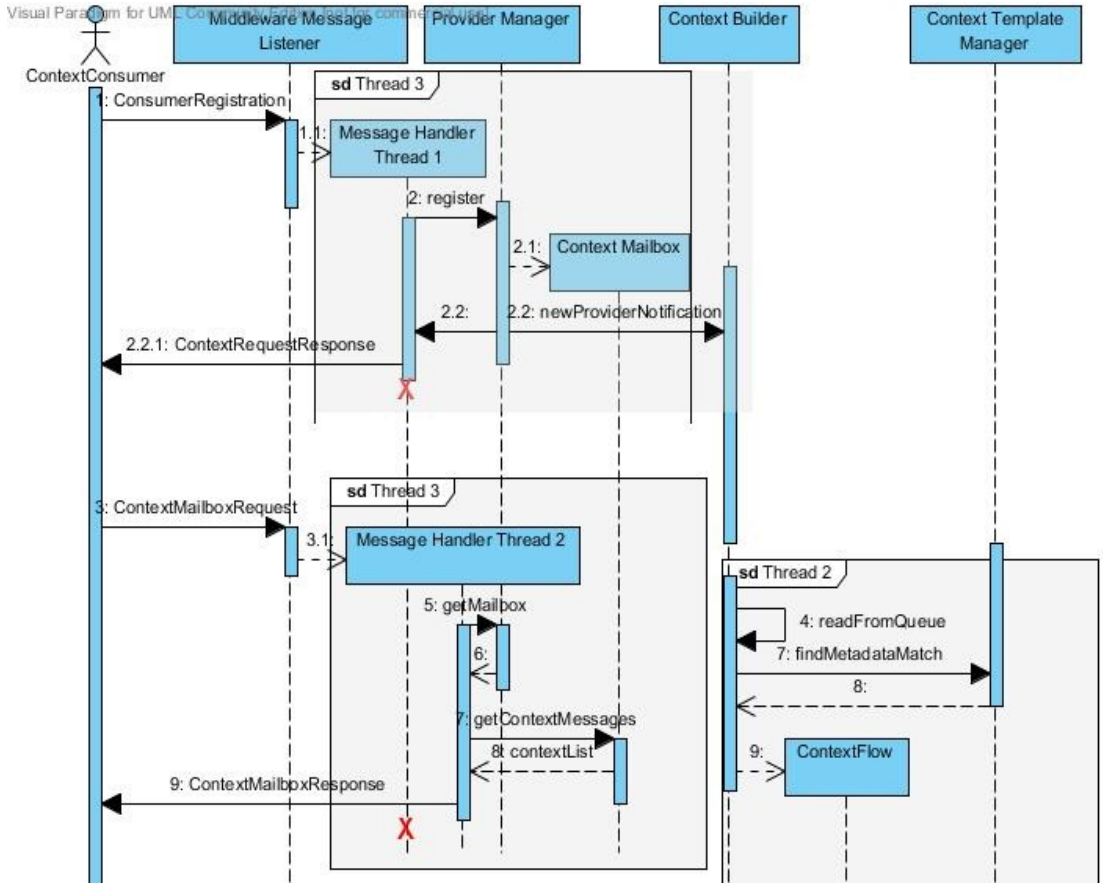

Figure 28 Consumer Registration w/mailbox Sequence Diagram

5.5 Context Flow Reasoning with Quality Integration and Provenance

Reasoning plan projections are realized and executed as context flows. The projections instantiated by the context builder represented as reasoning stream of serialized nodes (refer back to Figure 18). To illustrate, the context flow depicted in Figure 29 represents a context flow for inferencing a person's high level activity from the low level inputs of calendar information, accelerometer readings and GPS positioning.

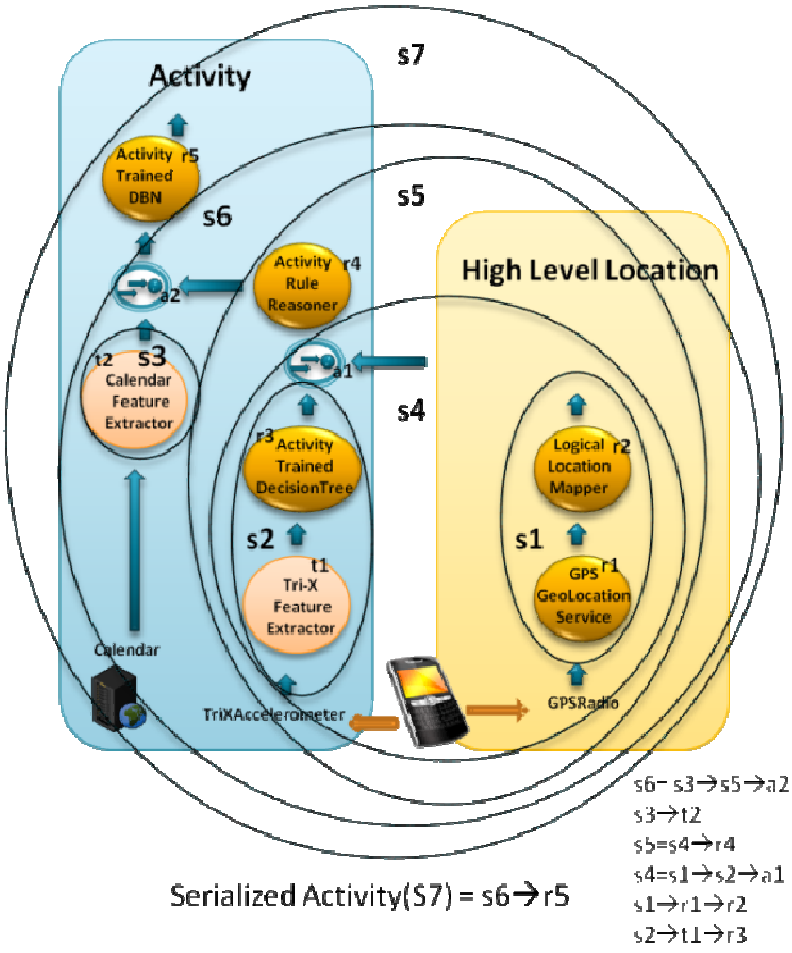

Figure 29 Serialized Complex Context Flow

The label s7 represents the entire reasoning stream serialized as a linked list of nodes where the nodes represent RUC nodes (‘*t*’ for transformations, ‘*r*’ for reasoners), aggregation points (‘*a*’), or other reasoning streams. This illustration is left abstract purposefully to highlight the concept or HyCoRE reasoning. The labels used in the diagram are as follows: S7 infers a high level activity such as (‘in transit’, ‘shopping’, ‘in a meeting’, ‘working out’) and is composed of several sub-streams (s1-s6). R1 could be any geo location service that translates a GPS position to the nearest postal address. R2 represents an application specific mapping of a user’s GPS position to a logical locations (i.e. school, work, gym). T1 represents a generalized utility for

extracting features from accelerometer readings (i.e. XY, XZ, ZY correlation, standard deviation, mean … etc.). The context of each node is published in their corresponding service contract. There a two aggregation nodes in the figures (a1 and a2). Aggregation nodes combine stream outputs for form aggregated input for the next node. The rest of the figure is self-explanatory.

Following are a few rules regarding HyCoRE reasoning: The reasoning w/quality integration algorithm assumes the context flow has been serialized as a linked list of reasoning stream nodes, and the following rules apply: i) A context flow may have several context input items. However a reasoning stream has only one input and represents a single threaded reasoning path; ii) Streams may terminate with an aggregation node, but may not contain any intermediate aggregation nodes; iii) several streams may terminate with the same aggregation node reference. For streams terminating the with same aggregation node, the order of execution is inconsequential. In fact they can be made concurrent; iv) All potential starting nodes of concurrent processing must be identifiable as such. In a HyCoRE reasoning stream it is assumed that for every node after the first non-start node, normal quality integration rules apply and v) The first aggregation node in the serialized flow sequence represents the end of start node processing.

Figure 30 illustrates the sequence and components participating in context flow reasoning. Context flows will ideally be executed in a thread independent of other context flows. The periodicity of flow execution is established by the context builder and based on refresh rate provided in the service contract. On execution of the reasoning stream (sequence item #1), fresh context for each node is retrieved successively. If there is no fresh/buffered context, the node's actuator is called upon for reasoning. The knowledge manager and context reporter are notified of new context inferences and are able to activate consumer actuators for reporting or knowledge repositories for storage. Figure 31 contains baseline pseudo logic for reasoning based on configured quality integration strategies.

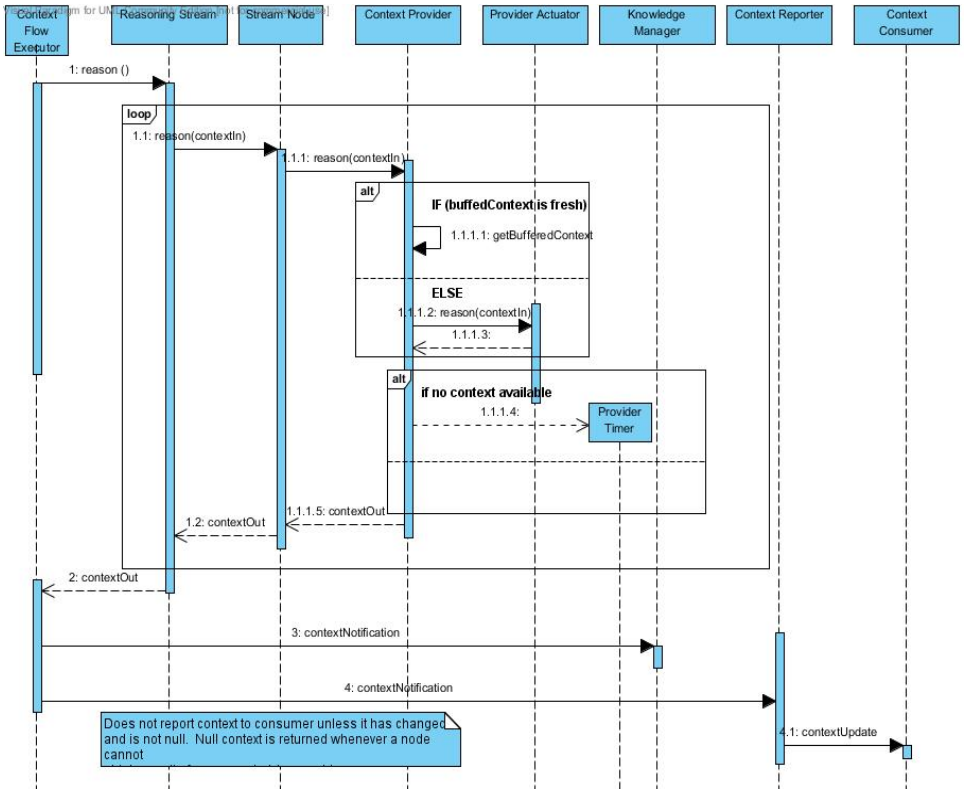

Figure 30 Context Reasoning Sequence Diagram

Context flows are suspended whenever a provider service contract is cancelled of if fresh context cannot be retrieved for an extended period of time. System events affecting the constitution of reasoning flows include new provider registrations, provider availability/quality changes, and template registrations. Through processing by the Adaptation Trigger Manager, suspended flows are reconfigured or replaced with alternative reasoning plans.

```
Context [] streamReason(ReasoningStream  stream){
        parallelExecution= stream->parallellExecution;
          for all  stream->nodes {
                    if node_i is a ReasoningStream{
                              contextOut = streamReason(node_i)
                    }
                    else if node_i is a SearchNode{
                              contextOut = searchReason (node_i)
                    }
                    else if node_i is a ContextFlow{
                              contextOut = flowReason (node_i)
                    }
                    else if node_i is a AggregationNode{
                              contextOut = aggregationReason (node_i)
                              parallelExecution=false
                    }
                    else{
                              contextOut = rucReason (node_i)
                    }
                    //only chain output in serialize mode
                    if ((node_i+1 exists)(!parallelExecution)){
                              node_i+1 ->addInput(contextOut)
                    }
          }
          return contextOut;
}
Context[] flowReason(ContextFlow flow){
           contextOut= streamReason(flow)
           contextOut->addProvider(flow)
           flow->averageQualityIndicator(latency, currtime-starttime)
           flow->copyQualityIndicators(contextOut->getQuality)
           return contextOut
}
Context [] RUCReason(IRUCNode node){
          for all context_i  node->getInput{
            context_i->setQualityIndicator(latency, currlatency+(receipttime-transmissiontime)
          }
          aqm = aggregateQ(node->getInput)
          contextOut = node->reason()
          node-> averageQualityIndicator(latency, currtime-starttime)
          aqm = propagateQ(aqm,node->getQuality)
          contextOut->setQuality(propagateQ(aqm, contextOut>getQuality)
          contextOut->addProvider(node);
        return contextOut
}
QualityMeasure[]  aggregateQ( Context[] context){
//Let vector QV_i vertically represent quality indicator  I or a all context
          for all QV_i{
                    aqm[i] = AST[i](QV_i);  //AST contains vector of  aggregation functions
          }
          return aqm;
}
QualityMeasure[]  propagateQ(QualityMeasures[] effector, QualityMeasures[] affected){
          for all Qmi {
                    pqm[i] = PST[i](effector[i], affected[i];
          }
          return pqm;
}
```

Figure 31 Context Reasoning w Quality Integration Pseudo Logic

5.6 Quality Verification

Quality perception requires knowledge and observation, but there is also need to verify quality. Otherwise we would have to trust the quality declared by a provider. This could cause our initial perception to be too high or low. Quality verification aides in forming a realistic view of true provider quality. It is also useful to maintain historical provider quality details. In this way, as the system encounters past context providers, this time to form realistic perception of their provided quality achieved is minimized. Using a quality feedback loop HyCoRE establishes the success or failure of context flow reasoning. When an inference is wrong, the context flow QIs are reduced. Conversely they are increased when correct. We define a positive reinforcement factor λ as the policy specified rate of quality increase. The negative reinforcement factor θ is the rate of quality decrease. The reinforcement factors are distributed through the context flow such that the total effect on the integrated context flow quality model corresponds to the factor. λ is implicitly positive, θ is implicitly negative. Setting λ or θ to 0 effectively disables the effects of quality feedback. Figure 32 illustrates negative quality feedback. This illustration highlights the fact that the node with the greatest impact on the high level context is also impacted the most by negative feedback. Of course there must be bounds on the positive or negative effects of this quality reinforcement. Figure 33 shows general quality feedback processing using HyCoRE components. The Quality feedback manager is responsible for identifying the correct flow and updating node quality. Once feedback had been performed, context flows are verified against system and consumer requirements. An adaptation trigger is generated if a flow fails to meet requirements.

λ-positive reinforcement factor = .1
θ-negative reinforcement factor= .2

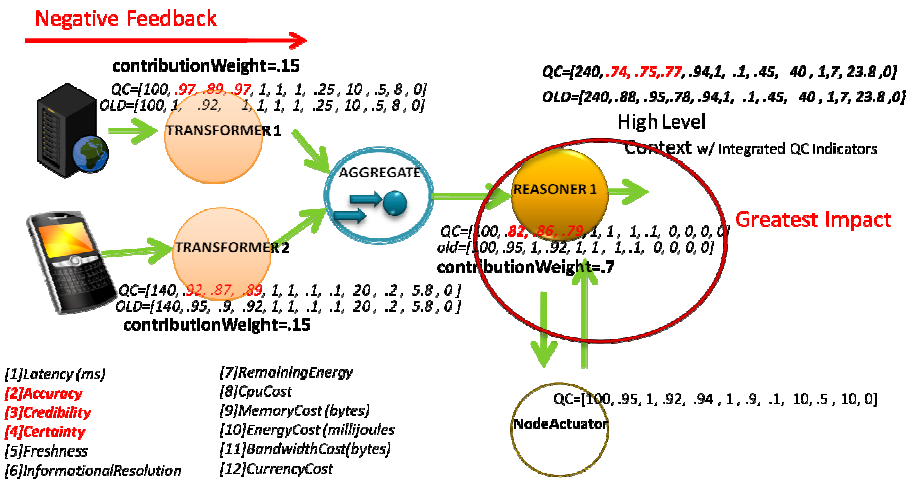

Figure 32 Quality Verification Example

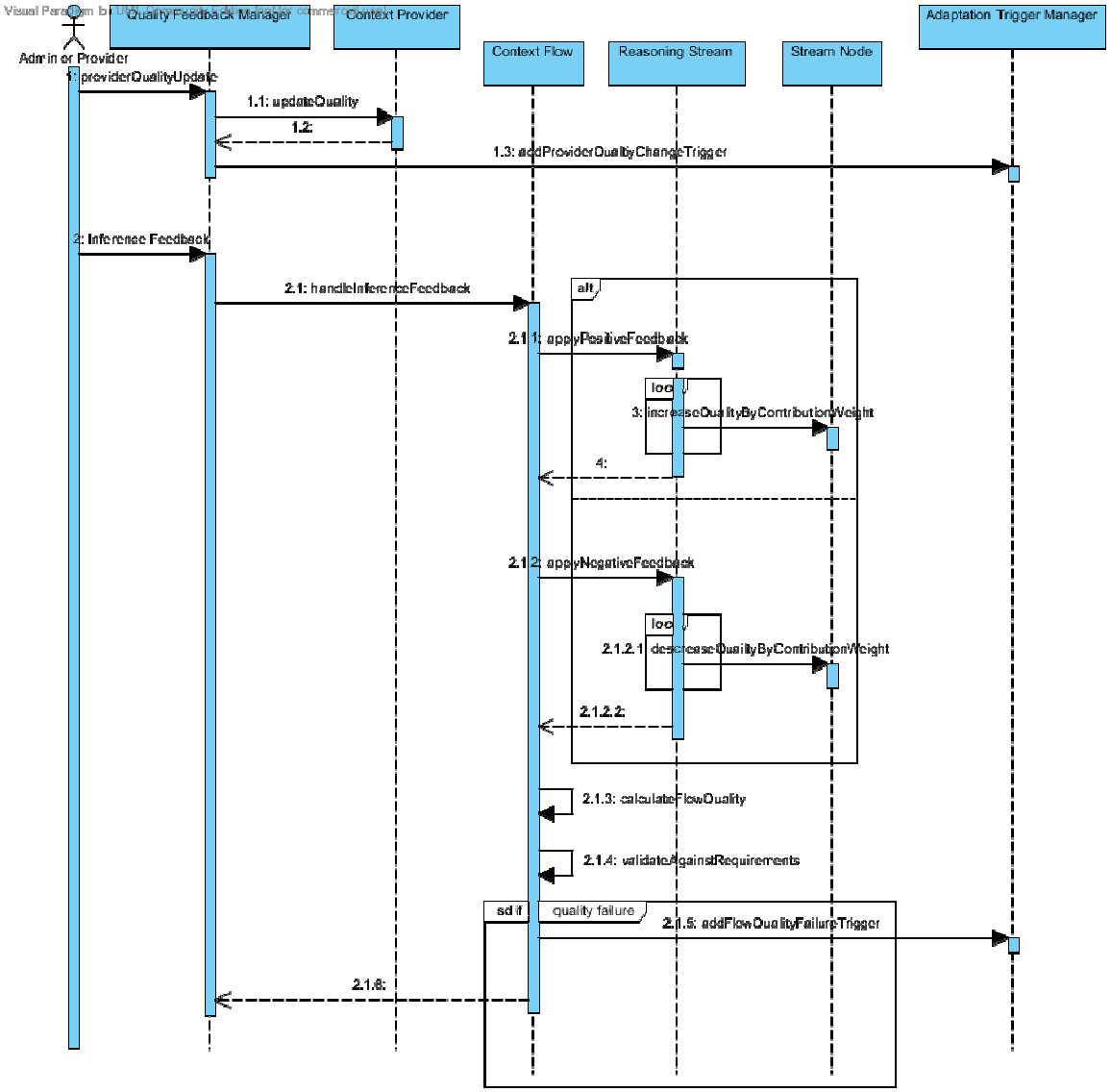

Figure 33 Inference Feedback Sequence Diagram

5.7 Quality Aware Reasoning Adaptation

This discussion begins with the assumption of existing providers, consumers and executing context flows. Consumer and system context requirements drive adaptation. To this end continuous monitoring to evaluate provider quality against requirements is required. Note that establishing context quality and cost is not as simple as selecting the device with the best *accuracy* or other singular quality indication. As can be seen by our component approach to reasoning, establishing true context quality requires an integrated approach to acquiring heterogeneous quality measures and propagating them through reasoning and transformational processes to produce a composite high level context quality (refer back to Section 4.3 for details). Context reasoning adaptation is based on the adaptation action that is determined after evaluating integrated QC indication against context requirements during adaptation trigger processing. An adaptation trigger is a system event which affects the constitution of reasoning. These system events include: i) new context provider registration; ii) provider failure or quality update notifications; iii) consumer context requests; iv) and quality feed-back notifications.

HyCoRE has established QC attestation points and associated adaptation triggers for evaluating integrated context QC indicators. The design of the Adaptation Trigger manager is such that adaptation actions can be traced back to associated attestation points and trigger handlers. Figure 34 illustrates considerations of the context adaptation process.

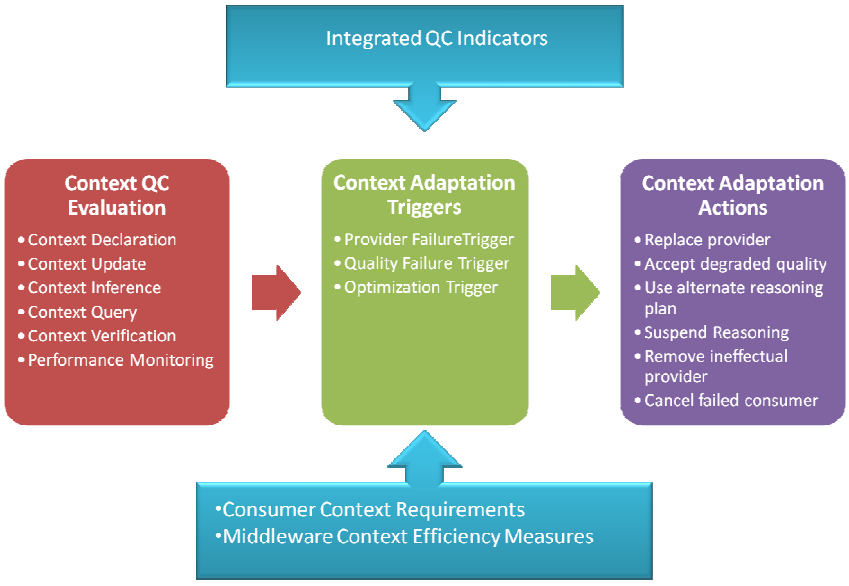

Figure 34 Context Reasoning Adaptation

The quality evaluation/attestation processing points in HyCoRE are as follows:

Context Declaration Processing Point (DPP) occurs when a context provider or consumer initially registers with middleware, declaring its context and QC indicators.

Context Update Processing Point (UPP) occurs when a context provider or consumer updates its quality indicators. This also includes service cancellation events. A provider cancellation reduces the availability of context. A consumer cancellation reduces the middleware requirement for inferring context.

Context Query Processing Point (QPP) occurs when context is found in a knowledge base. This processing point is associated with the search component of a context flow.

Context Inference Processing Point (IPP) occurs on every execution of a context flow. This implies observation, propagation and aggregation of QC indicators.

Context Verification Processing Point (VPP) occurs when feedback is used to update QoC positively or negatively.

Performance Monitoring Processing Point (PPP) in addition to event driven quality attestation, HyCoRE monitors for adaptation triggering events.

Adaptation triggers may be produced at quality attestation processing points. A *quality failure trigger* (QFT) occurs when a context flow node's quality degrades unacceptably, no longer meeting combined application and middleware goals. A *provider failure trigger (PFT)* occurs when context flow node is unresponsive or for any reason is no longer providing context. *Optimization Triggers (OPT)* occur when action that would result increased middleware performance can be taken (i.e. higher quality providers available).

Trigger processing results in one of six categories of adaptation actions. These are: i) replace a context reasoning flow provider; ii) continue inferencing with degraded quality; iii) derive an alternative reasoning plan; iv) suspend reasoning for a specific context; v) cancel ineffective context providers; and vi) cancel irrecoverably failed consumers. We detail our approach in the sections to follow.

5.7.1 *Adaptation Triggered by Dynamic Provider Availability*

There are a number of specific triggers associated with provider availability. However, the processing can be summarized as follows: When any provider becomes unavailable, we must attempt to find a suitable replacement in the context flow. The system checks for either an alternative node or an alternative context template which may be instantiated for continued inference. If no replacement node can be found, inferencing plans associated with the failed provider are halted (i.e. context flows become inactive). A flow may be inactive for a configured amount of time. During this time it may be reactivated if any previously unavailable nodes become available or if a new provider, suitable for replacement, becomes available. Any provider or flow that has been inactive longer than a configured period is removed from the system.

On events where context providers become reactivated after a period of inactivity, a search for a suitable use case begins. The first use is to reactivate any flow still associated with

the provider. If the provider is not associated with any flow, we check to see if it can be used to repair any other inactive flow. The next use would be to complete requirements of an existing consumer contract. In some cases, the system may not be inferring all of the contexts required by a consumer. The last use would be for optimizing any existing flows.

5.7.2 *Adaptation Triggered by Dynamic Provider Quality*

Provider quality is evaluated against consumer and system requirements during context inferencing, on receipt of provider quality update or new provider context declaration. A number of triggers are associated with quality changes at these verification points. A provider quality update causes the systems to re-evaluate any inferencing flows associated with the provider. If any participating provider quality causes the overall high level context to fall below requirements, a search for the best alternative is begun. An alternative context provider will match on context-meta data information. A context meta-model specifies the physical and semantic criteria for context equivalency. Context providers are interchangeable based on matching meta-model information. In addition to matching meta-data, a chosen provider alternative must have sufficient quality to yield an integrated quality minimum.

If no alternative is found, context inferencing associated with the provider is halted in the same manner as mentioned in the availability discussion. The system will wait on the first of the following events to occur: i) context provider quality improvement resulting to restoration of inferencing; ii) new context provider alternatives enter the system leading to restoration of inferencing; iii) context flow inactivity timeout which causes context flow and associated consumer contracts to be removed from the system. Cancelling a consumer contract ensures that the system operation efficiency is not indefinitely penalized. A context provider is only removed if its quality falls below system requirement, remaining energy is zero, or remains unavailable for an extended period. Though a provider may no longer satisfy current consumer requirements, new consumers could arrive for which lower QC indicators are sufficient.

5.7.3 *Adaptation for Reuse and Optimization*

Receiving new consumer requests and provider offers affords opportunity to improve system performance. Existing flow quality may be improved by replacing a participant. In addition to event driven optimization, periodic performance monitoring identifies when middleware is falling below established thresholds. The system only supports a configured number of providers at any given time, so ineffectual provider contracts are also cancelled after a configured period of inactivity. Though not actively used for inferencing, inactive context flows consume limited system resources and are also discarded. Middleware operational efficiency is directly affected by existing consumer contracts, so HyCoRE will eventually cancel consumers for whom context inferencing is irrecoverably failing. Cancelling failed consumer contracts causes the middleware operational efficiency to rise.

5.7.4 *Quality Monitor*

The quality monitor component of Figure 26 is a suggested background task for verifying the integrity of reasoning, producing adaptation triggers as needed. HyCoRE has many events that trigger adaptation, but as with any system, race conditions and anomalies lead to unexpected states. The quality monitor catches what adaptation trigger handlers miss. Also, the context builder can be configured to quickly instantiate the first available flow rather than the most optimal. The quality monitor identifies optimizations of flows, triggering run-time reasoning adaptation for improved performance.

5.7.5 *Platform Performance Manager*

In Section 4.4, several QoCS measures were introduced. It is the Platform performance manager which maintains the current state of these measures and evaluates them against requirements. Adaptation Triggers are generated as reasoning deficiencies are identified.

5.7.6 *Adaption Trigger Manager*

An adaptation trigger is a system event which affects the constitution of reasoning. Though, adaptation trigger handlers may be distributed and executed in independent threads, the adaptation trigger manager provides a central point in the system to which reasoning adaptation can be traced. The events show in

Table 7 are identified and handled in HyCoRE. Each event occurs at an identifiable quality attestation point originating from one to the components shown in Figure 26.

Table 7 Detailed Adaptation Event Descriptions

Event Explanation	Trigger	QVP
Template Manager has identified new template projections	NewProjectionAvailable	DPP
More optimal projection for existing flow exists	OptimizedProjectionAvailable	PPP
New context provider is available	ProviderAvailable	DPP
Existing provider availability has changed	ProviderAvailabiltyChange	UPP
Consumer has update quality requirements	ConsumerQualityChange	UPP
Provider's quality had changed	ProviderQualityChange	UPP,VPP
Context flows is failing to meet consumer or system requirements	FlowQualityFailure	PPP
Provider is not responding	Provider Failure	PPP
Provider is failing to meet system quality minimums	ProviderQualityFailure	PPP, IPP,VPP
Provider had been unavailable longer than the allowable time	Provider Availabilty Expiration	PPP
Flow has been inactive due to failed or inactice providers longer than allowable time	Flow Inactivity	PPP
Middleware QoCS Context Effectiveness measure is low	Low Context Effectiveness	PPP
Middleware QoCS Operational Efficiency measure is low	Low Operational Efficiency	PPP
System has been unable to meet consumer context request for longer than the allowable time	Consumer Not Assigned Flow Expiration	PPP

5.8 HyCoRE Reasoning Summary

The HyCoRE architecture and data models were implemented in Java for concept validation. The choice of Java with the associated threading and serialized object representation are prototypical design choices. Figure 35 and Figure 36 provide a detailed view of the prototypical HyCoRE implementation architecture from two perspectives. There are several algorithms used to orchestrate adaptive high level context reasoning. These include: i) reasoning plan projections and scoring; ii) meta-data matching; iii) reasoning with integrated quality and provenance; and iv) periodic QoCS measurement. Baseline algorithms have been provided to capture the functionality. Since, these are the largest potential performance bottlenecks in HyCoRE; they should be implemented in pluggable fashion. In future iterations of HyCoRE, baseline reasoning algorithms may be replaced with more efficient ones

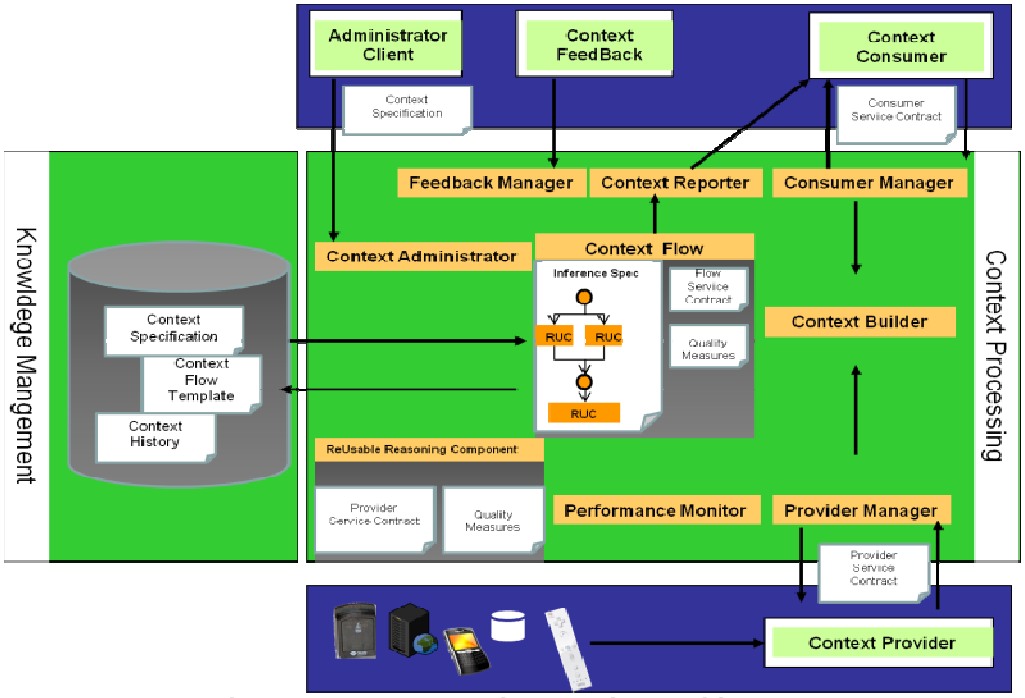

Figure 35 HyCoRE Implementation Architecture

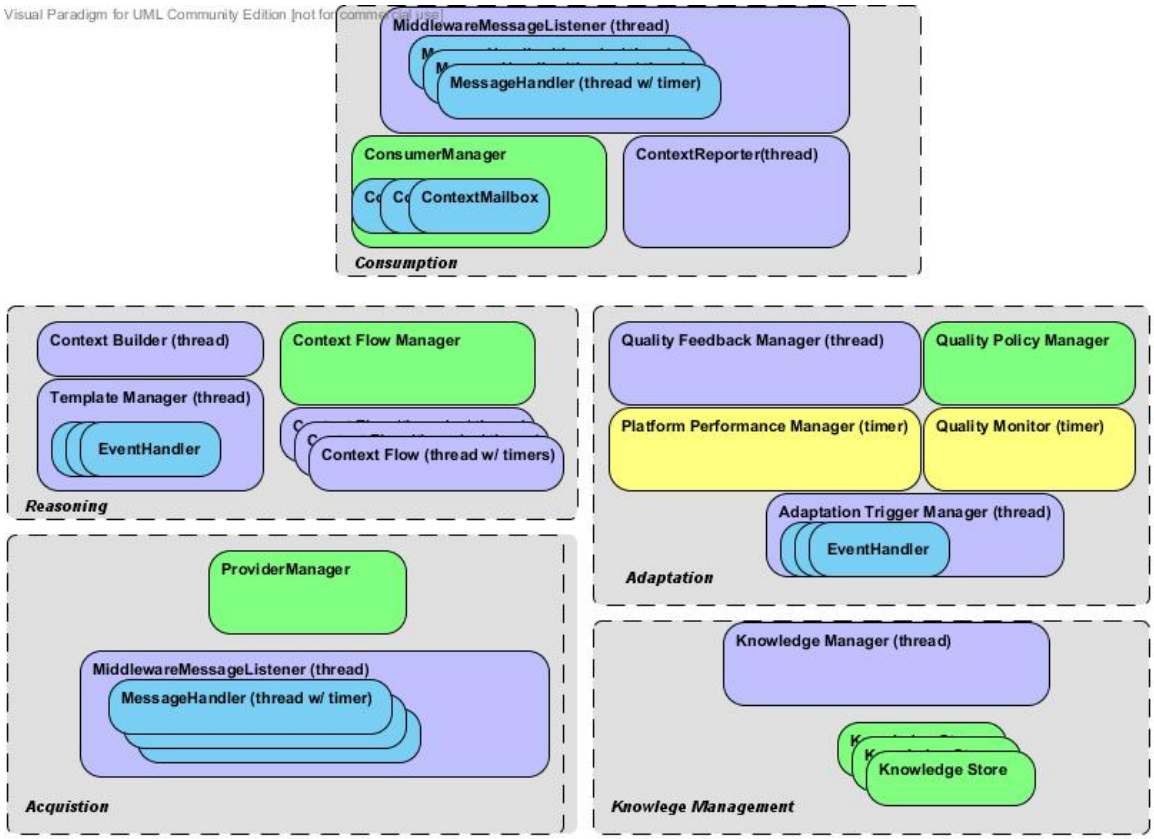

Figure 36 HyCoRE Implementation Components

The following section briefly describes each component's role in high level context reasoning.

5.8.1 *Components and Threading*

This reference implementation of HyCoRE is designed with a mixture of persistent and transient threads, timers and container components. Following is an explanation of each component.

Message Handler- HyCoRE listens for message on a configurable port. A transient thread is spawned to handle each message.

Consumer Manager- Registration messages, cancellation and general management of context consumers are handled by the Consumer Manager. In this design, the Consumer manager is a singleton container component.

Context Reporter- threaded component which distributes inferred context to its destination.

Context Builder- threaded component which build contexts flows to meet consumer requests. Refer to Section 5.2 for design discussion.

Context Flow Manager- Container component which holds references to all context flows

Template Manager– multi-threaded component which handles context template registration messages and others that affect the configuration of templates. This manager is responsible for dynamic creation and updating of context template projection. A separate transient thread is used for processing messages since the function of finding projection can be time consuming.

Context Flow (executor thread w/ timer)- Context Flows are instantiated reasoning plans bound to specific providers. Each flow executes in a separate transient thread so the timing of each inference is independent. Refer to Section 5.1 for design discussion.

Provider Manager- Registration messages, updates, cancellation and general management of context providers are handled by the Provider Manager. As with the consumer manager, this is a singleton container component.

Quality Feedback Manager- thread component which adjusts the QoC of context flow components based on positive or negative feedback.

Quality Policy Manager- singleton container component which holds all configured system policies.

Quality Monitor- periodic timer component which looks for unhandled failure situation and ways to optimize existing context flows. Refer to Section 5.7.4 for design discussion.

Platform Performance Manager- periodic timer component which measures QoCS. Failures result in adaptation triggers passed to the Adaptation Trigger Manager. Refer to Section 5.7.5 for design discussion.

Adaptation Trigger Manager- threaded component which handles adaptation trigger events. Refer to Section 5.7.6 for design discussion.

Knowledge Manager- component which handles knowledge persistence and retrieval. For this implement KM is a marker component which is not fully functional, though easily extendable to meet design function.

Knowledge Store- provides an interface where interested stores may be registered with the knowledge manager.

.

CHAPTER 6

HYCORE EVALUATION

The HyCoRE context reasoning framework adapts to sustain high level context inference in the face of dynamic device quality and availability. A derived composite measure of high level context quality is used as a basis for adaptation. Reasoning plans are flexible in that components may be replaced with current or future context providers with match generalized information description called context meta-data. Context consumers specify context and quality requirement using context meta-data. The middleware operation measures reflect success inferring high level context while meeting consumer requirements. It is difficult to find a single application that requires everything HyCoRE supports. HyCoRE is general purpose framework, supporting a variety of applications. For this reason, we demonstrate two applications of HyCoRE reasoning to highlight the various features of the architecture.

6.1 Law Enforcement Search Evaluation

An interesting use case for middleware mediated context is sustaining an application's context needs in the face of fluctuating cost or quality indicators and efficiently managing the resource constrained devices serving as context providers. Our solution for this use case employs a reusable component based approach to context reasoning; interjecting into the composition of context to facilitate adaptation. All adaptation is based on integrated quality and cost indicators. Integrated indicators are evaluated against consumer and middleware system requirements. This approach to reasoning can prove useful in several context reasoning situations where diverse context providers participate in producing high level context. Through a real world scenario, we

demonstrate a middleware approach that provides i) efficient actuation of devices for context sensing; ii) participatory context processing/inference; and iii) sustained context dissemination.

In the HyCoRE framework, application and middleware system quality requirements drive adaptation. The system adapts to meet consumer requirements according to middleware QC priorities. We illustrate the HyCoRE process of quality integration and corresponding effects on QoC and QoCS measures while adapting for context sustainability with the following simulated scenario: Law enforcement is conducting a search for suspected criminals. The investigation has lasted for some time and manual techniques have yielded no leads. It is certain that the suspects are hiding in one of three geographic areas. Rogue law enforcement officers may be withholding vital information, or maybe the answers are just outside human perceptive faculties. A recent study on *inattentional blindness* [17] verifies that humans cognitively miss events occurring right before their eyes. Automated visual and audio data collection bridges the void. For a brief period, pervasive cameras, video, microphones are deployed in the designated areas using plain clothes enforcement agents and unmarked vehicles. The sensors are positioned inside and outside of each agent's field of view as criminal elements will likely remain outside of view or be disguised. HyCoRE can be deployed on devices in such a pervasive environment to aid in context inference, energy efficient device actuation and context sustenance.

In this evaluation, HyCoRE is deployed in each area for assistance with inferring intelligence and actuation of deployed sensors. Each geographic area is divided into cells for which a single set of sensor is sufficient for coverage. Agents and vehicles move between the geographic areas, each time registering with the local HyCoRE instance as a context provider for the targeted cell. Within a sensor period, activated sensors collect data for a period and transmit batches to HyCoRE for intelligence processing. Inferred location changes for designated suspects are reported to interested law enforcement agencies, each having distinct qualitative context requirements.

Figure 37 Law Enforcement Scenario

HyCoRE function is simulated for a single geographic cell where sensor availability fluctuates due to mobility of agents and vehicles. Also, device quality indicators degrade as the sensors are used over time. Sensors located in the unmarked vehicles have significantly higher quality and power than the invisible low powered sensors the agents are wearing. HyCoRE has been configured with reasoning plans, feature extraction and reasoning algorithms to identify persons and vehicles of interest. The only pieces of inference that do not reside within HyCoRE are the low level video and audio context sources. HyCoRE makes a best effort sustaining context inferencing through the dynamic situation. Energy levels of low-level sources among other quality/cost indicator as well as device availability are dynamic. There are no consumer requirements in this evaluation. Energy conservation during reasoning is the primary system concern for this middleware evaluation, so *energyCost* is configured as a middleware QC priority.

Of course, this is a fictitious scenario and there are numerous details not discussed. This work illustrates how pervasive middleware can be used for inferencing beyond personal monitoring. More specifically, the evaluation herein demonstrates how the HyCoRE framework supports sustaining context with resource efficiency. Middleware performance, where no adaptation is applied, is compared to those obtained using adaption. As we show, the mean time before context reasoning failure and overall operational efficiency increase where adaptation is applied. In summary, the middleware features demonstrated by this law enforcement scenario are:

- ✓ Quality Integration
- ✓ Middleware Operational Efficiency
- ✓ Reasoning plan using synchronous context pull from providers
- ✓ Adaptation to Low Level Context Provider Availability

6.1.1 *Consumer and Providers*

For our law enforcement evaluation scenario, the participating context providers and consumers are illustrated in Figure 38 and Figure 39.

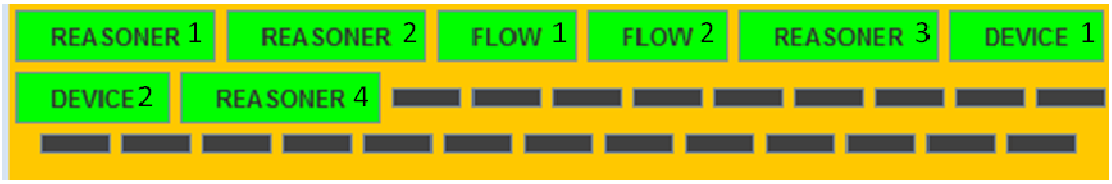

Figure 38 HyCoRE Providers (Law Enforcement Scenario)

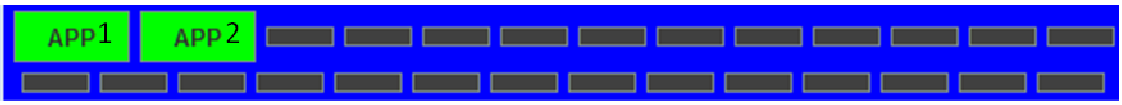

Figure 39 HyCoRE Consumers (Law Enforcement Scenario

The consumers (App1 and App2 of Figure 25) of our evaluation scenario are two distinct law enforcement agency applications monitoring the location of suspect. For simplicity, the interested agencies have requested the same context and qualitative criteria. We illustrate such

consumer requirement in the context meta-data shown in Figure 16 Example Context Consumer Requirements. The eight providers of our evaluation include image and sound recorders deployed in the targeted cells. Many context reasoning devices are multisensory, but for illustrative purposes we show each capability as a distinct provider. Also, we initialize the system with algorithmic providers for reasoning. With regard to Figure 38: Provider 1 (*Device 1*) is an audio recording device. Provider 2 (*Device 2)* is an image capture device. Provider 3 (*Reasoner 1*) is an audio Mel-frequency cepstral coefficient **(**MFCC) Transformer. Provider 4 (*Reasoner 2*) is a generalized Hidden Markov Model based audio reasoner that requires MFCC inputs. Provider 5 (*Reasoner 3*) is an image recognition reasoner. Provider 6 (*Reasoner 4*) is a distribution based suspect location reasoner. Provider 7 (*Flow 1*) is an audio based suspect locater reasoning plan based on providers 1, 3, 4 and 6. Provider 8 (*Flow 2*) is an image based suspect locater reasoning plan based on providers 2, 5 and 6.

6.1.2 *Context Flows*

Both context flows are illustrated in Figure 40. Notice that context providers may be of several types: devices, reasoners and flows.

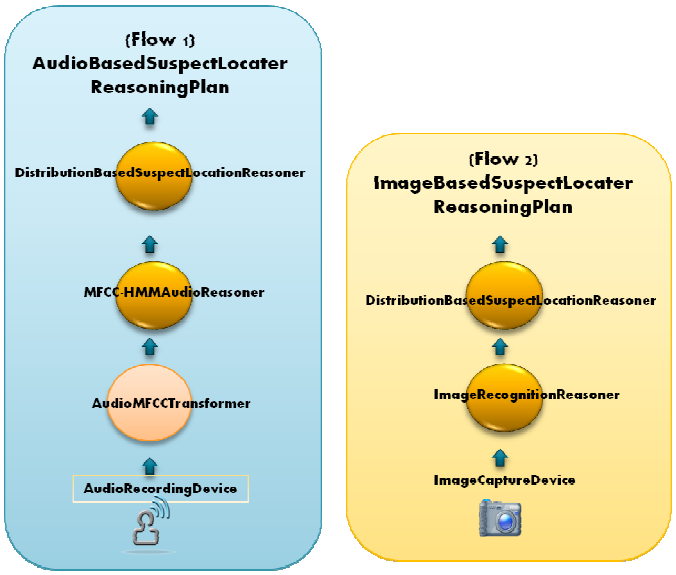

Figure 40 Context Flows (Law Enforcement Scenario)

.

6.1.3 *Setup and Operation*

The middleware periodically executes context flows, reporting context and associated QC indicators. Context flow configuration is fixed and device costs are constant. The consumers of the evaluation have a subscription for context for a period of time. The system endeavors to sustain context and quality over the period using adaptation. As the low level sensors (also *referred to as context providers*) vary in quality, so does the resulting high-level context. Additionally, the middleware quality of context service measures appropriately reflect quality changes. These indications are the basis for reasoning adaptation. In the following evaluation scenario, middleware operational efficiency[7], meantime before context failure[8], integrated energy and bandwidth cost[9] indicators are observed. The simulated lifespans of source devices is scaled down to five minutes for illustrative purposes. Initial system configuration is a follows:

A device energy decline pattern is configured for uniform decline of .05 every 15 seconds, so that device energy is exhausted after 5 minutes. The system is set to clean up inactive providers after two minutes and flows after three minutes. Consumers who can no longer be served are also cancelled after two minutes of being detached from a reasoning plan. The effect of consumer contract cancellation can be seen in the OPEFF snapshots, where operational efficiency returns to 100%. Essential system behavior highlighted in this evaluation is: i) context reasoning failure; ii) reasoning adaptation, integrated cost indications; iii) cancellation of ineffective providers and consumers that can no longer be served. Each scenario ends when there are no more consumers and system operational efficiency has returned to 100%. To capture essential system behavior, system performance measurements are taken at progressive time intervals described in Table 8.

---

[7] See Section 4.4.5

[8] See section 4.4.6

[9] See section 4.4.3

Table 8 Initial Performance Snapshots

Snapshot	System Performance Snapshot Times	Description
Snapshot 1	T0+312000*ms*	Devices of scenario 1 have depleted energy and context reasoning cannot be sustained. Other scenarios are continuing to sustain context.
Snapshot 2	T0+624000*ms*	Following system clean-up of inactive providers< including flows> and cancellation of consumer contracts which can no longer be served in scenario 1. The middleware operational efficiency increases back to 1 for scenario 1.
Snapshot 3	T0+936000*ms*	Following system cancelling consumer contracts which can no longer be served in scenario 3. The middleware operational efficiency increases back to 1 for scenario 3.
Snapshot 4	T0+1900000*ms*	Energy for devices of scenarios 2 and 4 are depleted. Context reasoning failure occurs.
Snapshot 5	T0+2050000*ms*	Following system clean-up of inactive providers< including flows> and cancellation of consumer contracts which can no longer be served in scenario 2. The middleware operational efficiency increases back to 1 for scenario 3.

The following sections describe each scenario. Results are presented in section 6.1.8.

6.1.4 *Scenario 1: No Adaptation – Declining Energy*

In scenario 1, HyCoRE is initialized with only a single instance of each type of provider shown in Figure 37. The energy of the image capture and audio recording devices declines uniformly. When device power is exhausted, the devices are no longer available for use as context providers. HyCoRE adaptation is triggered; either replacing the provider or suspending reasoning until an alternative provider is available. There are no alternative providers in this scenario, thus

adaptation fails and operation efficiency declines to 0%. In this scenario, operational efficiency does not return to 100% until consumer contracts are cancelled.

6.1.5 *Scenario 2: Reasoning Adaptation – Declining Energy*

Scenario 2 is the same as scenario 1 with added alternative audio and image context providers. Scenario 2 employs simulated self-replicating audio and image providers, where the number of replications is five. The bandwidth and energy cost of each replication successively increases by 15% and 30%. This is an improvement over the non-adapting scenario 1. When the system detects a failed provider, it immediately replaces it with the best alternative provider. Inferencing is minimally interrupted as reflected in the OPEFF and MTBCF graphs. Device energy decline pattern is configured so that each device can actively collect for five minutes before energy is exhausted. Device actuation which initiates energy decline does not begin until HyCoRE identifies and adds provider to a context flow and the context flow is started. So, HyCoRE may sustain context based on these providers for up to 25 minutes. The operational efficiency remains at 100% while there are alternative providers available for adaption. Middleware cost fluctuations reflect the changing cost of inferencing as source providers are replaced. It is difficult to discern that the system is adapting for to the provider with least energy cost. This is due to the fact that the providers are continuously moving in and out of range. However, scenario 2 does show that each adaptation selects with increasing cost.

6.1.6 *Scenario 3: No Adaptation – Energy Decline w/Dynamic Availability*

Scenario 3 makes use of the same singular instances of image capture and audio devices from scenario 1. No device replication is applied. In addition to energy decline, the devices move in and out of HyCoRE range as simulated through the application of an availability pattern. For this case we use an *on/off* pattern with 1 minute durations. Availability pattern is applied for eleven minutes. When a device becomes unavailable the middleware attempts to find another provider to sustain context inference. However, since there are no other devices available, inferencing will be suspended until the device comes back into HyCoRE range. Energy only declines when a

provider is being actively used for inferencing, so simulated energy decline suspends when the device becomes unavailable. Device energy is exhausted after being used for a total of five minutes; at which time HyCoRE can no longer adapt. Accounting for *off* periods, context inference should fail permanently after nine minutes. Thereafter, we observe provider and consumer cancellation. Adaptation initiated by availability changes can be observed in all of the graphs. Fluctuations in middleware cost, MTBCF and OPEFF are all indicative of flows being suspended and restarted. As with the other scenarios, observe that operational efficiency returns to 100% when consumer contracts are cancelled.

6.1.7 *Scenario 4: Reasoning Adaptation – Energy Decline w/Dynamic Availability*

Scenario 4 functions the same as scenario 2 with the added dynamic availability of scenario 3. Alternative providers are available for adaptation, so HyCoRE is able to successfully sustain context inferencing despite changing availability. Unlike the drastic performance drop shown for scenario 3, successful context sustenance is shown in that the operational efficiency remains at 100% until after either all image or audio device energy is exhausted. Also, the MTBCF is more normalized around 60s, which is consistent with the availability interval simulation setting.

6.1.8 *Law Enforcement Search Evaluation Results*

As discussed previously, we capture progressive system snapshots are progressive time intervals. This allows us to observe system measures as i) quality is integrated; ii) reasoning fails and iii) adaptation is handled. The first measure on each X axis represents the indicated number of milliseconds since system start. Each subsequent measure is 11 seconds later. System collection is set for every 11 seconds with a history size of 40, so that each graph reflects 7.3 minutes of performance. Refer to the preceding evaluation scenario descriptions for a discussion on the results presented below.

6.1.9 *Snapshot 1- System Startup + 312000ms*

Snapshot 1 graphs begin 2ms after system startup, covering 30, 11 second time intervals. As mentioned previously, the source device energy is configured to be depleted after five minutes of

total use. This snapshot covers the period where devices of scenario 1 have depleted energy and context reasoning cannot be sustained. Other scenarios are continuing to sustain context. The OPEFF graph shows that scenario 1 experiences permanent context failure around 300ms. Since, scenarios 2-4 include intervals of unavailability, where devices are not actively used, the lifetime of each device is extended. However, we do see cyclic OPEFF, MTBF and Cost in scenario 2 that mirror periods of unavailability where there are no alternative devices. The trend in the MTBCF is also indicative of system behavior. Scenario 1 and 2 MTBCF continue to rise for five minutes, indicating uninterrupted context reasoning for this period. In the case of scenario 1 the MTBCF remain constant afterwards since reasoning is suspended and never restarted. In scenario 2, reasoning is suspended, adapted and restarted. Thus, the average running time is reduced. Integrated middleware costs graphs are indicative of the cost using actives devices at that specific time instant. Scenarios 1 and 3 do not adapt, so the cost is either fixed or 0. Scenarios 2 and 4 reflect adaptation using the most energy efficient devices available at the time.

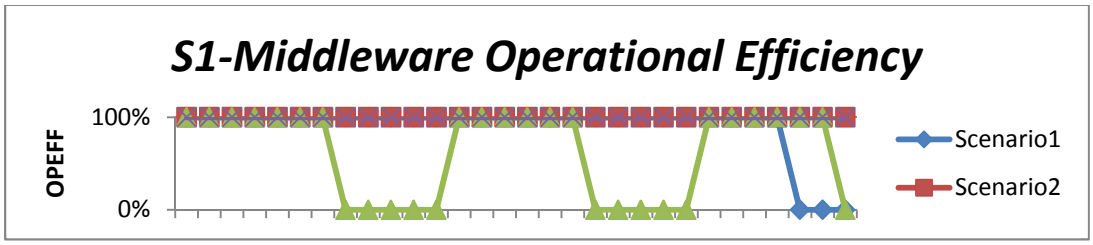

Figure 41 Snapshot 1 – Operational Efficiency

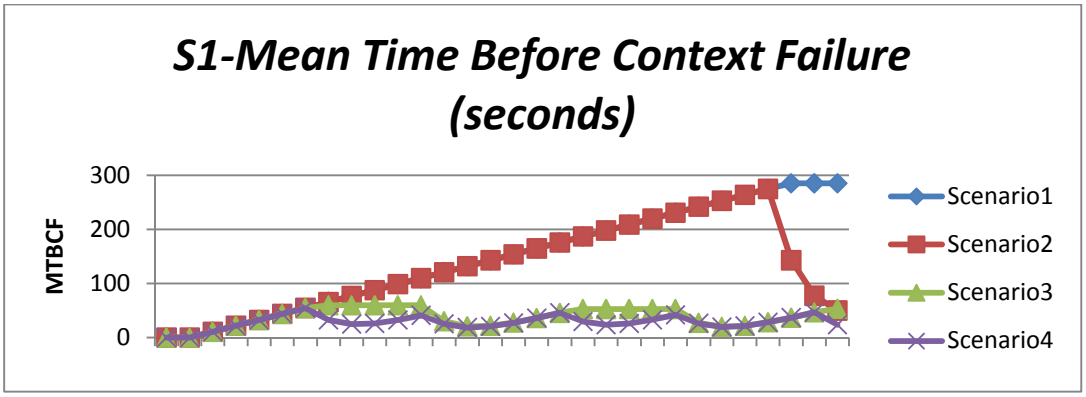

Figure 42 Snapshot 1 – Meantime Before Context Failure

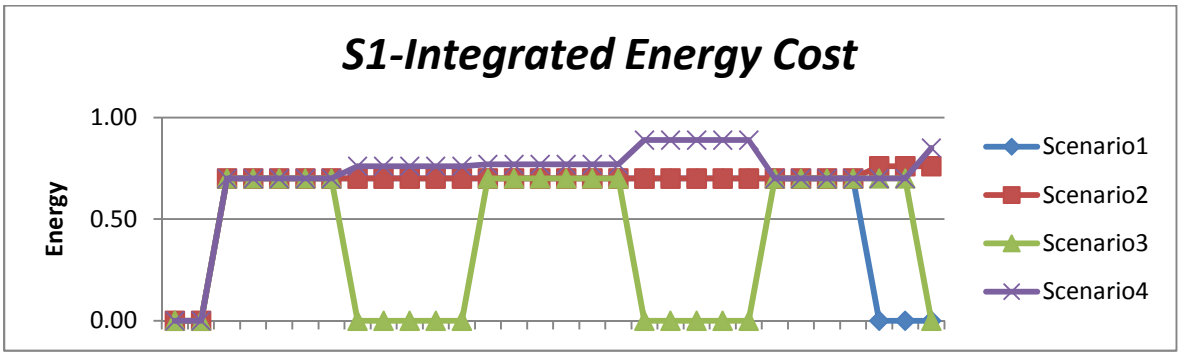

Figure 43 Snapshot 1 – Integrated Energy Cost

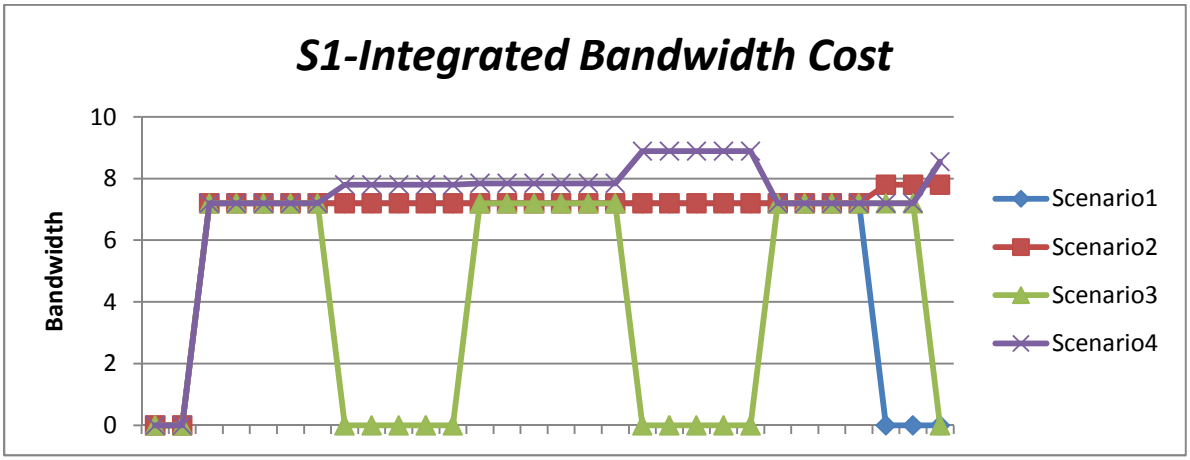

Figure 44 Snapshot 1- Integrated Bandwidth Cost

6.1.10 *Snapshot 2- System Startup + 624000ms*

Snapshot 2 graphs begin 1870788ms after system startup, covering 40, 11000ms time intervals. This snapshot covers the period where scenario1 fails and includes the cancellation of consumer contracts. After five minutes, scenario 1 fails irrecoverably since device energy is depleted. Inferencing costs drop go to 0 and OPEFF returns to 100% since the system is no longer inferencing. In contrast, scenarios 2and 4 are continuing to sustain context using alternative providers. There are no alternative providers in scenario 3, so intermittent context failure can be observed in all graphs.

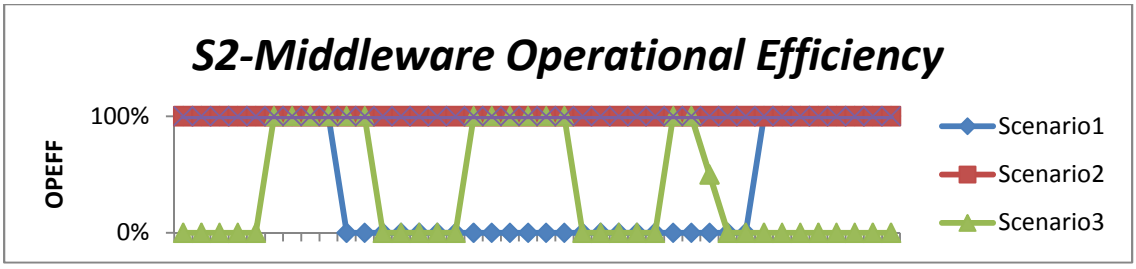

Figure 45 Snapshot 2- Operational Efficiency

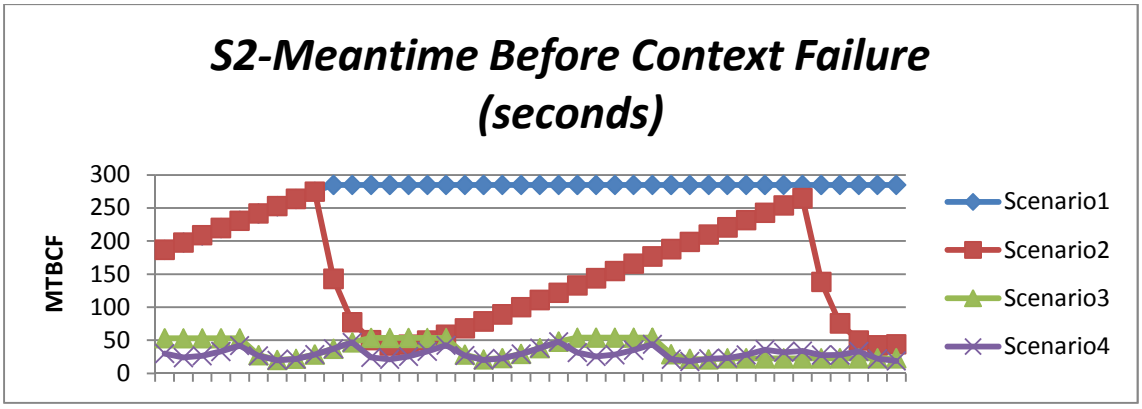

Figure 46 Snapshot 2- Meantime Before Context Failure

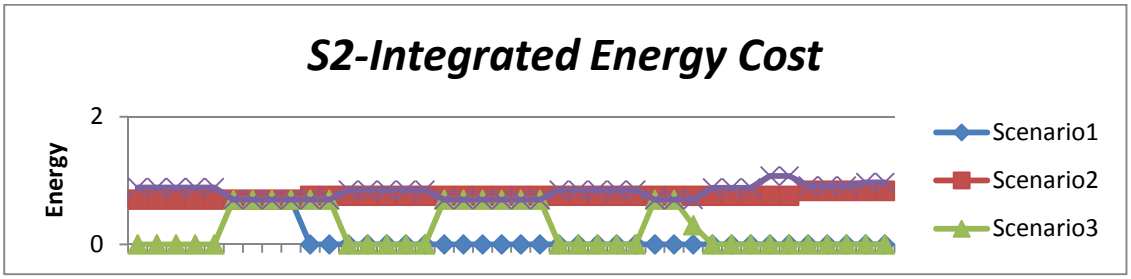

Figure 47 Snapshot 2- Integrated Energy Cost

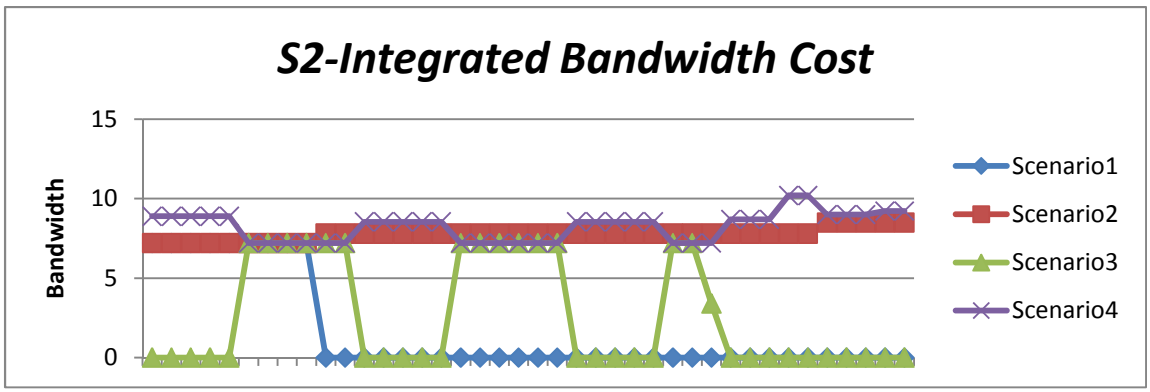

Figure 48 Snapshot 2- Integrated Bandwidth Cost

6.1.11 *Snapshot 3- System Startup + 936000ms*

Snapshot 3 graphs begin 506078ms after system startup, covering 40, 11 second time intervals. This snapshot covers the period following system cancellation of consumer contracts which can no longer be served in scenario 3. Note that fluctuations in MTBCF and cost indicate points of adaptation while a consistent 100% value for OPEFF is indicative of successfully sustaining context. The OPEFF of Scenario 3 returns to 100% when its consumer contracts are deleted after 12.5 min. Scenarios 2and 4 are continuing to sustain context using alternative providers. It can be seen in scenario 2 costs, that the more energy efficient alternative is being selected for adaptation. This fact is obscured in scenario 4 since it includes dynamic availability. The MTCF of scenario 4 destabilizes and begins to rise since the availability simulation expires after 11 min. Thereafter, the devices are always available. Scenario 3 has already failed permanently and the MTBCF remains fixed to the last reasoning values.

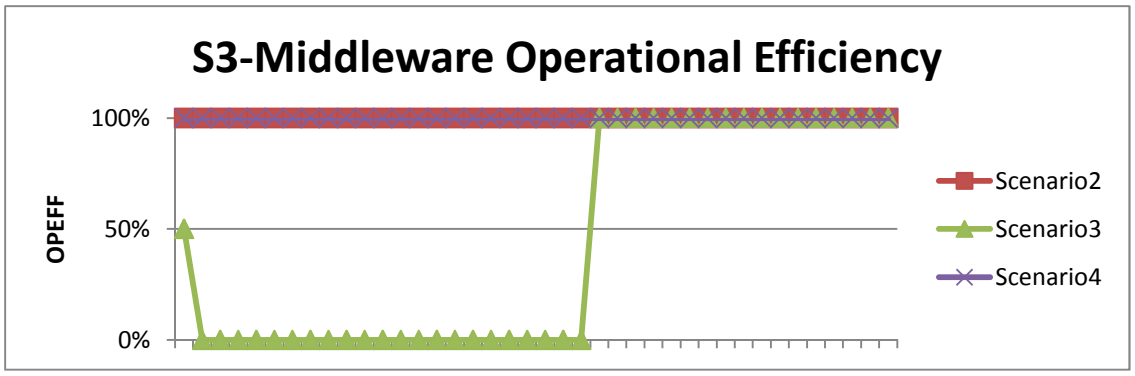

Figure 49 Snapshot 3- Operational Efficiency

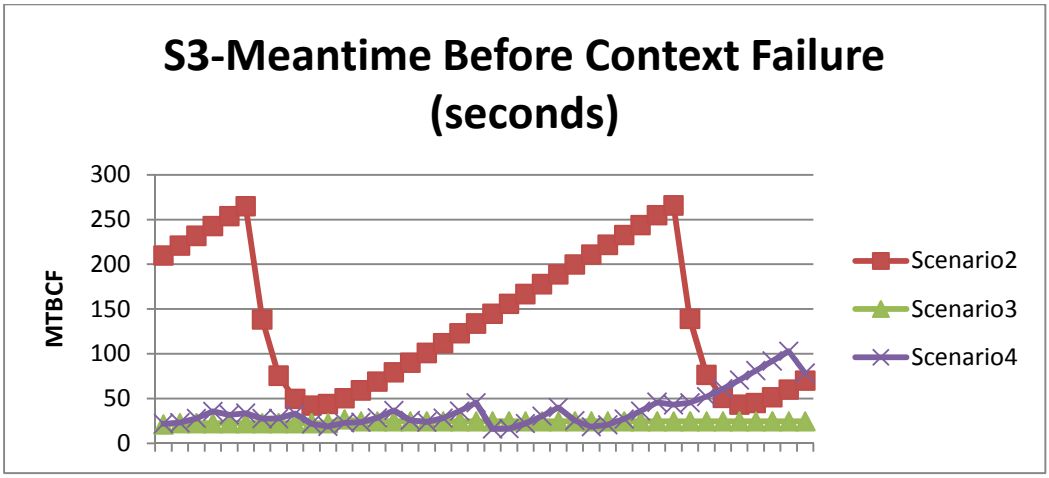

Figure 50 Snapshot 3- Meantime Before Context Failure

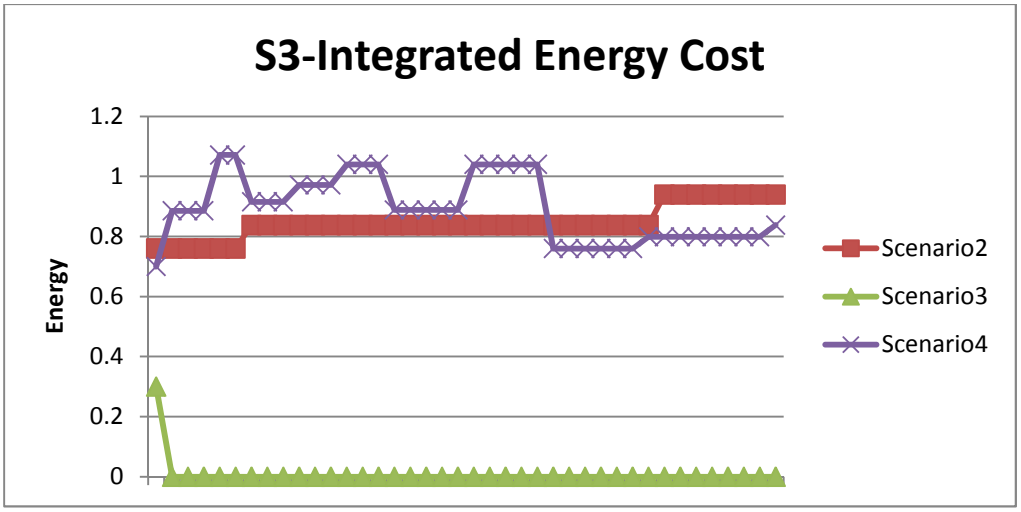

Figure 51 Snapshot 3- Integrated Energy Cost

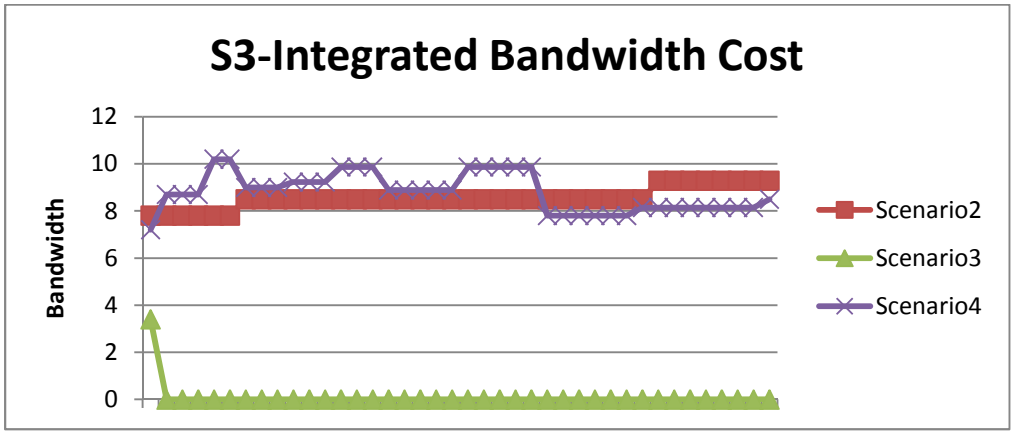

Figure 52 Snapshot 3- Integrated Bandwidth Cost

6.1.12 *Snapshot 4- System Startup + 190000ms*

Snapshot 4 graphs begin 1463078ms after system startup, covering 40, 11 second time intervals. This snapshot cover the period where context reasoning failure occurs for scenarios 2 and 4. Scenarios 1 and 3 have already failed. Scenarios 2 and 4 continue to sustain context using alternative providers for almost 30 minutes. Eventually, all device energy is depleted and context reasoning failure occurs. The MTBCF of scenario 2 is consistent with the energy lifespan of simulated devices which is 300 seconds. In scenario 4, the availability changed every 60 seconds for the first eleven minutes, so the MTBF could not grow far beyond that even after availability simulation expired. Please refer to the following section for more discussion on system behavior in scenarios 2 and 4.

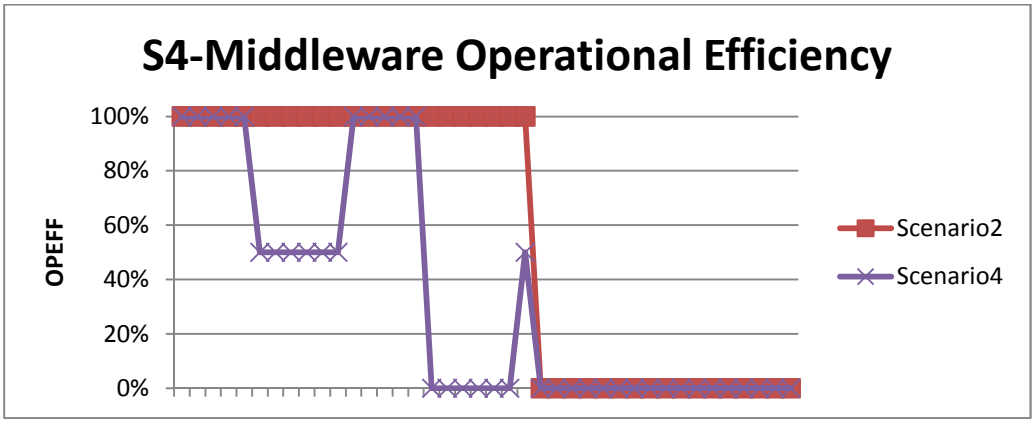

Figure 53 Snapshot 4- Operational Efficiency

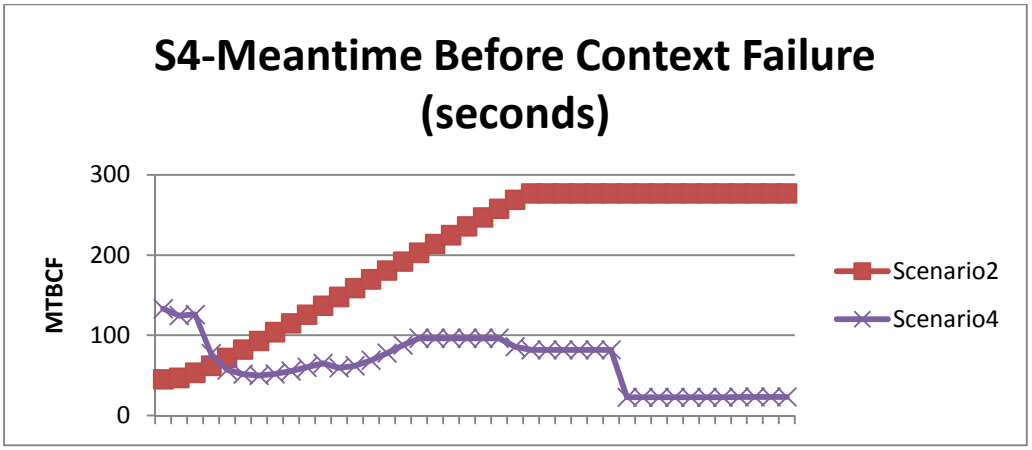

Figure 54 Snapshot 4- Meantime Before Context Failure

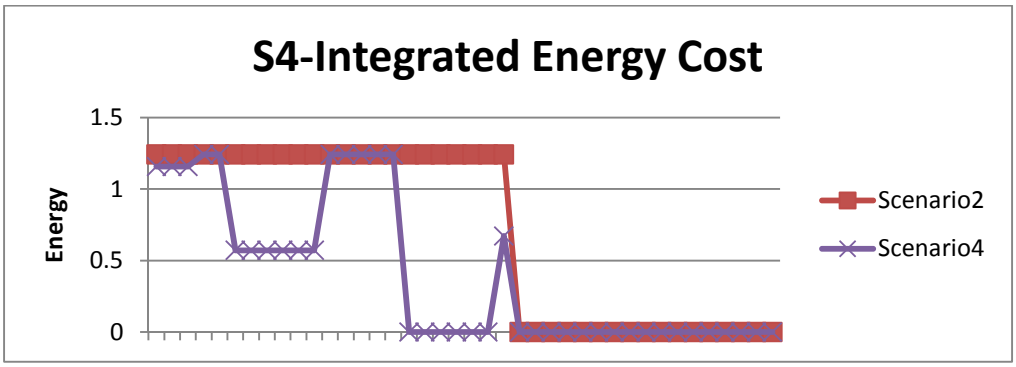

Figure 55 Snapshot 4- Integrated Energy Cost

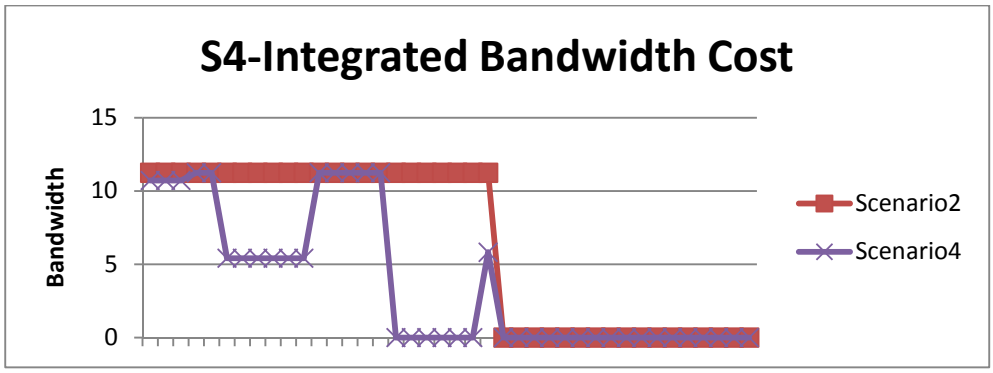

Figure 56 Snapshot 4 Integrated Bandwidth Cost

6.1.13 *Snapshot 5- System Startup + 205000ms*

Snapshot 5 graphs begin 1617079ms after system startup, covering 40, 11 second time intervals. This snapshot cover the period following system clean-up of inactive providers and cancellation of

consumer contracts which can no longer be served in scenarios 2 and 4. Context reasoning is sustained for nearly 30 minutes, adapting to use one the six alternative devices. When energy for all devices is depleted, OPEFF temporarily drops to 0, but returns to 100% after consumer obligation are cancelled. Also, all costs related to inferencing return to 0. The MTBCF remains at the levels obtained prior to permanent inferencing failure.

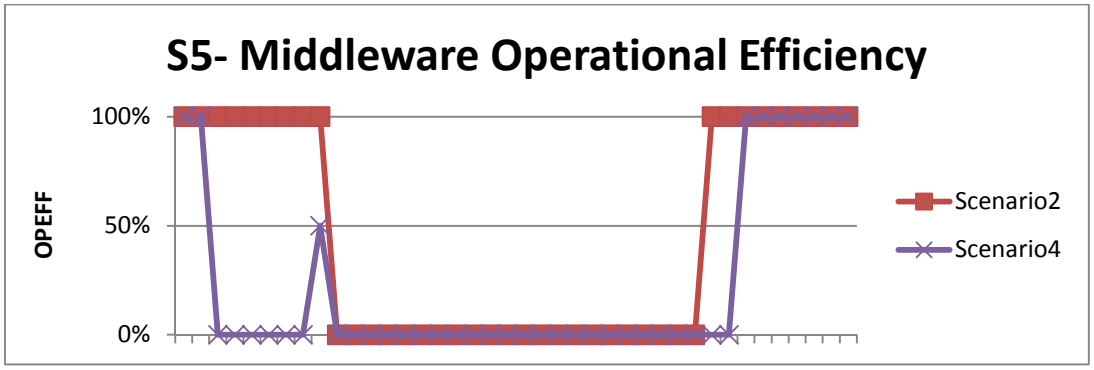

Figure 57 Snapshot 5- Operational Efficiency

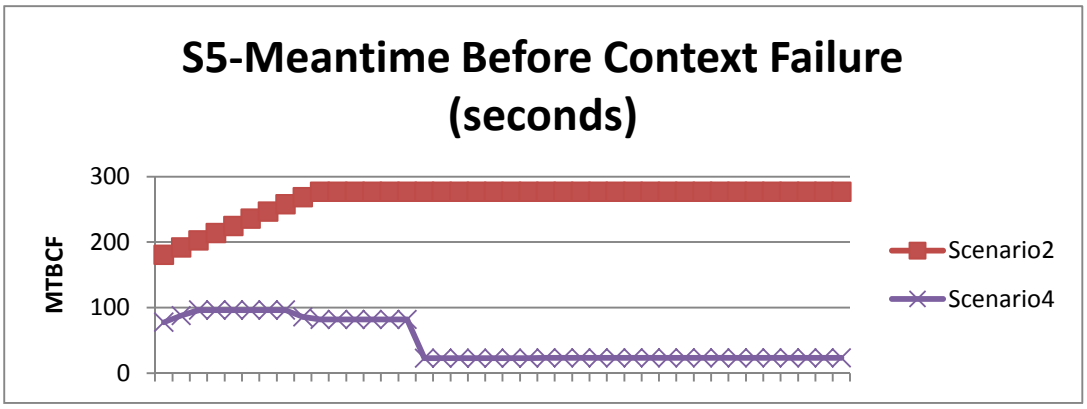

Figure 58 Snapshot 5- Meantime Before Context Failure

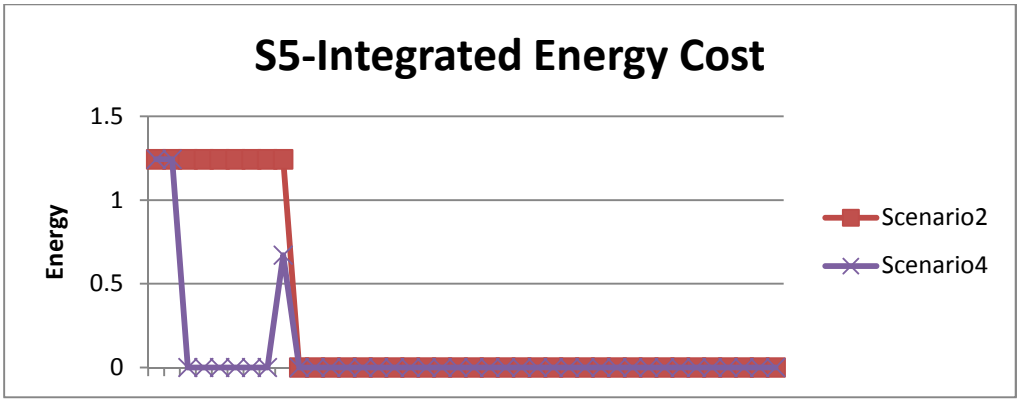

Figure 59 Snapshot 5- Integrated Energy Cost

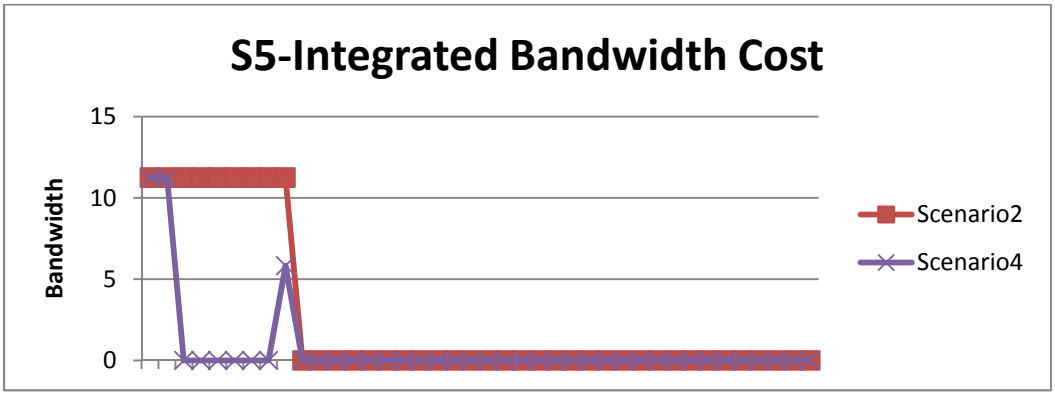

Figure 60 Snapshot 5 Integrated Bandwidth Cost

6.2 Hybrid Reasoning using Mobile Device Contexts

HyCoRE supports efficient context processing as it endeavors to sustain context by adapting reasoning components and employing context reuse where possible. There are elements of the architecture that are only used in specific scenarios. So we use several simple examples in our evaluation to show the generality of the framework for heterogeneous context and applications.

In this evaluation we stress the heterogeneous support in the architecture. We illustrate reasoning for physical activity. The previous law enforcement evaluation tested the architecture in cases where the low level devices drive the high level quality and system adaptation behavior. In this android evaluation, it is the intermediate reasoner quality that drives behavior. Figure 61 Mobile Device Context Illustration illustrates the scenario.

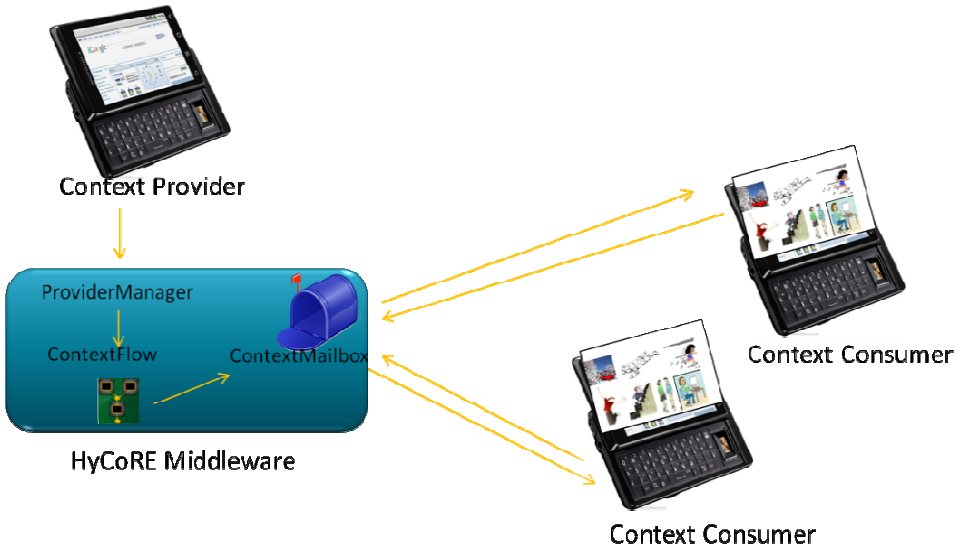

Figure 61 Mobile Device Context Illustration

One mobile device subscribes as an ‘activity’ context provider for a specific user target. Two other mobile users subscribe as consumers of ‘activity’ for the same target. Two trained classifiers (naïve bayes and J48 decision tree) are available as reasoners for inferring activity from a mobile device accelerometer samples. HyCoRE has an existing reasoning template for inferring ‘activity’ from accelerometer readings. The appropriate template is instantiated into a

context flow upon receiving a consumer request. The two consuming mobiles receive context notifications at their respective context mailboxes. Later, manual context feedback is used to degrade the quality of one classifier so that adaptation is triggered. HyCoRE adapts by replacing the failing classifier with its alternative; thereby sustaining context and maintaining quality. In summary, the middleware features demonstrated by this mobile device scenario are:

- ✓ Quality Integration
- ✓ Middleware Operational Efficiency
- ✓ Reasoning plan using asynchronous context push and synchronous context pull from providers
- ✓ Quality Feedback
- ✓ Adaptation to Mid- Level Context Provider Quality
- ✓ Context Sharing
- ✓ Context Mailbox Actuation

6.2.1 *Hybrid Reasoning using Mobile Device Contexts Demo Results*

This demonstration is in being completed at the time of this writing. Results will be presented in a separate work or as a supplement.

CHAPTER 7

CONCLUSIONS

This dissertation presents middleware solutions for efficient, effective and adaptable high level context reasoning. Architecture design and context reasoning solutions were evaluated through a prototypical implementation of HyCoRE: Hybrid Hierarchical Context Reasoning Engine. The novelty of HyCoRE begins with its data model and architecture design. It is specifically designed to bridge gaps in existing frameworks that hinder reusability. There exists many existing frameworks for deriving context in pervasive domains. However, there is still a need in the area of quality awareness and data generalization. Often, existing frameworks specialized in specific classes of context. Others, which are more general, include little consideration for quality. Solutions unique to HyCoRE include: i)integrated high level context quality derivation; ii)context provenance, reusable/adaptive reasoning plans; iii) quality of context middleware service measurements. Further, HyCoRE offers: i) context data models which are a refreshingly clear, generalized approach to modeling context, quality and cost; ii) (QoCS) Quality Based Middleware Performance Measures which serve as a basis for adaptive reasoning; iii) high level context quality integration and validation scheme; and iv) an adaptive reasoning scheme called context flows .This collective concentration of features supporting heterogeneous high level context derivation makes HyCoRE unique.

The current implementation is a fully functional prototype implemented in Java; and was used in evaluation of the aforementioned context middleware features. This is a extensible, generalized architecture that can be use to infer a variety of contexts. Our evaluation illustrates activity and location context inference. The QoCS measures demonstrated show that integrated quality metrics are a valuable way to measure framework reasoning performance; Also, that

quality is a sound basis for reasoning adaptation. The architecture can be extended to support more context types using pluggable context providers. It is the infrastructure that supports adapting components and measuring integrated quality that is the value of this architecture.

HyCoRE may be used in environments lacking sufficient reasoning infrastructure. By adapting existing devices to HyCoRE interfaces, a fully functional reasoning system can be achieved in diverse application scenarios, including: i) health/biometric situation monitoring ii)smart homes; iii) environmental monitoring; and iv) dynamic situation detection in mobile device field deployments. We envision future independent development of reasoning templates and generalized reasoners. The types of context that can be inferred with HyCoRE depends on these pluggable components. Pervasive domains are typically characterized by CPU, power and memory limitations. Lightweight reasoning components can be developed and used with HyCoRE in support of mobile and pervasive environments. The current prototype implements baseline processing for the core functions of reasoning plan projection, meta-data matching, adaptive context inference ,and quality feedback. These core algorithms can be improved to make HyCoRE more efficient. It would be a worthwhile effort to build HyCoRE to full scale with the robustness and versatility envisioned in the design. Also, the development of a repository of generalized reusable reasoning components for used in building context templates and flows could be a critical enabler for rapid context aware application development; a necessary step in achieving the vision of pervasive computing.

REFERENCES

1. Agostini, A; C. Bettini; D. Riboni, “A performance evaluation of ontology-based context reasoning,” In PERCOMW ’07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops, (Washington, DC, USA), pp. 3–8, IEEE Computer Society, 2007.
2. Agostini,A.; C. Bettini; D. Riboni, “Loosely coupling ontological reasoning with an efficient middleware for context-awareness,” In MOBIQUITOUS’05: Proceedings of the The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services,(Washington, DC, USA), pp. 175–182, IEEE Computer Society, 2005.
3. Bannach, D; O. Amft; P. Lukowicz, “Rapid prototyping of activity recognition applications,” IEEE Pervasive Computing, vol. 7, no. 2,pp. 22–31, 2008.
4. Beamon, Bridget, B.; Mohan, Kumar, J., “Adaptive Context Reasoning in Pervasive Systems”, 9th Workshop on Adaptive and Reflexive Middleware (ARM 2010), ACM Middleware 2010, November 30-December 2, 2010, Bangalore, India.
5. Beamon, Bridget, B.; Mohan, Kumar, J., “HyCoRE: Towards a Generalized Hierarchical Hybrid Context Reasoning Engine”, In Proceedings of the 8th Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010
6. Becker, C.; Schiele, G.; Gubbels, H.; Rothermel, K.; , "BASE - a micro-broker-based middleware for pervasive computing," Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on , vol., no., pp. 443- 451, 23-26 March 2003

7. Bettini, C. ; O. Brdiczka; K. Henricksen; J. Indulska; D. Nicklas; A. Ranganathan; D. Riboni, “A survey of context modelling and reasoning techniques,” Pervasive and Mobile Computing, June 2009.
8. Bisdikian, C.; Kaplan, L.M.; Srivastava, M.B.; Thornley, D.J.; Verma, D.; Young, R.I.; , "Building principles for a quality of information specification for sensor information," Information Fusion, 2009. FUSION '09. 12th International Conference on , vol., no., pp.1370-1377, 6-9 July 2009
9. Bisdikian, Chatschik; Joel Branch; Kin K. Leung; and Robert I. Young. 2009. A letter soup for the quality of information in sensor networks. In Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications (PERCOM '09). IEEE Computer Society, Washington, DC, USA, 1-6.
10. Bishop C. M., Pattern Recognition and Machine Learning (Information Science and Statistics). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
11. Blackstock, M.; R. Lea; and C. Krasic, “Evaluation and analysis of a common model for ubiquitous systems interoperability,” In Pervasive □08: Proceedings of the 6th International Conference on Pervasive Computing, (Berlin, Heidelberg), pp. 180–196, Springer-Verlag, 2008
12. Branch, Joel W. ;John S. Davis II; Daby M. Sow; Chatschik Bisdikian, "Sentire: A Framework for Building Middleware for Sensor and Actuator Networks," Pervasive Computing and Communications Workshops, IEEE International Conference on, pp. 396-400, Third IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'05), 2005
13. Bringel, Jose; Filho, Alina; Dia Miron; Ichiro Satoh; Jerome Gensel; Herve Martin, "Modeling and Measuring Quality of Context Information in Pervasive Environments," Advanced Information Networking and Applications, International Conference on, pp.

690-697, 2010 24th IEEE International Conference on Advanced Information Networking and Applications, 2010

14. Bruce, Henry; Giuseppe Raffa; Louis LeGrand; Jonathan Huang; Bernie Keany; Rick Edgecombe, "An Extensible Sensor based Inferencing Framework for Context Aware Applications, 10th IEEE International Conference on Computer and Information Technology, 2010
15. Buchholz, Qua T.; A. Küpper; M. Schiffers, “Quality of context: What it is and why we need it?”, *In Proceedings of the Workshop of the HP OpenView University Association 2003* (HPOVUA 2003), 2003.
16. Campbell, A.T.; Eisenman, S.B.; Lane, N.D.; Miluzzo, E.; Peterson, R.A.; Hong Lu; Xiao Zheng; Musolesi, M.; Fodor, K.; Gahng-Seop Ahn; , "The Rise of People-Centric Sensing," Internet Computing, IEEE , vol.12, no.4, pp.12-21, July-Aug. 2008
17. Chabris C F, Weinberger A, Fontaine M, Simons D J, 2011, "You do not talk about Fight Club if you do not notice Fight Club: Inattentional blindness for a simulated real-world assault" *I-Perception* 2(2) 150–153
18. Chen H., “An intelligent broker for context-aware systems,” In Adjunct Proceedings of Ubicomp, pp. 183–184, 2003.
19. Chen, H.; F. Perich; T.W. Finin; A. Joshi., SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications, In Proceedings of the 1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2004) Networking and Services, pages 258–267. IEEE ComputerSociety, 2004.
20. Chen, H.; T. Finin; A. Joshi., Semantic Web in the Context Broker Architecture. In Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004), pages 277–286.
21. Chen, H; T. Finin; A. Joshi, “The SOUPA Ontology for Pervasive Computing,” In Ontologies for Agents: Theory and Experiences, pp. 233–258, BirkHauser, 2005.

22. Chen, Z.; W. Zhe; Y. Liu; Y. Piao, “Intelligent home-hospital system based on context-aware technology,” in IIS ’09: Proceedings of the 2009 International Conference on Industrial and Information Systems, (Washington, DC, USA), pp. 23–26, IEEE Computer Society, 2009.
23. Cheng, Shang-Wen; David Garlan; Bradley R. Schmerl; Pedro Sousa; Bridget Spitznagel; Peter Steenkiste; Ningning Hu. ;, “Software Architecture-Based Adaptation for Pervasive Systems”, In *Proceedings of the International Conference on Architecture of Computing Systems: Trends in Network and Pervasive Computing* (ARCS '02). Springer-Verlag, London, UK, UK, 67-82.
24. Chetan, S.; Al-Muhtadi, J.; Campbell, R.; Mickunas, M.D.; , "Mobile Gaia: a middleware for ad-hoc pervasive computing," *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE* , vol., no., pp. 223- 228, 3-6 Jan. 2005
25. Clarke, M.;Blair G.S.;Coulson, G,;Parlavantzas, N.: “An efficient component model for the construction of adaptive middleware”, In Middleware '01, Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms , pp.160–178
26. Coppola, P.; Della Mea, V.; Di Gaspero, L.; Lomuscio, R.; Mischis, D.; Mizzaro, S.; Nazzi, E.; Scagnetto, I. & Vassena, L., “AI Techniques in a Context-Aware Ubiquitous Environment”. Pervasive Computing: Innovations in Intelligent Multimedia and Applications , pp. 157-180, 2009
27. Dargie, W. “The role of probabilistic schemes in multisensor context awareness, In PERCOMW ’07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops, (Washington, DC, USA), pp. 27–32, IEEE Computer Society, 2007.
28. Dey A. K., “Understanding and using context,” Personal Ubiquitous Computing, vol. 5, no. 1, pp. 4–7, 2001.

29. Dey, A.; Abowd, G. ; Salber, D., “A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications,” Human-Computer Interaction, vol. 16, no. 2, pp. 97–166, 2001.
30. Dourish,P.; “What we talk about when we talk about context,” Personal Ubiquitous Computing, vol. 8, no. 1, pp. 19–30, 2004.
31. Ermes, M.; J. Parkka; J. Mantyjarvi; I. Korhonen, “Detection of daily activities and sports with wearable sensors in controlled and uncontrolled conditions,” IEEE Transactions on Information Technology in Biomedicine, vol. 12, pp. 20–26, Jan. 2008.
32. Filho, J. ; Miron, A.; Satoh, I.; Gensel, J, and Martin H., “Modeling and Measuring Quality of Context Information in Pervasive Environments”, *In Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications*, IEEE, 2010.
33. Fröhlich, N.; A. Meier; T. Möller; M. Savini; H. Schuldt; J. Vogt, “Loca: Towards a context-aware infrastructure for ehealth applications,” Proceedings of fifteenth International Conference on Distributed Multimedia Systems, 2009.
34. Garlan, D.; Cheng, S.-W.; Huang, A.-C.; Schmerl, B.; Steenkiste, P.; , "Rainbow: architecture-based self-adaptation with reusable infrastructure," Computer , vol.37, no.10, pp. 46- 54, Oct. 2004
35. Garlan, D.; Siewiorek, D.P.; Smailagic, A.; Steenkiste, P.; , "Project Aura: toward distraction-free pervasive computing," Pervasive Computing, IEEE , vol.1, no.2, pp. 22-31, Apr-Jun 2002
36. Haghighi, P.D.; Burstein, F.; Al Taiar, H.; Arbon, P.; Krishnaswamy, S.; , "Ontology-based service-oriented architecture for emergency management in mass gatherings," *IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 2010*, vol., no., pp.1-7, 13-15 Dec. 2010

37. Hemmati , Hadi ; Jalili, Rasool;, “ Self-reconfiguration in Highly Available Pervasive Computing Systems” , In *Proceedings of the 5th international conference on Autonomic and Trusted Computing* (ATC '08), Chunming Rong, Martin Gilje Jaatun, Frode Eika Sandnes, Laurence T. Yang, and Jianhua Ma (Eds.). Springer-Verlag, Berlin, Heidelberg, 289-301.
38. Henricksen, K.; S. Livingstone; and J. Indulska., Towards a hybrid approach to context modelling, reasoning and interoperation, In UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management, pages 54-61, Nottingham, September 2004.
39. Hyun Jung La and Soo Dong Kim. 2010. A Conceptual Framework for Provisioning Context-aware Mobile Cloud Services. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing* (CLOUD '10). IEEE Computer Society, Washington, DC, USA, 466-473.
40. Indulska J.; Robinson R., “Modelling Weiser’s "Sal" scenario with CML,” In PERCOM ’09: Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications, (Washington,DC, USA), pp. 1–6, IEEE Computer Society, 2009.
41. Kang, Seungwoo; Jinwon Lee; Hyukjae Jang; Youngki Lee; Souneil Park; Junehwa Song; , "A Scalable and Energy-Efficient Context Monitoring Framework for Mobile Personal Sensor Networks," *Mobile Computing, IEEE Transactions on* , vol.9, no.5, pp.686-702, May 2010
42. Kang, Seungwoo; Youngki Lee; Chulhong Min; Younghyun Ju; Taiwoo Park; Jinwon Lee; Yunseok Rhee; Junehwa Song, “Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments”, In Proceeding of 2010 IEEE International Conference on Pervasive Computing and Communications (PerCom), 2010

43. Krummenacher, R; T. Strang, “Ontology-based context modeling,” In Workshop on Context-Aware Proactive Systems, 2007.

44. Kumar, M.; Shirazi, B.A.; Das, S.K.; Sung, B.Y.; Levine, D.; Singhal, M.; , "PICO: a middleware framework for pervasive computing," Pervasive Computing, IEEE , vol.2, no.3, pp. 72- 79, July-Sept. 2003

45. Lange, R.; N. Cipriani; L. Geiger; M. Grossmann; H. Weinschrott; A. Brodt; M. Wieland; S. Rizou; K. Rothermel, “Making the world wide space happen: New challenges for the nexus context platform,” In PERCOM ’09: Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications, (Washington, DC, USA), pp. 1–4, IEEE Computer Society, 2009.

46. Laskey, K. B. (2005) MEBN: A Logic for Open-World Probabilistic Reasoning. The Volnegau School of Information Technology and Engineering. George Mason University, Fairfax, VA, USA.

47. Lim, Brian Y. and Anind K. Dey. 2010. Toolkit to support intelligibility in context-aware applications. In Proceedings of the 12th ACM international conference on Ubiquitous computing (Ubicomp '10)

48. Liu, C.H.; Bisdikian, C.; Branch, J.W.; Leung, K.K.; , "QoI-Aware Wireless Sensor Network Management for Dynamic Multi-Task Operations," Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on , vol., no., pp.1-9, 21-25 June 2010

49. Lu, H; W. Pan; N. D. Lane; T. Choudhury; A. T. Campbell, “Soundsense: scalable sound sensing for people-centric applications on mobile phones,” In MobiSys ’09: Proceedings of the 7th international conference on Mobile systems, applications, and services, (New York, NY, USA), pp. 165–178, ACM, 2009.

50. MacLarty, Ian; Ludovic Langevine; Michel Vanden Bossche; and Peter Ross, “Using SWRL for Rule Driven Applications”, 2009 http://www.missioncriticalit.com/pdfs/rules-challenge-2009.pdf

51. Manzoor, Atif; Hong-Linh Truong; Schahram Dustdar, “Quality Aware Context Information Aggregation System for Pervasive Environments”, In Proceedings of the 23th IEEE International Conference on Advanced Information Networking and Application Workshops, IEEE, 2009.

52. Miluzzo, E.; N. D. Lane; K. Fodor; R. Peterson; H. Lu; M. Musolesi; S. B. Eisenman, X. Zheng; A. T. Campbell, “Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application,” In SenSys ’08: Proceedings of the 6th ACM conference on Embedded network sensor systems, (New York, NY, USA), pp. 337–350, ACM, 2008.

53. Nicklas, D.; M. Grossmann; J. Mínguez; M. Wieland, “Adding highlevel reasoning to efficient low-level context management: A hybrid approach,” In PERCOM ’08: Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications, (Washington, DC, USA), pp. 447–452, IEEE Computer Society, 2008.

54. Olsen, Dan R. Jr.. 2007. Evaluating user interface systems research. In Proceedings of the 20th annual ACM symposium on User interface software and technology (UIST '07). ACM, New York, NY, USA, 251-258.

55. Open Geospatial Consortium, “Sensor Model language,” http://www.opengeospatial.org/standards/sensorml.

56. Patkos, T.; A. Bikakis; G. Antoniou; M. Papadopouli; D. Plexousakis, “A semantics-based framework for context-aware services: Lessons learned and challenges,” In: Proceedings of 4th International Conference on Ubiquitous Intelligence and Computing, pp. 839–848, 2007.

57. Peizhi, L and Jian, Z., “A context-aware application infrastructure with reasoning mechanism based on dempster-shafer evidence theory,” In VTC Spring, pp. 2834–2838, 2008.

58. Pham, H.; J Mazzola Paluska; U. Saif C. Stawarz; C. Terman; and S. Ward, " A Dynamic Platform for Runtime Adaptation," Proc. Of PerCom, 2009

59. Preuveneers, Davy; Victor, Koen; Vanrompay, Yves; Rigole, Peter; Kirsch-Pinheiro, Manuele; Berbers, Yolande, “Context-aware adaptation in an ecology of applications”, Context-aware Mobile and Ubiquitous Computing for Enhanced Usability: Adaptive Technologies and Applications. Stojanovic, Dragan (ed.), pages 1-25, IGI Global, 2009

60. Rabiner, L., “Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition”, Feb. 1989

61. Riboni, D. and C. Bettini, “Context-aware activity recognition through a combination of ontological and statistical reasoning,” In UIC □09: Proceedings of the 6th International Conference on Ubiquitous Intelligence and Computing, (Berlin, Heidelberg), pp. 39–53, Springer-Verlag, 2009.

62. Riboni, D.; Pareschi, L.; Bettini, C. , "Towards the adaptive integration of multiple context reasoners in pervasive computing environments," Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on , vol., no., pp.25-29

63. Roussaki,I.; M. Strimpakou; N. Kalatzis; M. Anagnostou; C. Pils, “Hybrid context modeling: A location-based scheme using ontologies,” In PERCOMW ’06: Proceedings of the 4th annual IEEE international conference on Pervasive Computing and Communications Workshops,(Washington, DC, USA), p. 2, IEEE Computer Society, 2006.

64. Russell S. J.; Norvig, P, Artificial Intelligence: A Modern Approach. Pearson Education, 2003.

65. Salber,Daniel; Anind K. Dey; Gregory D. Abowd. 1999. The context toolkit: aiding the development of context-enabled applications. *In Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit* (CHI '99). ACM, New York, NY, USA, 434-441.
66. Schilit,B; N. Adams; R. Want;, Context-Aware Computing Applications. In Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications *(WMCSA '94*). IEEE Computer Society, Washington, DC, USA, 85-90.
67. Semantic Web Rule Language. http://www.w3.org/Submission/SWRL/
68. Seungwoo Kang; Jinwon Lee; Hyukjae Jang; Youngki Lee; Souneil Park; Junehwa Song; , "A Scalable and Energy-Efficient Context Monitoring Framework for Mobile Personal Sensor Networks," Mobile Computing, IEEE Transactions on , vol.9, no.5, pp.686-702, May 2010
69. Sheikh, K. ; M. Wegdam and M. van Sinderen, “Middleware support for quality of context in pervasive context-aware systems”, In Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops(PerComW 07), 2007
70. Staab ,Steffen; Studer, Rudi;, Handbook on Ontologies,International Handbooks on Information Systems. Springer, 2004.
71. Sudit, M.; M. Holender; A. Stotz; T. Rickard; R. Yager, “INFERD and Entropy for Situational Awareness”, Journal of Advances In Information Fusion Vol. 2, No. 1 June 2007, available at http://www.isif.org/4075D01.pdf
72. Suganuma, T; K. Yamanaka; Y. Tokairin; H. Takahashi;N. Shiratori, “A ubiquitous supervisory system based on social context awareness,” In AINA ’08: Proceedings of the 22nd International Conference on Advanced Information Networking and Applications, (Washington, DC, USA), pp. 370–377, IEEE Computer Society, 2008.

73. Vanrompay, Yves; Stephan Mehlhase; Yolande Berbers, “An effective quality measure for *prediction of context information”, In Proceedings of the 8th Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), ,2010*
74. W3C OWL Working Group, www.w3.org
75. Web Ontology Language. W3C OWL Working Group. http://www.w3.org
76. Weiser, Mark;, “The Computer for the 21st Century”, Palo Alto Research Center (PARC). Scientific American, September 1991, pgs. 94-104
77. Xiang.,Yang; ,Probabilistic Reasoning in Multiagent Sysems. Cambridge University Press, New York, NY USA, 2002.
78. XML Schema. http://www.w3.org/XML/Schema
79. Yau, Stephen S. ; Karim, Fariaz;, “An Adaptive Middleware for Context-Sensitive Communications for Real-Time Applications in Ubiquitous Computing Environments”, *Real-Time Syst.* 26, 1 (January 2004), 29-61.
80. Zhou, Jiehan; Ekaterina Gilman; Juha Palola; Jukka Riekki; Mika Ylianttila; Junzhao Sun;, “Context-aware pervasive service composition and its implementation”, *Personal Ubiquitous Computing* 15, 3 (March 2011), pp. 291-303.

BIOGRAPHICAL INFORMATION

Bridget Beamon graduated from the University of Texas at Arlington (UTA) with her Ph.D Degree in Computer Science and Engineering in August 2011. Her research interests include software systems architecture, context reasoning, knowledge management and machine learning. Before and during her Ph. D tenure at The University of Texas at Arlington she worked for Raytheon Intelligence and Information Systems in Garland Texas (2003-2009). She received a Master's degree in 2003 in Computer Science from the University of Texas at Dallas. Previously, she worked for Nortel Networks in Richardson, TX as a Member of the Scientific Staff (1995-2001). Bridget first received her B.S. degree in 1995 from UTA.