3. Conceptual Framework

The previous chapter has provided the background needed to identify the necessary issues and requirements of the research approach. Context-aware applications interact with contexts. By introducing constructive memory to context-aware applications, they can learn from those interactions with the context, determine contexts relative to a history of previous interaction experiences and take actions to further induce contexts without any human selection. A framework is provided of the component middleware technology approach, describing conceptual components of a modified architecture for context-aware systems incorporating a constructive memory. Section 3.1 discusses the issues that are recognised as important in framing the research. Section 3.2 explains the middleware requirements and the expectations for each requirement. Section 3.3 aims to provide a framework for describing the functional components that make up a constructive memory model for context awareness. Section 3.4 introduces the modified architecture approach and explains the functionality of each layer in the architecture in relation to the research. Section 3.5 offers an analysis of the modified architecture and identifies three requirements in how the architecture supports pervasive systems. Section 3.6 summarises the conceptual framework and reviews the main discussion points that emerge from this chapter.

3.1 Issues Related to Context Awareness and Constructive Memory

3.1.1 Alternative Definitions of Context

Having examined the definitions assigned by the pervasive computing community to the term *context*, some of the alternative perspectives adopted by other branches of computer science research are briefly contrasted. The principal goal is to illustrate the separation that exists between the various investigations of *context* and in doing so, to better position the research contributions of the thesis. This is intended to be illustrative rather then exhaustive, as the applications of context to computer science are extremely broad.

The concept of context is applied extensively within artificial intelligence. In machine learning, context is used to improve the effectiveness of learning algorithms. Widmer and Kubat (1993) described an approach in which forgetting of learned concepts are stored and later retrieved when the context reverts to the earlier state. Turney (1996) presents a review of a broader set of strategies for handling context-sensitivity in machine learning. This demonstrates that the types of context considered in learning tasks (termed contextual features) are highly domain-dependent. Examples of this can include lighting conditions in the case of digital image classification and the vocal inflections in the case of speech recognition. In the related domain of knowledge

acquisition, the explicit representation of context in knowledge bases is viewed as crucial to the success of expert systems. Context is variously encoded with knowledge bases as premises associated with rules, as metadata taking the form of context ontologies (Walther et al., 1992). Another related area of research examines the application of context to reasoning. The best known work in this area is of McCarthy (1986), who explored a form of reasoning to allow artificial intelligent systems designed for narrower contexts of use to be later generalised to be of broader relevance. McCarthy (1986) defined context only as abstract objects and did not address context-modelling methods.

The field of information retrieval aims to identify and present to the user documents that are relevant to the current context. The usual approach is to analyse the content of the documents viewed by the user, often by computing the frequency of terms that appear in the text, with the assumption that similar documents will also be useful. The extraction of contextual information only from viewed documents is clearly quite restrictive. Other approaches analyse patterns to generate profiles of user interests, which can be used in conjunction with the current browsing state in order to support a richer form of context awareness (Bauer and Leake, 2001). Brown and Jones (2002) investigated the use of the pervasive computing notion of context to further enhance information retrieval. They described the use of contextual information, including sensed data related to parameters such as location and temperature, and context triggers to support interactive and proactive document retrieval. In addition to this, linguistics and natural language processing define context as a tool for disambiguating sentence meaning. Two types of context are considered: the linguistic context which is determined by surrounding words and sentences, and the non-linguistic or situational context which includes the identity of the author or speaker, the purpose, the intended audience and so on (Connolly, 2001).

Each of the described disciplines has unique requirements in terms of types of context that are relevant. Moreover, these techniques are appropriate for acquisition and modelling contextual information. As a result, the various disciplines are largely independent of one another and are likely to remain so in the future. The research presented in this thesis attempts to align itself with a combination: the reasoning of context and the construction of context based on context information retrieval, to which the pervasive computing notion of context awareness can be applied.

3.1.2 Context-aware Lifecycle

As stated in Section 2.1, the context-aware lifecycle is defined by three processes: context discovery, context interpretation and context use in applications. Recent research on context awareness has largely focused on the development of software infrastructures that perform tasks such as the acquisition of contextual information from sensors, persistent storage, information within servers and the dissemination to applications. These infrastructures serve an important role, shifting much of the complex functionality from applications onto the middleware, and thereby simplifying the constructions of robust applications even in the face of evolving sensing infrastructures (Hong and Landay, 2001). However, these solutions exhibit a variety of shortcomings.

The main issue that context-aware systems need to consider is the amount of the information they will need to discover and to map to contexts. Most are founded on oversimplified models of context that fail to fully account for the challenges inherent in obtaining accurate contextual information from sensors, users and software agents. While it seems like the more information discovered and captured, the more helpful the system is, but this is not the case. In some situations such infrastructures have limitations. First, the emphasis on sensed context naturally biases them against providing strong support for static and profiled information. Second, the usual construction as distributed components, each concerned with only one type of contextual information, tends to lead to a fragmented context model. This makes it difficult or impossible to capture relationships and interdependencies to support tasks that span different types of context. Lastly, applications which use a rich set of contextual information are usually required to interact with multiple components. These problems can be overcome by introducing a layer of separation between the application and the context sensing infrastructure to form a context server. This server will maintain a rich, integrated model of context and provide applications with a single interface through which they can request contextual information. There is a further consideration of the fact that the activity of discovering contextual information might raise social fears regarding personal freedom and individual privacy rights.

The information models implemented by these context services exhibit various limitations. Most of the proposed infrastructures focus primarily on the discovery of contextual information from sensors or on the issues of storage and dissemination. There is therefore a need for solutions that combine these functionalities into a single and comprehensive infrastructure. Several, but most notably the Cooltown model (Kindberg et al., 2002), lack the formal basis that is required in order to capture contextual information in an unambiguous way and to support reasoning about its properties. The attribute-based model is unsuited to capturing historical information. The object-based models are less restrictive but provide no special support for the modelling of

histories either. Most of the solutions allow relationships to be expressed, but all fail to provide any special support for the modelling of dependencies or for related context tasks. Finally, all of the models fail to address uncertainty of context. Some allow quality metadata to be expressed, but the modelling of ambiguous information and the explicit modelling of unknowns have been overlooked. This last problem is crucial, as effective reasoning is impossible with the ability to distinguish between information that is absent from the context server because it is false, unknown or uncertain.

3.1.3 Characteristics of Contextual Information

Contextual information can originate from a wide variety of sources, leading to heterogeneity in terms of quality and persistence. While most of the previous research in context-aware applications focuses only on sensed contextual information, the most useful applications are usually those that combine sensed with non-sensed information. The non-sensed information is frequently obtained from users, often indirectly from user profiles or applications such as scheduling tools that record user activities. This research uses the sensed contextual information with non-sensed information that has been previously stored and recalled by the application. This non-sensed information aids in the prediction of a context based on the current contextual information being sensed. When sensing the current contextual information, the user is able to obtain information from the application by recalling previous experiences of a particular context.

Sensed contextual information is often dynamic and, even when not changing, is usually updated frequently in response to continuous or periodic sensor output. It is prone to inaccuracies as a result of sensing errors and at times may be completely unknown, owing to sensor failures, network disconnections or limitations inherent with sensing technologies. A common example of this is the inability of GPS receivers to function effectively indoors. In addition, when the context is changing rapidly, the delays introduced by distribution and the interpretation process that transforms the sensor output into high-level contextual information can lead to staleness.

User supplied contextual information can be partitioned into static and dynamic information. Intuitively, static information describes persistent properties such as the type of a computing device or communication channel. A high degree of confidence can be assigned to this type of information. In contrast, dynamic contextual information can suffer from staleness as a result of neglect on the part of the user to update the information as it changes. Unless obtained using indirect means, it is usually unreasonable to capture properties that change on a daily or even weekly basis as profiled information, because of the burden it places on the user. The characteristics of derived contextual information are largely determined by the properties of the

input data and of the derivation mechanism. Derived information usually retains any inaccuracies present within the input data. Table 3.1 lists the typical property types of contextual information.

Class of Information	Persistence	Quality Issues	Sources of Inaccuracy
Sensed	Low	May be inaccurate, unknown or stale	Sensing errors, sensor failures or network disconnections
Static	Forever	Usually none	Human error
Profiled	Moderate	Prone to staleness	Omission of user to update in response to changes
Derived	Variable	Subject to errors and inaccuracies	Imperfect inputs

Table 3.1. Typical properties of contextual information

Various sources of inaccuracies and error are characterised and the problem of imperfect contextual information is addressed more formally. An attribute of the environment covered by the context (such as the location of a given person at a specified time) is:

- unknown – when no information about that aspect is available;
- ambiguous – when several different reports about the attribute exist. An example is when two distinct readings are supplied by separate positioning devices. These readings may conflict or correspond to overlapping regions or simply describe the same location at different levels of granularity;
- imprecise – when the reported state is correct yet there are inexact approximations of the true state. An example of this is when the location of an object is known to be within a given region, but the precise bearings of the object within this region cannot be discovered; and
- erroneous – when there is a mismatch between the actual state of the environment and the reported environment. An example is when a user becomes separated from their device.

It is not always possible to overcome these levels of imperfect contextual information. However, context-aware systems can minimise the impact of imperfect information on applications by using appropriate context modelling techniques. Although an investigation into these context modelling techniques are beyond the scope of this research, only a small subset of the literature reviewed on context awareness for this research addresses even one of the classes. Context modelling techniques do provide a source of motivation for future work.

Previously, an important distinction was introduced between static (user supplied) contextual information and dynamic (sensed, profiled, derived) information. When dealing with the latter,

applications may not be solely interested in the current state but also in future or past states, or changes in the state over time. In order to support these types of behaviour, techniques for modelling and querying historical contextual information are required. The uses of historical memories of contextual information include the following:

- prediction – based on extrapolation from past behaviour;
- detection of changes – adaptation actions in response to context changes; and
- exploitation of a planned context – when available information about the user's future plans can serve as a useful type of context input.

A context model typically combines information about diverse yet interrelated entities, such as users, computing devices and the physical environments in which these reside. The model should capture relevant relationships between (such as the proximity to computing peripherals) as well as relationships that exist among different elements of the context description itself or, more precisely, between their states.

3.1.4 Constructive Memory and Reasoning

To support the modified context-aware architecture, a form of memory and a way of adapting that memory to store new experiences must be included. One way of considering memory is as constructive; memories that are always reconstructed in the moment of recollection with regard to the current situation. The system is not simply recalling the past but reconstructing the past constantly in terms of what it currently believes. Unfortunately, current generation mobile and embedded devices frequently suffer from resource limitations in general and are not equipped with infinite memories using unlimited storage capacity. Other limitations may include communication capabilities, battery lifetime and especially the processing power required to compute the reasoning and construct memory on the mobile device.

Some assumptions must be made about the importance of previous experiences in determining what is to be recalled. This memory model will be grounded in its experiences by implementing a memory system similar to that described by Saunders (2002). In this memory system, a distinction is maintained between the levels of memory; short-term and long-term memory. A short-term memory unit stores a number of percepts, actions and concepts for a short period of time to support communication between processes, time-based processing of inputs, and decision-making (Saunders, 2002). Long-term memory stores representations of previous experiences (Saunders, 2002). The implementation of constructive memory processes for sensing, perceiving, acting and effecting is domain specific.

In this research, a record in memory will store a number of percepts for a short period of time to support the processes in constructing the context based on the percepts it has discovered. Memory can be implemented simply as a store of recent percepts sensed by the various processes that interact with it. In this way, memory is represented as an accurate record of recent experiences that make up a context. To implement the constructive memory model for this research, a process is required that can store representations of previous experiences and their associations with other experiences. These previous experiences can be retrieved at a later time with a new experience and may only be a partial match to the original. This type of memory is called an associative memory and as the name suggests stores associations between stored representations. To store an association between two representations, they are presented together to the constructive memory model and it will learn a mapping from one to the other. In this research, an associative memory can be used to map between different percepts of an experience so they may be reasoned about, either in the same experience or in future ones.

Computer learning has generally stayed within the heuristic search or deductive reasoning paradigm, whereby most learning algorithms try to derive a set of rules from a body of data. Constructive memory and reasoning in its pure form differs from traditional learning methods in that no rules are ever created. There are three main advantages identified regarding why this approach is selected for this thesis. Firstly, deductions made by the system are achieved without the intermediate use of rules as a representation. Consequently, there are fewer opportunities for inadvertently introducing inaccuracies such as those that result from combining confidence levels. Secondly, rule generation has a combinational search space to contend with, while memory-based reasoning systems can focus their analysis on the cases at hand. It may be that to capture a large knowledge base, a large amount of rules are needed and therefore the rule generation process will never finish. There is never any certainty that the resulting rule set is complete and the resulting system may fail at any moment. With a constructive memory-based system, the data used as the basis of reasoning is always available. Therefore, within the limits imposed by the accuracy of the retrieval mechanism and contents of the database, errors of omission should not occur. Lastly, rule-based systems often include invented variables to insure proper sequencing of rules. Ordering problems and unexpected interactions are not a difficulty in constructive memory and reasoning-based systems.

Rule-based and constructive memory-based systems are limited by the databases they work from and by their retrieval methods. There will be cases in which a knowledge repository is too sparse to give meaningful answers, and there may be cases in which spurious correlations will give bad answers. In both classes of systems, statistical tests can guard against these problems. For both systems, there may also be cases in which the recall mechanism fails to find all the relevant data.

It is necessary to take steps to ensure such recall failures do not introduce systematic biases. There are several inherent advantages to memory-based reasoning, identified as:

- these systems require little or no training and begin to produce results immediately;
- they are understandable and can potentially explain why they act as they do; and
- they may inherently be more powerful that other forms of learning.

3.1.5 Summary of the Issues

A problem exists in training context-aware systems with all possible variations of contextual information and associated contexts. There is a need for a learning algorithm to help in this overall process and reduce the burden on the application and developers. A type of memory model is required to help provide the application with enough contextual information so it can recall and construct a context from its memory. This process is known as constructive memory and this model can be beneficial to the problem mentioned for context-aware applications. Typically such learning algorithms can be processing power and memory intensive on computers, let alone resource constrained devices like mobile phones. This is overcome by introducing the memory model as a component middleware technology for context-aware systems.

3.2 Middleware

In the computer industry, middleware is a general term for any programming that serves to *glue together* or mediate between two separate and often already existing programs. A much clearer definition of middleware is the software layer that lies between the operating system and the applications on each site of the system. The function of middleware is to mediate interaction between the parts of an application, or between applications. The role of middleware is to make application development easier, by providing common programming abstractions, by masking the heterogeneity and the distribution of the underlying hardware and operating systems

Middleware for context-aware systems resides between the fourth layer application component and the zero layer; that is sensors and actuators. The component middleware technology being introduced in this research belongs in the third layer of a context-aware architecture. This layer is known as the decision support layer in the context-aware architecture, but in the modified architecture it is the constructive memory query layer and is detailed later on in the thesis. This middleware must add many of the requirements of traditional distributed systems, such as heterogeneity, mobility, scalability and tolerance for component failures and disconnections. Additionally, it must protect users' personal information such as location and preferences, in accordance with their privacy preferences. Table 3.2 provides a summary of these requirements.

Support for Heterogeneity	Hardware components ranging from resource-poor sensors, actuators and mobile client devices to high performance servers, varieties of networking interfaces and programming languages, must be supported.
Support for Mobility	All components (especially sensors and applications) can be mobile and the communications protocols that underpin the system must therefore support appropriately flexible forms of routing. Context information may need to migrate with context-aware components. Flexible component discovery mechanisms are required.
Scalability	Context processing components and communication protocols must perform adequately in systems ranging from few to many sensors, actuators and application components.
Support for Privacy	Flows of context information between the distributed components of a context-aware system must be controlled according to users' privacy needs and expectations.
Traceability and Control	The state of the system components and information flows between components should be open to inspection in order to provide adequate understanding and controls of the systems to users.
Tolerance for Failures	Sensors and other components are likely to fail in the ordinary operation of a context-aware system. Disconnections may also occur. The system must continue operation without requiring excessive resources and to detect failures.
Deployment and Configuration	Distributed hardware and software of context-aware systems must be easily deployed and configured to meet user and environmental requirements, even by non-experts.

Table 3.2. The requirements for middleware for context-aware systems.

Context-aware systems consist of a variety of distributed components. Early systems are often constructed simply as distributed application components communicating directly with local and remote sensors. It is widely acknowledged that additional infrastructural components are desirable, in order to reduce the complexity of context-aware applications, improve maintainability and promote reuse. Figure 3.1 illustrates the distributed components found in many current context-aware systems. In addition to application components, sensors and actuators, shown at the two extremities in this layered diagram, include:

- components that assist with processing sensor outputs to produce contextual information that can be used by applications and map update operations on the higher order information back down to actions on actuators (layer 1);
- context repositories that provide persistent storage of contextual information and advanced query facilities;

- decision support tools that help applications to select appropriate actions and adaptations based on the available contextual information (layer 3); and
- programming toolkits are often also incorporated at the application layer (layer 4) to support the interactions of the application components with other components of the context-aware system.

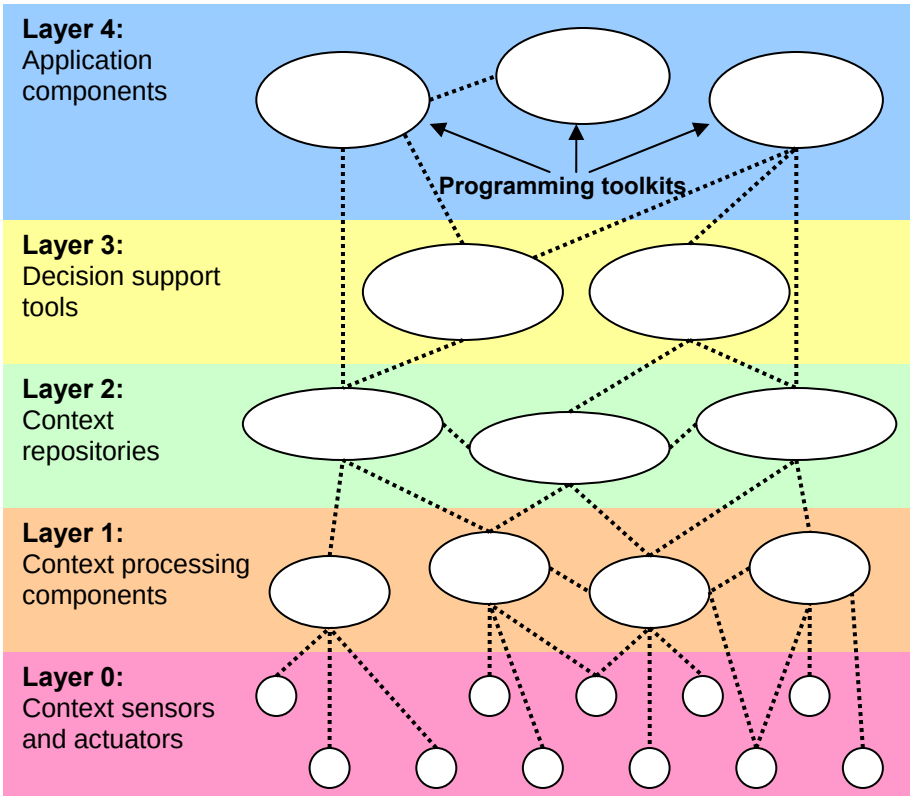

Figure 3.1. Components of a context-aware system (adapted from Henricksen et al., 2005)

A review and analysis are conducted on some of the proposed middleware solutions for context-aware systems. The focus is only on discovery methods for capturing contextual information and the solutions that span multiple layers of the system architecture, illustrated in Figure 3.1. Single layer solutions are excluded, such as context servers (layer 2), and models for context interpretation and use (layer 1). Additionally, solutions that are not general are excluded, such as those that only deal with location sensing and management.

3.2.1 The Context Toolkit

Revisiting Dey et al.'s Context Toolkit (2001) from the literature, it provides a set of abstractions that can be used to implement reusable software components for context sensing and

interpretation. The context widget abstraction represents a component that is responsible for acquiring contextual information directly from a sensor. Widgets can be combined with interpreters, which transform low-level information into higher-level information that is more useful to applications, and aggregators, which group related contextual information together in a single component (Salber et al., 1999). Finally, services can be used by context-aware applications to invoke actions using actuators, and discoverers can be used by applications to locate suitable widgets, interpreters, aggregators and services. The toolkit is implemented as a set of Java objects that represent the abstractions above. They provide a basic communication protocol based on HTTP and XML. The use of these Web standards allows for interoperation with components implemented in other languages, thereby providing basic support for heterogeneity. The toolkit's discoverers address component discovery, which is one of the requirements for mobility. However, the toolkit does not specifically address scalability, privacy, traceability/control, component failures, or deployment and configuration.

3.2.2 Context Fusion Networks

Chen et al. (2004) proposed the use of Context Fusion Networks (CFNs) to provide data fusion services (aggregation and interpretation of sensor data) to context-aware applications. CFNs are based on an operator graph model, in which context processing is specified by application developers in terms of sources, sinks and channels. In this model, sensors are represented by sources, and applications by sinks. Operators, which are responsible for data processing, act as both sources and sinks. Chen et al. (2004) implemented the CFN model in the form of Solar, a scalable peer-to-peer platform which instantiates the operator graphs at runtime on behalf of context-aware applications. The Solar hosts (Planets) support application and sensor mobility by buffering events during periods of disconnection; they also address component failures by providing monitoring and recovery, as well as preservation of component states (Chen et al., 2004). However, Solar does not yet address heterogeneity, privacy, or monitoring and control of the system by users.

3.2.3 The Context Fabric

The Context Fabric (Confab) proposed by Hong and Landay (2004) is primarily concerned with privacy rather than with context sensing and processing. Confab provides architecture for privacy-sensitive systems, as well as a set of privacy mechanisms that can be used by application developers. The architecture structures contextual information into *Infospaces*, which store tuples about a given entity. Infospaces are populated by context sources such as sensors, and queried by context-aware applications. Hong and Landay (2004) implemented the Infospace model using Web technologies, such that Infospaces are identified by URLs and tuples are exchanged in an

XML format. Privacy can be supported by adding operators to an Infospace to carry out actions when tuples enter or leave the space; for example, operators can be used to perform access control, notify users of information disclosure, and enforce privacy tags that describe how information can be used after it flows from one Infospace to another (Hong and Landay, 2004). Confab focuses on privacy and does not address distributed systems requirements such as mobility, scalability, component failures and deployment/configuration. However, it does partially address heterogeneity and provides privacy-related traceability and control via the operator mechanism.

3.2.4 Gaia

Gaia (2002) is designed to facilitate the construction of applications for smart spaces, such as smart homes and meeting rooms. It consists of a set of core services and a framework for building distributed context-aware applications (Román et al., 2002). Gaia's event manager service enables applications to be developed as loosely coupled components, and can provide basic fault tolerance by allowing failed event producers to be automatically replaced. Gaia's remaining four services support various forms of context awareness, and include (Román et al., 2002):

- context services, which allow applications to find providers for the contextual information they require;
- presence services, which monitor the entities entering and leaving a smart space, including people as well as hardware and software components;
- space repositories, which maintain descriptions of hardware and software components; and
- context filing systems, which associate files with relevant contextual information and dynamically construct virtual directory hierarchies according to the current context.

As smart spaces are typically small, constrained environments, Gaia does not address scalability. Similarly, privacy is not addressed by any of the basic services, but can potentially be provided by additional services, while user monitoring/control is outside Gaia's scope. Heterogeneity, mobility and component configuration can all be supported by Gaia in limited forms.

3.2.5 Reconfigurable Context Sensitive Middleware

Yau et al. (2004) proposed a Reconfigurable Context Sensitive Middleware (RCSM) for context-aware applications. The RCSM provides application developers with a novel Interface Definition Language (IDL) that can be used to specify context requirements, including the types of context

that are relevant to the application, the actions to be triggered, and the timing of these actions (Yau et al., 2004). The IDL interfaces are compiled to produce application skeletons; these interact at run-time with the RCSM Object Request Broker (R-ORB), which manages context acquisition, and the Situation-Awareness (SA) processor, which is responsible for managing triggers. The R-ORB provides a context manager that uses a context discovery protocol to manage registrations of local sensors and discover remote sensors (Yau et al., 2004). When a context-aware application starts up, the discovery protocol is used to look for local or remote sensors that satisfy the application's context requirements. The prototype described by Yau et al. (2004) does not satisfy the heterogeneity requirement, as it supports only C++ applications for the Windows CE platform. However, the IDL compiler can potentially be modified to produce skeletons for a variety of platforms and communication protocols. Additionally, the context discovery protocol is not flexible enough to support mobility or component failure, and Yau et al. (2004) do not attempt to address scalability, privacy or traceability/control. The main strength of the approach comes from the use of an IDL to specify context requirements. This makes it possible to incorporate new types of context and context-aware behaviour by editing and recompiling IDL interfaces, and partially addresses ease of deployment and configuration.

3.2.6 Middleware Summary

Based on the analysis, Table 3.3 summarises the capabilities of the surveyed solutions and demonstrates that comprehensive solutions do not yet exist. A further shortcoming, which is not revealed in the table, is that none of the solutions provide decision support tools (layer 3). In the middleware implemented in this research, a decision support tool is introduced in the form of a constructive memory query layer. This is represented as one of the *bubbles* in layer 3 of Figure 3.1.

Requirement	Context Toolkit	CFN/ Solar	Context Fabric	Gaia	RCSM
Support for Heterogeneity	P	N	P	P	P
Support for Mobility	P	C	N	P	N
Scalability	N	C	N	N	N
Support for Privacy	N	N	C	N	N
Traceability and Control	N	N	P	N	N
Tolerance for Failures	N	C	N	P	N
Deployment and Configuration	N	P	N	P	P

Table 3.3. Middleware requirements for supporting context-aware systems.[2]

[2] N.B. Key: C = Complete, P = Partial, N = None

3.3 Concept Formation

The aim of this section is to provide a framework for describing the functional components that make up a constructive memory model for context awareness. It must include some form of discovery system, memory-based storage, and a way of adapting memory to reason and store new experiences. A description of the components as an abstract architecture is provided. It represents all of the required functions to support the discovery and capture of contextual information, using the reasoning concept of constructive memory.

3.3.1 Environment

The environment is where the overall system resides. The environment is represented as a set W, of all possible environment or world states:

$W=\{w_1, w_2 \ldots w_n\}$

3.3.2 Sense Data

Sensors transform aspects of the external state of the world into a set of variables called sense data. Sense data is the name given to the variables provided by the environment that the system is able to sense. The range of sense data can be characterised as a set, S, of sensory states:

$S_D=\{s_1, s_2 \ldots s_n\}$,

Sensor data is represented as S.

3.3.3 Percepts

Perception is the process of extracting useful features and deriving percepts from the raw sense data. The range of percepts can be characterised as a set, P:

$P=\{p_1, p_2 \ldots p_n\}$

Percepts are represented as P.

3.3.4 Memories

Memory is constructive, which means that memories are always reconstructed in the moment of recollection with regard to the current context environment. That is to say, it is not simply recalling the past; it is reconstructing the past constantly in terms of the current percepts. The range of memories that are stored can be characterised as a set, M, of different memories:

$M=\{m_1, m_2 \ldots m_n\}$

Memories are represented as M.

Memories also act as a temporary storage mechanism for percepts before a context is actioned, based on the previous experiences of seeing those percepts.

3.3.5 Context

Context is characterised by the types of context the system stores in relation to different environment states. The ranges of context are not fixed, as one can add more contexts as the system evolves. Contexts are characterised as a set, C:

$C=\{c_1, c_2 \ldots c_n\}$

Context is represented as C.

3.3.6 Experience

The result of the overall process is stored as an experience and added to memory for retrieval when the system interacts with a new environment. The range of experiences that are stored can be characterised as a set, E, in memory:

$E=\{e_1, e_2 \ldots e_n\}$

Experiences are represented as E.

3.3.7 Overall Process

The descriptions above represent all of the required functional components of the system. A description of the system consisting of the overall processes for sensing, perceiving, effecting, constructing, storing and adding, can be illustrated like this:

$$W \xrightarrow{SENSE} S \xrightarrow{PERCEIVE} P \xrightarrow{EFFECT} M \xrightarrow{CONSTRUCT} C \xrightarrow{STORE} E \xrightarrow{ADD} M$$

The algorithm for constructive memory is $C = \sum P + E(P)$, where $E(P)$ represents all the experiences from memory where these percepts have been seen before in relation to the percepts being seen now, represented by P . The context constructed from the experiences is represented by C once the algorithm has completed the process.

3.4 Development of a Modified Architecture

Chapter 2 and Sections 3.1 and 3.2 have presented theoretical foundations for context-aware systems and constructive memory. The development of a modified architecture is built upon these foundations, as well as the concept formation, and details of each component are discussed. The design of the middleware architecture is inspired by the need to overcome

inherent challenges in pervasive computing related to resource limitations, scalability and the dynamic nature of computing environments. The architecture illustrated in Figure 3.2 follows a layered approach to the coupling of components and clean separations of concern. The lowest layers assume responsibility for context discovery, reception and management. The constructive memory layer offers an interface that allows the construction of a context, given a set of percepts. The following sections describe the functions and interactions of the layers in more detail. Additional layers not implemented but seen in Figure 3.2 are the adaptation and application layers. These layers are not the focus of this research and are beyond the scope of the thesis.

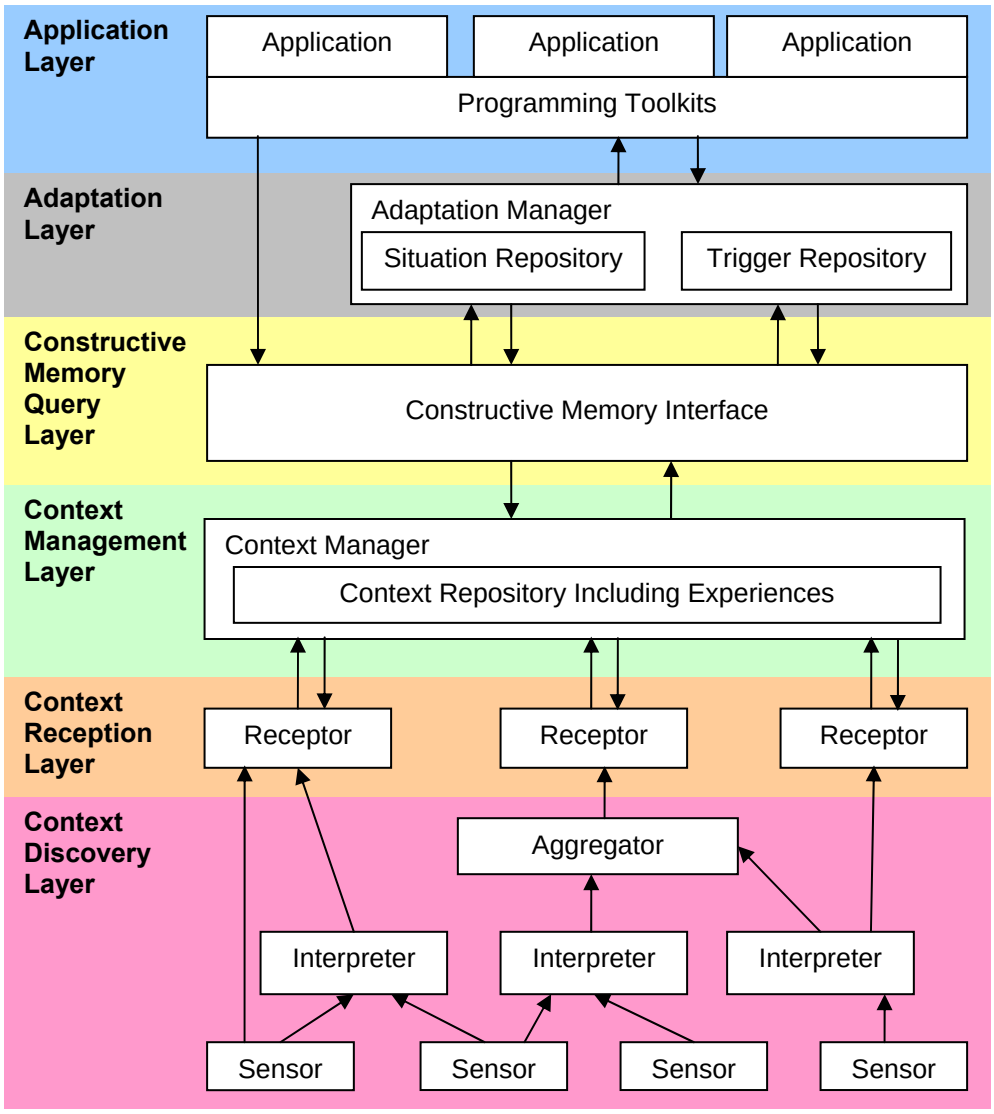

Figure 3.2. The layered architecture of our system.

3.4.1 Context Discovery Layer

The context discovery layer is responsible for the acquisition of sensed contextual information from the physical sensors and logical sensors (software on a user's workstation that monitors activity such as keystrokes on a keyboard). This layer takes the form of a distributed network of sensors and processing components, structured in a hierarchical organisation resembling that of the Context Toolkit (Salber et al., 1999) or Solar context discovery infrastructure (Chen et al., 2004). As the data derived from the sensors is generally far removed from the level of abstraction employed by the context management layer, processing occurs within a variety of distributed software components to raise the level of abstraction and perform fusion of data from multiple sensors. In the terminology of the Context Toolkit, these components are interpreters and aggregators. Sensor data may pass through a chain of processing components before arriving at the context reception layer, where it is integrated into the context repository.

The context discovery layer is implemented in the application prototype using Bluetooth and is explained in Section 4.1.2 in greater detail.

3.4.2 Context Reception Layer

The context reception layer forms the interface between the context discovery and management layers. Its main role is to perform a translation between the heterogeneous information produced by the sensing and processing components of the lower layer. As reception components will often need to integrate contextual information from a variety of resources, conflict resolution may be required. This is illustrated using the example of location information. User locations can be tracked using special positioning devices, wireless networking protocol information and swipe card logs. However, as individual readings are often imperfect or incorrect, for example, when the user is not carrying their positioning device, a history of readings must be examined and reconciled in order to obtain an accurate location estimate.

The context reception layer implementation for the prototype application uses Bluetooth as well. This process is further discussed in Section 4.1.2.

3.4.3 Context Management and Repository Layer

The context management layer acts as a repository of contextual information that serves many context-aware applications. As each application may have its own model of context, the management system is required to maintain many different models and potentially large quantities of information. The management system stores a common pool of contextual

information, as well as a setoff metadata corresponding to each context model. The metadata characterises the set of facts and constraints of each model and additionally provides a mapping of contextual information from the common pool to a view that conforms to the model.

The scalability challenges faced by the context management layer are significant, owing not only to the volume of information to be managed, but also to the highly dynamic nature of many types of contextual information and the need for rapid query responses in order to support timely adaptation to context changes by applications. The use of a common pool of contextual information, rather than a separate repository for each context model, is important, as it removes the need to duplicate shared contextual information, thereby reducing storage requirements. In addition to maintaining contextual information, it needs to maintain the histories of context. The context management layer must also allow users and applications to easily insert into, update and browse the contextual information.

The gathering and storage of contextual information introduces a variety of privacy issues. The context management layer assumes the primary responsibility for managing and enforcing the privacy policies that are needed to govern the discovery and storage of contextual information. However, in order to be effective, the policies must also be implemented within the context discovery and context reception layers.

The context management and repository layer is implemented in the application prototype using a record management system. Section 4.1.3 demonstrates the functionality of the application and overall process in more detail.

3.4.4 Constructive Memory Query Layer

In this layer the algorithm for constructive memory and reasoning on the context is presented. The constructive memory hypothesis is that reasoning may be accomplished by searching through the current database of past experiences of contexts for the best match to what the application sees right now. To implement the constructive memory layer, a process is required that can store representations of previous experiences and their associations with other experiences, such that they can be retrieved at a later time with a new experience that may be only a partial match to the original. This requires a means of judging how many current percepts match with previous percepts that the system has seen before. Once the algorithm has performed its calculation on the matching percepts per context type, it assigns a weighting to a context type which is converted to a percentage. The higher the percentage, the higher the prediction is that

the context type being constructed is most appropriate, based on what the system has perceived in that instance. Ultimately the algorithm is best described by this statement:

Constructed Context $= \sum$ What I see now + [What I have seen in relation to what I see now]

Another way to theorise the hypothesis is to understand it from a non-situated versus a situated point of view. A non-situated view of this algorithm is: based on a given a set of discovered percepts (X), all of these percepts are identified to be a perfect match to a particular known context type (Y). Therefore X is equal to Y. The situated view of this algorithm suggests that based on a given a set of discovered percepts (X), what else have I seen from our past context types in relation to the percepts just discovered? Therefore Y is a function of X because there might not be an exact match, but there might be partial matches with only a few percepts with previous context types. As an example, the following scenario is set up to illustrate the problem.

Sensed Percepts				Context Type
Desk	Filing Cabinet	Computer	2 Chairs	Office
Table	Projector	Whiteboard	6 Chairs	Meeting Room
Desk	Computer	Bed	1 Chair	Home

Table 3.4. Previous experiences of context and their associated percepts

In a non-situated view, if the exact same percepts (X) are not seen that correlate to the correct context type (Y), then the system cannot calculate or assign the context type because not all the percepts are available or known. However, from a situated viewpoint, the system senses the following percepts: table, projector, and five chairs. It can construct a context type from previous experiences stored in the database that this is potentially a meeting room, although not sensing the whiteboard and chair percepts. Additionally, it can assume the context type is a meeting room, due to the high rate of percepts that match comparatively to the other context types. Based on this example, Table 3.5 shows the number of percepts per context type from the database of experiences, the number of percepts that match with this scenario, and the percentage of successful matches to unsuccessful. It should be noted that a single percept denotes one item, device or object in the context. For example, the office context has two chairs; therefore they are counted as two percepts.

Context Type	Number of Percepts	Percept Matches	Percentage
Office	5	2	40%
Meeting Room	9	7	78%
Home	4	1	25%

Table 3.5. Results of constructing a context based on the matched percepts.

The constructive memory model proposed is sometimes also known as associative memory (Saunders, 2002). As the name suggests, associative memories store associations between stored representations. To store an association between two representations they are presented simultaneously to the associative memory and it learns a mapping from one to the other. An associative memory can be used to map between different features of an experience such that expectations of other contexts, either in the same experience or in future ones, can be envisaged. Additionally, this approach seeks to make decisions by remembering similar circumstances in the past. This decision making process is undertaken by counting combinations of features, using these counts to produce a metric, using the metric to find the dissimilarity between the current context and every item in memory, and retrieving the best matches. Overall, the reasoning process will degrade gracefully when it cannot come up with a definitive answer to a problem. It may respond that no answer is possible, give one or more plausible answers, or ask for more information.

The goal of the constructive memory layer is not so much to produce the best possible algorithm, as to produce one that is adequate to test the constructive memory reasoning hypothesis. Readers well versed in mathematical statistics or decision theories may object that this algorithm is ad hoc in the sense that it cannot offer any rigorous explanation of why it works. The point is well taken, and more rigorous examinations will probably produce a better algorithm. Nevertheless, this algorithm produces the right answer often enough to support the constructive memory hypothesis.

The constructive memory query layer is implemented in the application prototype using a memory-matching algorithm. The process by which the application computes this algorithm is described in Section 4.1.4.

3.5 Analysis of the Modified Architecture

An analysis of the modified architecture in this thesis and its underlying conceptual foundations, with regard to the key requirements of support for pervasive systems, has identified the following:

- dynamic computer environments, so that software opportunistically exploits changing sets of computing resources;
- scalability, such that large volumes of information (context), and a large number of applications and devices (including sensors), can be accommodated; and
- resource limitations, particularly in terms of battery power and network bandwidth in the case of the sensing devices and in terms of input/output capabilities in the case of user devices.

3.5.1 Requirement 1: Dynamic Computing Environments

The support of the modified architecture for dynamic computing environments is twofold. First, the adaptation of applications to changing environments is supported by treating resource characteristics as a type of contextual information. Second, the modified architecture itself supports evolution particularly of resources within the context discovery infrastructure. Context awareness has been widely exploited to allow applications to make opportunistic use of changing sets of computing resources. Scenarios that have been considered in this area include the use of contextual information to identify suitable printing services, and to adapt applications' communication patterns according to availability of bandwidth and power. These types of behaviour are easily supported within the framework of the modified architecture. Within the modified architecture itself, dynamic changes in the resources of the context discovery layer, caused, for example, by the addition or removal of sensors, are supported in two separate but complementary ways. First, the loose coupling in the lower layers, accomplished largely through the use of content-based routing to avoid explicit connections between components, allows context sensing and processing components to be added, removed and modified on-the-fly. Second, the separation of applications from their respective context models allows the latter to be evolved in response to changes in the context discovery infrastructure, with minimal impact. This can be typically achieved with only corresponding changes to the affected context definitions, and possibly also to some of the user preferences that are dependent on these. These are minor tasks in comparison to modifying the source code.

3.5.2 Requirement 2: Scalability

The requirement for scalability is inherent in pervasive systems. Consequently, scalability was factored into the design of the modified architecture in several ways. In the context discovery layer, the consequences of large numbers of components and frequent (and often continuous) sensor outputs are addressed through the use of selective forwarding of events using a content-based routing scheme, careful placement of processing components to enable data fusion and reduction at an early stage, and the use of the active update model for types of context in which the frequency of updates far outweighs the frequency of constructive memory queries. Within the context management layer, the storage and processing overheads associated with managing a separate repository of contextual information for each distinct context model are avoided by opting instead for a shared pool of information and a set of mappings from this pool to each of the models. This approach is combined with the use of selective distribution and replication in order to prevent bottlenecks and promote failure tolerance, as well as strategies of active and real-time databases in relation to frequently changing data. In the constructive memory query layer, scalability is dependent on the ability to efficiently evaluate and reason about experiences. The

constructive memory interface must handle issues of distribution, including disconnections and component failures, using strategies such as caching of query results and hoarding of frequently required contextual information, together with appropriate exception handling and reporting.

3.5.3 Requirement 3: Resource Limitations

Pervasive computing environments are characterised by the heterogeneity of computing devices and networking infrastructure. This implies that resource limitations must be tolerated and overcome in some areas of the modified architecture. This is particularly true at the two extremities in the modified architecture: the context discovery layer and the application layer. In general, the severest resource limitations are present within the sensing infrastructure. For most sensors, and particularly those that are mobile, the key factors are battery power and power intensive resources, including network bandwidth and CPU speed. Correspondingly, the modified architecture places minimal requirements on these resources. A lightweight Bluetooth communication model is used to transmit sensor data. A trade-off between CPU and bandwidth requirements is often necessary. Increasing the processing onboard the sensor can reduce bandwidth consumption and vice versa. Resource constraints with regard to the client devices on which user applications execute are generally less severe and of a different nature. Here, heterogeneity is often most noticeable with regard to input and output devices, owing to diverse contexts of use. As an example, in-car devices may rely mainly on speech-based input and output, while PDAs or mobile phones provide screen and pen-based interaction. As discussed previously, context awareness offers a natural way to overcome these resource variations. Bandwidth limitations and disconnections, as well as constraints on primary and secondary memory capacity, may also be problematic for some client devices. When this is the case, the distribution of functionality between the client device and other non-resourced-constrained devices can be examined. Memory requirements can be minimised by offloading the functionality of the query layers from the client device. Bandwidth requirements can also be reduced by this solution, as communication between the application layer and the query layer is typically less frequent that that between the latter layers and context management layer. This is because the invocation of the query can trigger multiple queries on the context repository.

3.6 Summary

This chapter has explained the conceptual framework for this thesis. It has discussed the issues that are relevant to framing this research and introduces the concept formation for the system architecture. Although this architecture is not entirely new, it demonstrates how a constructive memory layer is implemented in a context-aware system. The architecture still includes a

discovery and context management layer seen in traditional context-aware systems. Integrating a constructive memory model in context-aware systems requires the addition of only a few functions to monitor the discovery and reception of contextual information. The addition of functions to compare what is being discovered to what has previously been discovered can help determine an appropriate context. Taking actions to construct as opposed to recalling memory from previous experiences provides an alternative memory model that context-aware systems may have lacked in the past. Chapter 4 describes how this framework and modified architecture, incorporating constructive memory, are implemented in an application prototype.