INTELLIGENT HOME NETWORK

Architecture Paper for CS510
(Building Software Systems with Components)

Submitted on 12th of March, 2001

Shashidhar Lakkavalli(lakkavas@cs.pdx.edu)
Harkirat Singh(harkirat@cs.pdx.edu)

Department of Computer Science
Portland State University

Contents

INTRODUCTION

1.1 Introduction

Intelligent Home Network or (INTELHOMENET) is a network of home and office gadgets and any controllable devices, which affect the day-to-day life of people. With emerging technologies like Bluetooth, 3rd generation wireless, Personal Digital Assistants and smart mobile phones capable of providing ubiquitous connectivity, numerous applications come to the fore. In this paper, we propose an architecture adopting the above said emerging technologies to provide a smart network of home appliances, capable of reacting to user's personal requirements.

The "First Generation" of home networking has emerged to allow the sharing of files, printers, and Internet connections, and to enable networked PC games. The second "Second Generation" of home networking will be based on wireless technologies as communication media.

Compared to traditional networked environments, the home networking environment is more heterogeneous because of different types of consumer devices / appliances, manufactured by different vendors. In such scenarios, development of applications to encompass all these devices, requires standard interaction models.

With myriad kinds of devices, the PDA software must be scalable and substitutable enough to function with all the devices. We need standard interfaces, which all the different devices can use to contact the PDA and vice-versa. We therefore think that component software model is the best methodology to use to implement such a system, which enables to build our system based on interfaces, with which devices can interact.

We have identified dependability, extensibility, user-friendly interface, and remote access capability as the four key requirements useful for successful home networking. *Dependability* ensures that failure of hardware devices and software objects will be detected, appropriate recovery of cleanup operation will be performed, and homeowners will be alerted if necessary. *Extensibility* allows any new device to be added at the ease of user and becomes available to all existing applications. INTELHOMENET is based on wireless technology that operate on radio frequency, hence, no extra wiring is required for addition of new device and it is plug-in type. User-friendly interface allows users to control appliances.

We propose a Soft-State(context specific information) to be preserved by devices and which will expire if not refreshed within a predetermined, but configurable amount of time. Centralized Authority will periodically send keep-alive messages and if a devices fails to receive beacon it will go to sleep mode.

The paper is organized as follows. Section 2 identifies problems we set to solve by this architecture. Here we explain our concept of a context. Section 3 describes the design and overall architecture of INTELHOMENET. Section 4 describes how component based design meets the requirement. Section 5 explains key interfaces with relevant UML diagrams. Section 6 analysis interface reuse w.r.t our architecture. Section 7 describes about reference implementation, and section 8 summarizes the work and future work to be done in INTELHOMENET. The appendix contains UML diagrams.

2 Problem Statement

We have identified 3 important problems to be overcome to implement our architecture. They are

1. How to form a network of devices
2. How can different types of devices coexist
3. Who is the controller

We tackle each of these problems below.

2.1 How to form a network of devices?

The most important problem is how to form a network of devices. We could have all the instances of all different types of devices under one network, or we could separate out these devices depending on a criterion. The basic assumption of our concept is that the behavior of the devices should change with the location of the user and the time of the day. Having a single network for all the devices at home is not a feasible solution if we consider that the access technology will be either Bluetooth(802.15) or Wavelan(802.11) technologies, which rely on line of sight communications. So, their range is limited to a room. This constrains us to form a network of devices within a room. The configuration for the devices within a room is called a context.

Solution : A context symbolizes any enclosed physical space like a room. The constituents of a context are the appliances/devices present in the physical space represented by the context. A context also maintains device specific information, reflecting a physical device's parameters, characteristics and behavior. The device parameter values are user configured, to control the behavior of these devices, to suit user's preferences.

The type of devices, which we are dealing here are heaters, lighting, printers, televisions, radio etc., which need to have different working states, depending on 2 factors.

They are

a. The presence of user.
b. The time of the day

Some of the devices' behavior are affected by both of these factors. They include heaters, which needs to change their temperature settings when the user is the room or away. Also, it is time dependent because when the user is asleep, the temperature is normally lower than when he is awake.

Some devices are affected by the presence of user only. The devices include printers, TV's, VCR's and other multimedia devices and independent from time.

The parameter values of the devices are therefore affected by these 2 factors, and the context uses them as the basis of affecting a change in the devices' parameters. Since a context is configured by the user, it reflects his/her preferences.

Every physical space like a room is represented by a context. So, in a house for example, there is more than one context.

2.2 How can different types of devices coexist ?

A home network contains numerous kinds of devices and we need to form a network of these devices. Our architecture should be scalable enough to accommodate different kinds of devices.

Solution : We propose that each device type be represented by a device driver. The device driver needs to support a standard interface to work with INTELHOMENET software and its implementation details are specific to the component. Thus, substitutability and reusability is achieved.

2.3 Who is the controller ?

Solution : Since the context settings change with the presence of user and time, a device which is close to the user most of the time is best suited to carry the settings. The obvious choice is a mobile phone or a Personal Digital Assistant (PDA). This also has the advantage that with the PDA not only can we control home appliances, but also control devices outside a home, like the user's office. So, this becomes scalable.

There is another reason for mobile devices to be the best choice of embedding the software. Since mobile devices contain different applications and ability to interact with the internet, the cellular network, there are tremendous opportunities of our software to make use of these other applications. Also, having information about the status of the devices, close to the user is always useful for him to take appropriate actions.

In our architecture, we therefore assume that a mobile phone or a PDA is the carrier of context.

3 Architecture

3.1 Overview of Architecture

Our architecture is based on manager-agent and client-server paradigms. A device driver residing in the PDA is a manager, while the agent is the software residing in the devices. The manager initiates requests and commands, while the agent sends responses and also sends notifications in case of special cases like faulty behavior etc.,

It is also a client-server architecture because the different software components within the PDA provide services to others, while at the same time requiring services from other software components.

The standard interfaces are

1. Communication interface
2. Database access interface
3. Command interface

Communication interfaces : The communication component supporting these interfaces enable the PDA and the devices to communicate.

Database access interfaces : These set of interfaces provide the context and device drivers access to context specific and device specific information stored in PDA's.

Command interfaces : These interfaces delegate responsibility from the UI to context to the device drivers.

3.1.1 Components

List of the components are:

1. Context
2. Device Drivers
3. Database handler
4. Timer
5. Locator
6. Communication
7. Device agents
8. User Interface module

3.1.2 Design Concept

Our architecture is based upon a context, context specific information and action is initiated and controlled by the context component. It aggregates a set of device drivers, delegating device specific responsibility to the concerned device driver. Since it requires context specific information, it also needs the services of the database handlers. Thus the context is a client for device drivers and the database interface.

A device driver is an independent self containing software module, like a DLL. A single device driver represents all instances of a single device type in a context. For example, if there are 2 printers in a room, then one single printer device driver suffices. If there are no instances of that device type, then the device driver is not instantiated. A device driver requires information like the number of instances of the device present in that particular context, the parameter values for each instance of the device. It stores and retrieves this information from the database. Since a device driver is also a manager over all its instances, it needs to communicate with its instances. It uses the services of the Communication component for this purpose.

The Communication component contains a receiver and transmitter, basically using the wireless interface. Some of the examples include bluetooth or 802.11 standards compatible devices. If it is bluetooth, then bluetooth chipsets are embedded in both the PDA's and the devices. If it is 802.11, wavelan cards need to be part of the devices. Since bluetooth chipsets are much smaller and cheaper than the wavelan cards, we propose that bluetooth provides the best communication interface

Other important components are the Database handler, Timers and Locators. The database is some sort of permanent storage where context settings and device parameters are stored. Timer provides stop clock functionality and it is used by the devices, to activate new settings to devices. One instance of timer is required per device driver. It is also used to periodically send keep alive messages to devices. They serve 2 purposes. If keep alive messages are not coming, then it means that the user is not around in that context. Therefore, it is better to change settings to default values, which could require far less power consumption than otherwise. For example, if the user is not around, then the heater temperature can be reduced. Also, it enables battery operated devices if any to go to sleep mode.

Locator components in the PDA enable the context component to load new contexts, depending on the location of the user. Locators could be voice detectors, where the user will specify the name of the context(in our case the room) and it will interrupt the context to load the new context. Alternatively, it could be a receiver like device responding to sensors, which detects that a user has entered the room and then it will interrupt the context to change the context.

4 Motivation for Component Object Model

4.1 Technological benefits of using Component Object Model

Home networking is dynamic in nature as new devices / appliances will be added and they are manufactured by different vendors. COM offers enormous potential advantages to software developers and end-users:

- *Reusability*: INTELHOMENET will be using the services of the Device driver components, which are developed and tested by third party. This not only reduces the development time but also results in higher quality product.

- *Efficiency:* Applications do not duplicate functionality. Instead, they can present a uniform user interface where new functionality can be added incrementally and integrated smoothly into the familiar environment. This can reduce training costs significantly while achieving a much higher level of satisfaction of the user. It is important to mention here that user of the INTELHOMENET could be a naïve computer user. With incremental update, the user can focus on what is new in the system without worrying about changes of overall environment. It is very important for customer confidence.

- *Contract*: The physical implementation of a component is hidden, and only accessed via, an interface. In this way, changes to the implementation of the components are isolated from the application that uses it. It gives flexibility to user to choose DLL's (Device drivers) from a group of providers.

- *Versioning problem*: COM's approach to versioning is based on the following three requirements: first, any interface (identified by an IID) must be immutable. Second, a new implementation of the same CLSID must support existing interface. Finally, any client must start interacting with a server by querying an interface with an IID. Such a combination allows independent evolution of client and server software.

  Suppose, INTELHOMENET component is upgraded before the client is (for an example device drivers component in PDA). The reason for this could be to add more features due to fierce market competition. Since the new server supports all the old interfaces, the old client can still obtain all the interface pointers that it needs and run smoothly. When the client software is also upgraded, the new client will query the new interfaces to enjoy the new features. In contrast, suppose the client software is upgraded first. The new client will try querying the new interfaces on the old server and fail. This procedure forces the new client to handle the failure by, for example, providing only old features. But it will not cause the new client to crash or unknowingly execute incorrectly.

4.2 Commercial benefits of Component Object Model

- *Low manufacturing cost*: It is not mandatory for a Device Driver (Device Component) to be manufactured by Original Equipment Manufacturer (OEM) hence COM reduces dependability between service provider and user. This gives freedom to negotiate with group of third party developers (who supports the contract) to get a best deal, which in turn helps in low project cost.

- *Freedom to choose a devices*: Aforesaid feature also benefits end-user by providing him/her competitive market for a device/appliance. It also gives flexibility to choose a physical device OEM and device dll separately.

- *Addition of new devices*: Suppose user is installing new printer and existing PDA.EXE supports IPrinter interface then at no additional cost printer can become part of existing network.

4.3 Drawback of Component Object model

We feel that it is new field and one major constraint of component technology could be that very exerts are available for the development of maintenance of Project.

5 Key Interface and UML Diagrams

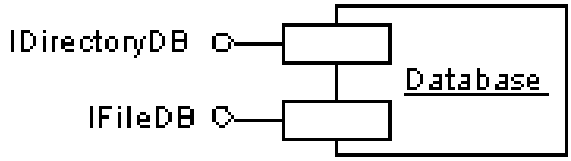
5.1 Details of interfaces

For higher level overview of various classes, interfaces and components please refer Fig 1, 2 & 3.

5.1.1 Database

It has 2 interfaces:

1. IDirectoryDB
2. IFileDB



5.1.1.1 IDirectoryDB:

This interface is implemented by the Database component and used by the Context component. The Context component uses this interface to get a listing of active contexts or devices for the current context.

Methods supported are:

1. opendir([in] BSTR* pPathforContext,[out] int *fd)
   This method opens a directory under /context. /context is the root of the database.
2. closedir([in] int fd)
   This method closes the file descriptor.
3. readdir([in] int fd, [out] BSTR*buffer)
   This method will read the current directory listing and return into buffer.
4. adddir([in] int fd, [in] BSTR *pContextName)

This method adds a new context in the database. This context is still not configured

5. getActiveDevices([in] BSTR *pContextName)
   Given a context name, it retrieves the list of device types currently active under the context.
6. deleteDirectory([in] BSTR * pDirName)
   This method will delete a directory corresponding to a context.

5.1.1.2 IFileDB

This interface is used by the Driver components. It provides file operations like open, read, write and close. This interface is used to read files containing the parameters for each instance of a device type.

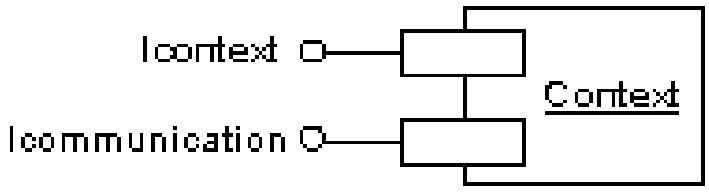
Methods supported are

a. open ([in] BSTR* pFileName, [in] int permissions,[out] int *pfd)
   This method opens a file.
b. Read ([in] int fd, [out] BSTR*buffer)
   This method reads the contents of a file into a local buffer.
c. Write ([in] int fd, [in] BSTR *buffer)
   This method writes new set of values for an instance. These values are set by the user either during configuration or using the Edit context option.
d. close([in] int fd)
   This method closes an instance's parameter file.

5.1.2 Context

This component contains 3 interfaces. These interfaces are callback interfaces, as the UI, Locator and the Communicator components will invoke the context methods asynchronously. The interfaces supported are :

1. IContext
2. IcommunicationContext



5.1.2.1 IContext:

This interface is used by the UI component to invoke commands into the context. It is also used by the Locator component.

Methods supported are

a. ActivateContext ([in] BSTR *pContextName)
This method is called, when the user wants to activate a context. This method will retrieve the list of devices active for the context and will invoke activateContexts () method on the devices using IcontextDriver interface, refer Fig 4.

b. EditContext ([in] BSTR *pContextName)
This method is called, when the user wants to edit a context. This method will call IcontextDB's getActiveDevices() method to show to the user the list of devices present. Please refer Fig 9.

c. displayContexts()
This method is called, when the user wants to see a list of contexts currently existing in the PDA. This method invokes IcontextDB's readdir() method to retrieve the list of contexts.

d. createContext()
This method is called when the user wants to create a new context. This method will open a dialog box for the user to key in the context name. This inturn will use the IcontextDB's adddir() method to make an entry in the database. Please refer Fig 5.

e. deleteContext([in] BSTR *pContextName)
This method will remove the entire directory corresponding to a context. Please refer Fig 6.

5.1.2.2 IcommunicationContext

This interface is used by the Communication component as a **callback interface**, to inform the context of any notifications, responses or new registration messages coming from the devices. This interface will have to decode the message type and then take suitable action. These methods delegate the functionality to the respective devices. This interface implementation maps the device type in the message and appropriately delegates to a device driver.

Methods supported are:

a. notification([in] BSTR *pdeviceType, [in] int deviceInstance, [in] BSTR * pmessage)

This is a notification method indicating an error like FAULTY event occurred in the device. The context will have map to the corresponding device and then passes the message to notification method on IdeviceDriver() method. This is a delegation. Please refer Fig 7.

b. Registration([in] BSTR *pdeviceType, [in] BSTR manufacturerID, [in] int yearOfManufacturing, [in] int productNumber, [in] int serviceContractNumber, [in] BSTR * pYearofWarrantyExpiry)

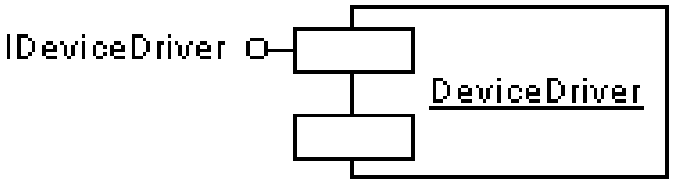
This registers a new device. This is pending configuration, before it can be activated. This will use the IfileDB methods to save this in a file named “system”, under the device.

c. Response([in] BSTR *pdeviceType, [in] int deviceType, [in] BSTR *pmessage)

This method is called as in response to a request sent by the device driver. It delegates the functionality to response() method on IdeviceDriver, passing the message.

5.1.3 Device Driver

This component represents a device. There is one instantiation of the DeviceDriver for one type of device. It handles multiple instances of the device type. It contains the following interfaces.



5.1.3.1 IdeviceDriver:

This interface is used by the context component. It implements the functionality specific to a device type. It is the only component, which knows the parameter names and values. Methods listed below, require the use of IFileDB interface. It has the following method names

a. activateContext([in] BSTR *contextName)

A device driver represents all its instances. Each instance might have different settings and it should send requests on each of them accordingly. We use the following procedure to enable the device driver to manage multiple instances.

1. When an activate command is received from the context, it instantiates the database component and uses the IDriverDB interface to load the context specific information for that device.
2. It will load parameter settings for each instance containing parameter values for different durations for one whole day.

For example, a heater instance will have the following settings.

Device Instance	Start Time (Hours)	End Time (Hours)	Temperature (Fahrenheit)
1	0	6	60
1	6	9	75
1	9	18	50
1	18	24	75

3. It merges these settings for each instance and then sorts them on time. For 2 instances of heater in a single context, the merged settings is shown in the table below.

Device Instance	Start Time	End Time	Temperature
1	0	6	60
2	0	8	60
1	6	9	75
2	8	9	75
1	9	18	50
2	9	20	50
1	18	24	75
2	20	24	75

The end time is not required and it is only shown for granularity here.

4. Depending on the current time, it initializes to one of the entries above. From then on, it follows the table entries accordingly.
5. It instantiates a timer, for the first time out. In our example, if the current time is 19 hours, then the timer is set for 1 hour, after which it should set new temperature values to timer 2.

b. editContext([in] BSTR *contextName)

This method is for the user to change a particular instance's settings for a context.

c. displayContext()

This method is used to display a list of active instances and their parameters

d. callbackforTimer()

This is a callback method called by the Timer when it times out. Apart from using the timer for waking up the device driver to send new set of values to a device, it is also used to periodically wake up the device driver to send keep alive messages. Keep alive messages are sent by PDA's because it is possible that some of the appliances are battery operated and therefore if the user is not in the context, they can go to sleep mode, thus saving power. Also, these devices are woken up by the PDA's again.

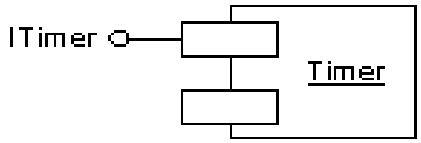
e. notification([in] int deviceInstance,[in] BSTR *message)

This method is invoked by the Context component. This method will decode the type of error from the message and will then change the database for that instance. As an optimization, it can notify the user with an email!!! Please refer Fig 7.

f. response([in] int deviceInstance, [in] *message)

This method handles a response from the device. It periodically retransmits a request and if a response is not forthcoming. Once the response is received, it will reset the timer.

5.1.4 Timer

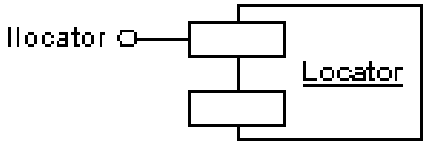


5.1.4.1 ITimer:

A timer component is instantiated by each device driver. It has a single interface, Itimer and device drivers register with this interface. ITimer contains notifyTimeOut method, which will call the callbackforTimer method of IdeviceDriver.

5.1.5 Locator

5.1.5.1 ILocator



This component is used to indicate a change in location to the context component. It has a single interface, Ilocator to which the context will register. Ilocator contains notifyChangeContext method, which will invoke activateContext() method on the Icontext interface of the context. Please refer fig 8.

5.1.6 Communication

This component constitutes a Receiver, transmitter, decoder and encoder. The receiver will receive notifications, responses and registration messages in form of an encoded form and it will decode the message and will then forward the requests to the context. In

case of the transmitter, it will encode the messages sent by the Device drivers and will transmit it over to the receiver on the other side.

6 Interface Reuse

We developed INTELHOMENET keeping in mind that it should be scalable and reusable enough to work with myriad kinds of devices. It is always possible for user to add new device any time in future. INTELHOMENET will be able work seamlessly with different kind of devices as long as device driver supports IdeviceDriver interface. Implementation of IdeviceDriver is device specific hence this interface hides the actual implementation of it. This gives flexibility to user to dynamically configure INTELHOMENET. We see polymorphism as IdeviceDriver interface is common but implementation is different.

Timer component is generic and is used by all the device drivers for send keep alive message as well as new attributes. We kept timer part out of device drivers as it reduces the weight of it and we can use any third party component, this reduces the time to develop our application.

Our design is not specific to underlying communication media, it can interoperate with any other type of communication media as Ethernet or ADSL etc.

7 Reference Implementation

In our demo system, we will be demonstrating how an ACTIVATE command will reflect a change in the PDA and the devices. Refer to fig on next page which shows the sequence of operations occurring for an ACTIVATE command.

We also had the question of whether the Communication component should be part of the INTELHOMENET EXE. But, we later realized that the access technology need not be only Bluetooth, but also other standards like 802.11. So, we had to separate out this component out from the INTELHOMENET exe.

DatabaseHandler

We explain below, how the context information is stored in a database. The names of directories and filenames are important as illustrated below.

The database is some sort of permanent storage where context settings and device parameters are stored. We assume the following directory structure. The root of the tree is context,

/context/

List of contexts are listed under the root like,
/context/room1
/context/livingroom

Under each context, different device types are listed, for example
/context/livingroom/heater
/context/livingroom/lighting
/context/livingroom/tv
/context/livingroom/vcr
/context/livingroom/lock etc.,

Under each device type, one directory for each instance is created.
/context/livingroom/heater/heater1

For each device instance, 2 files are present. They are the configuration file and system file. A configuration file contains parameter values for different timings. A system file contains the manufacturing details, warranty details of the instance. This information is created during the registration period and is a reference for the user. One optimization could be that once the warranty period expires, the user is automatically intimated.

Therefore the leaf directory would have

/context/livingroom/heater/heater1/configuration
/context/livingroom/heater/heater1/system

Demo System

In our demo system, we will be demonstrating how an ACTIVATE command will reflect a change in the PDA and the devices. Refer to Figure () which shows the sequence of operations occurring for an ACTIVATE command.

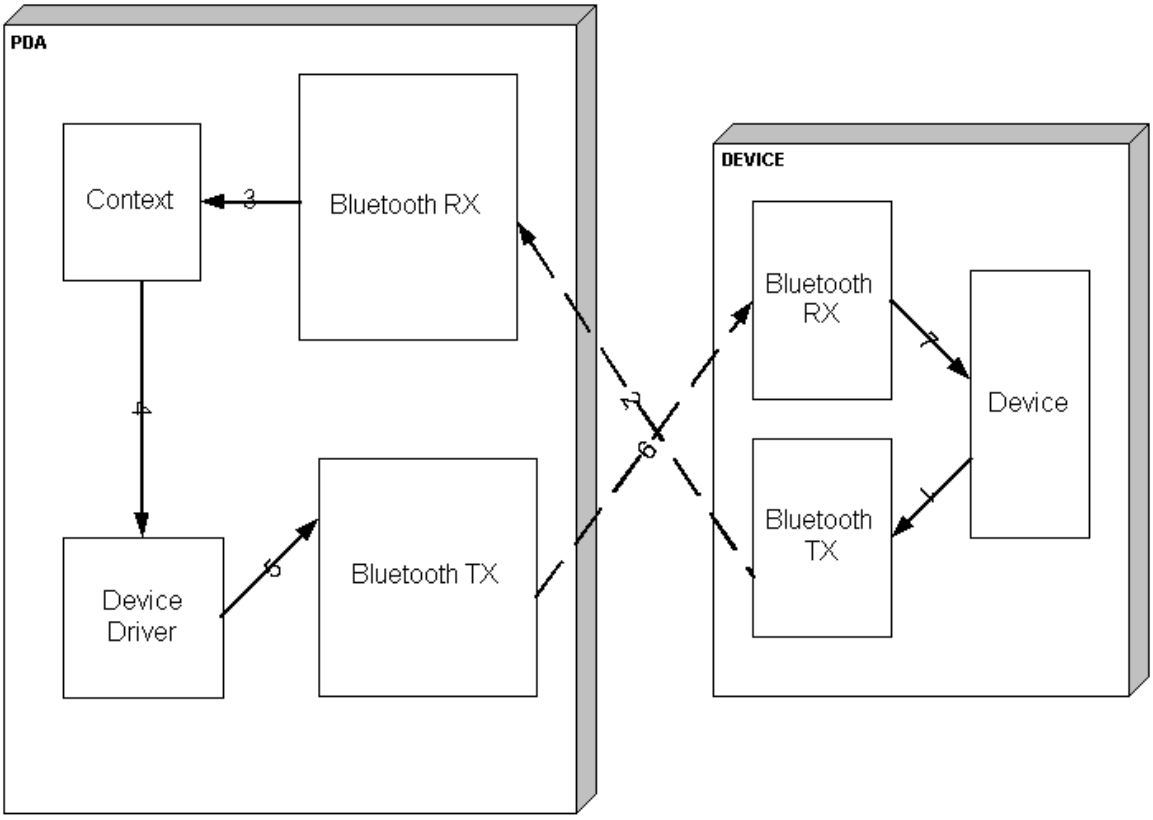
We also had the question of whether the Communication component should be part of the INTELHOMENET EXE. But, we later realized that the access technology need not be only Bluetooth, but also other standards like 802.11. So, we had to separate out this component out from the INTELHOMENET exe.

Proof of Demo and Architecture

Thinking in terms of implementation while developing the Proof of Demo, resulting in quite a few changes in our basic design. The most important changes are listed below.

1. Since different devices have different parameter names and values, any exchange of information between device drivers and other components like Communication, Context or Database becomes a problem. To avoid the situation, we make the attributes usable by the device drivers alone.

2. Exchange of attributes between the Device driver and the Database requires the use of Tag Length Value(TLV) approach, so that we specify the name, type of parameter along with the value. Different methods corresponding to different data types are supported to handle all data types like integer and BSTR.



3. Exchange of attributes between the Device driver and the Communication interface is again similar to TLV format. Also, this changed our architecture in the sense that we now need to support overloading methods and also that any request might require more than one transmission and reception, for each of the attribute values.

4. Another change affected was for the databaseHandler component. We initially had the database as a central repository merged with the context. But, we saw that for database access by the devices, they need to forward requests to the context and then back again. This was an overhead. Also, when multiple devices are

active, then the context and therefore the single instance of the databaseHandler will result in synchronization problems. To avoid this scenario, we made the databaseHandler an independent component, being able to individually instantiable by the context and the Device drivers.

The DeviceDriver's as part of their initialization will instantiate a databaseHandler and also a Timer component.

Here, we see that that every component is a DLL, giving us the advantage of easy substitutability and polymorphism. This becomes an easy plug and play solution to the problem of inter-component communication.

8 Conclusions and future work

We developed an architecture for an Intelligent Home Network. We named in INTELHOMENET. We introduced a concept like context, to combine a set of devices and then capable of controlling them with mobile devices like PDA's or mobile phones. We saw that we could build an architecture using the paradigms of Software component model.

We did not consider security constrains in our architecture. For an example, since remote automation scenarios involve the use of emails, the INTELHOMENET system is susceptible to email server unavailability and email delivery latency. Fir example, when "I LOVEYOU" computer virus/worm and its variants plagued the email system around the globe, the homeowner can loose control to home devices for many hours.

But, we think that only simple authentication mechanisms are required to overcome any security problems. It is logical that these contexts would have to be stored in more than one carrier(PDA). If there are more than one user in a house, then all the users should be able to have a control over them. We could achieve this by duplicating the contexts into multiple mobile devices, assuming that all users have access to mobile phones.

We also feel that PDA should be voice enabled means that it should be able to recognize homeowners voice and parse it and then execute appropriate command requested by the user.

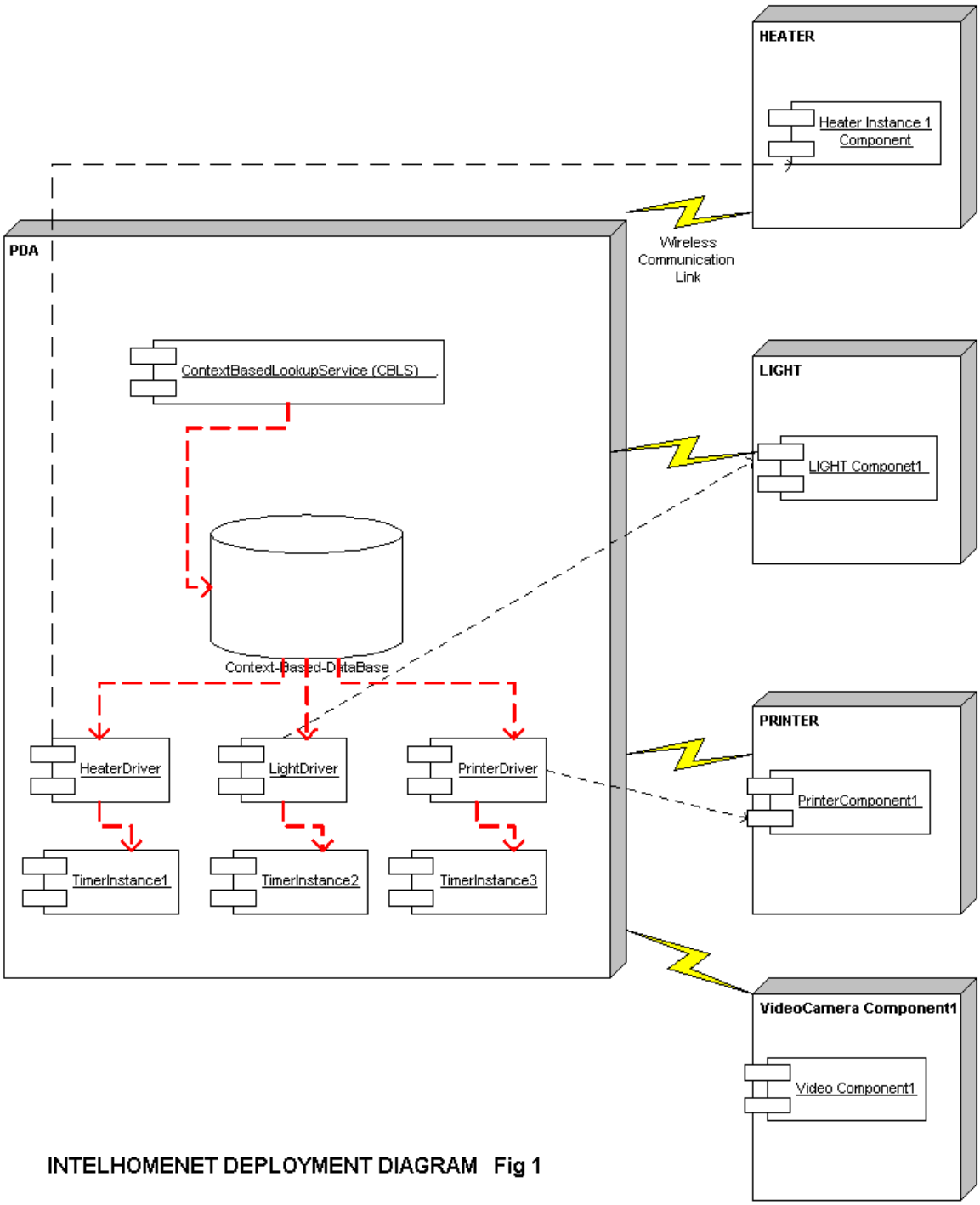
9 Reference

www.bluetooth.org

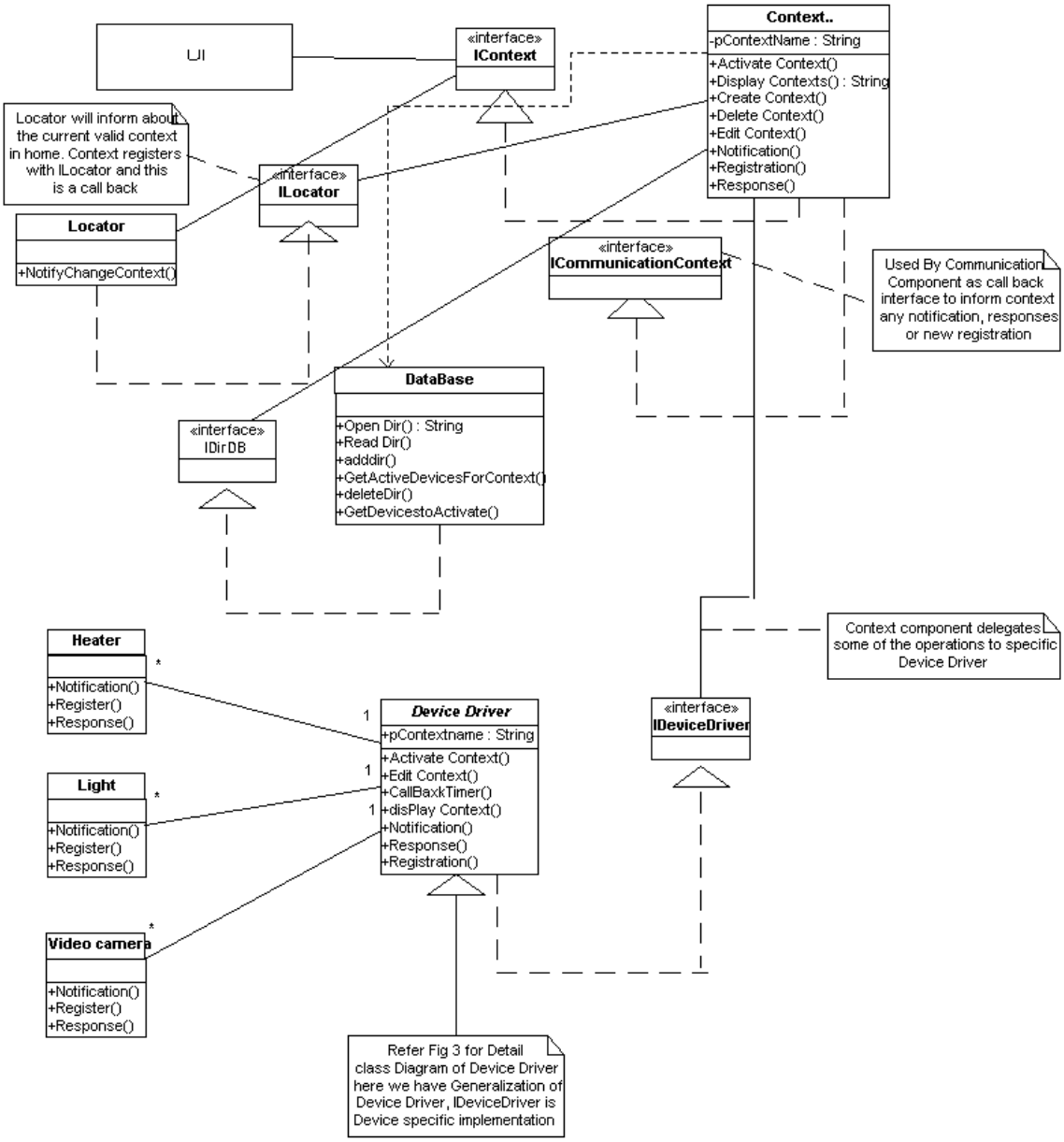
Component Software Beyond Object-Oriented Programming by Clemens Szyperski, Addison-Wesley 1999

Appendices

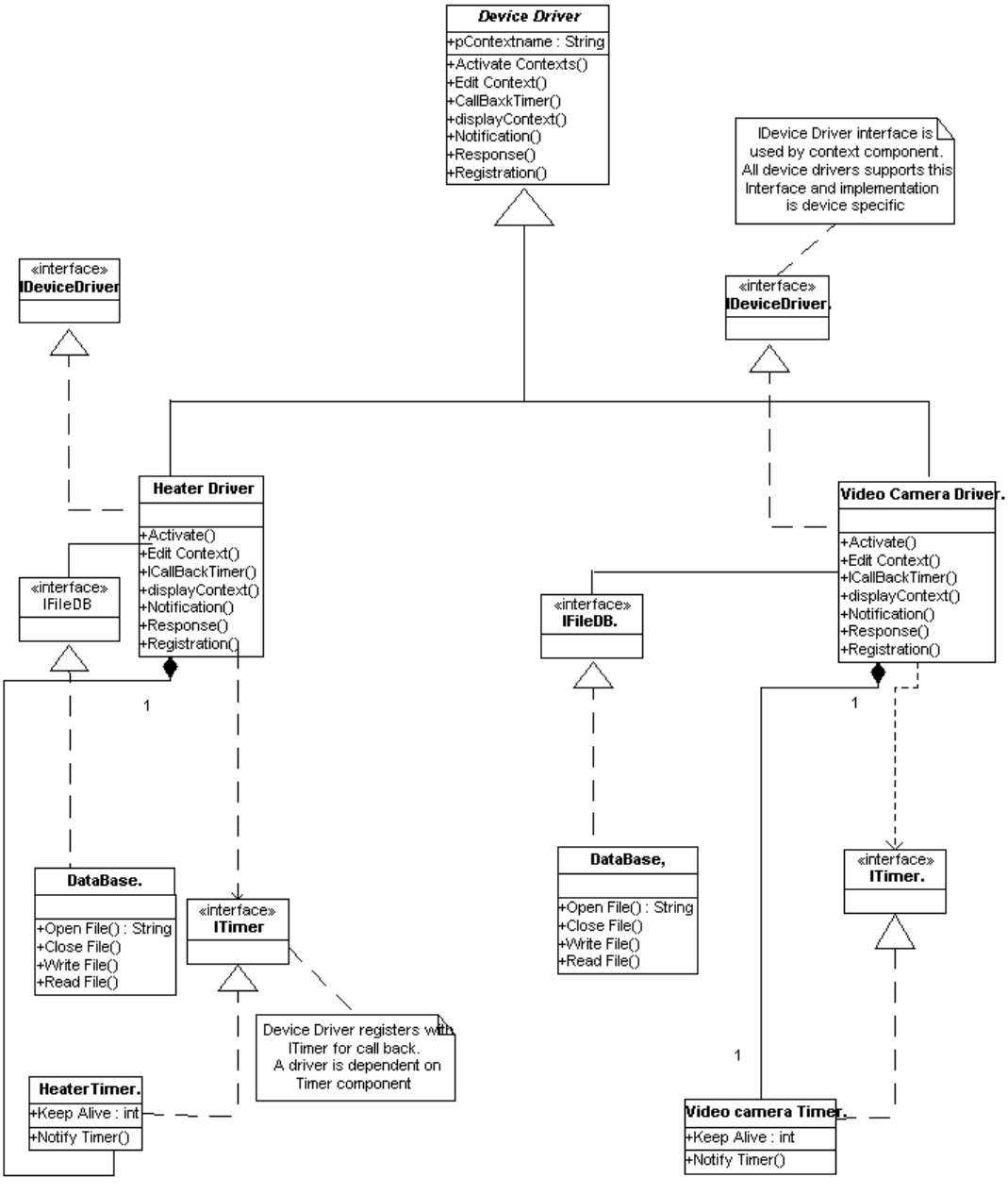
Diagrams



INTELHOMENET DEPLOYMENT DIAGRAM Fig 1

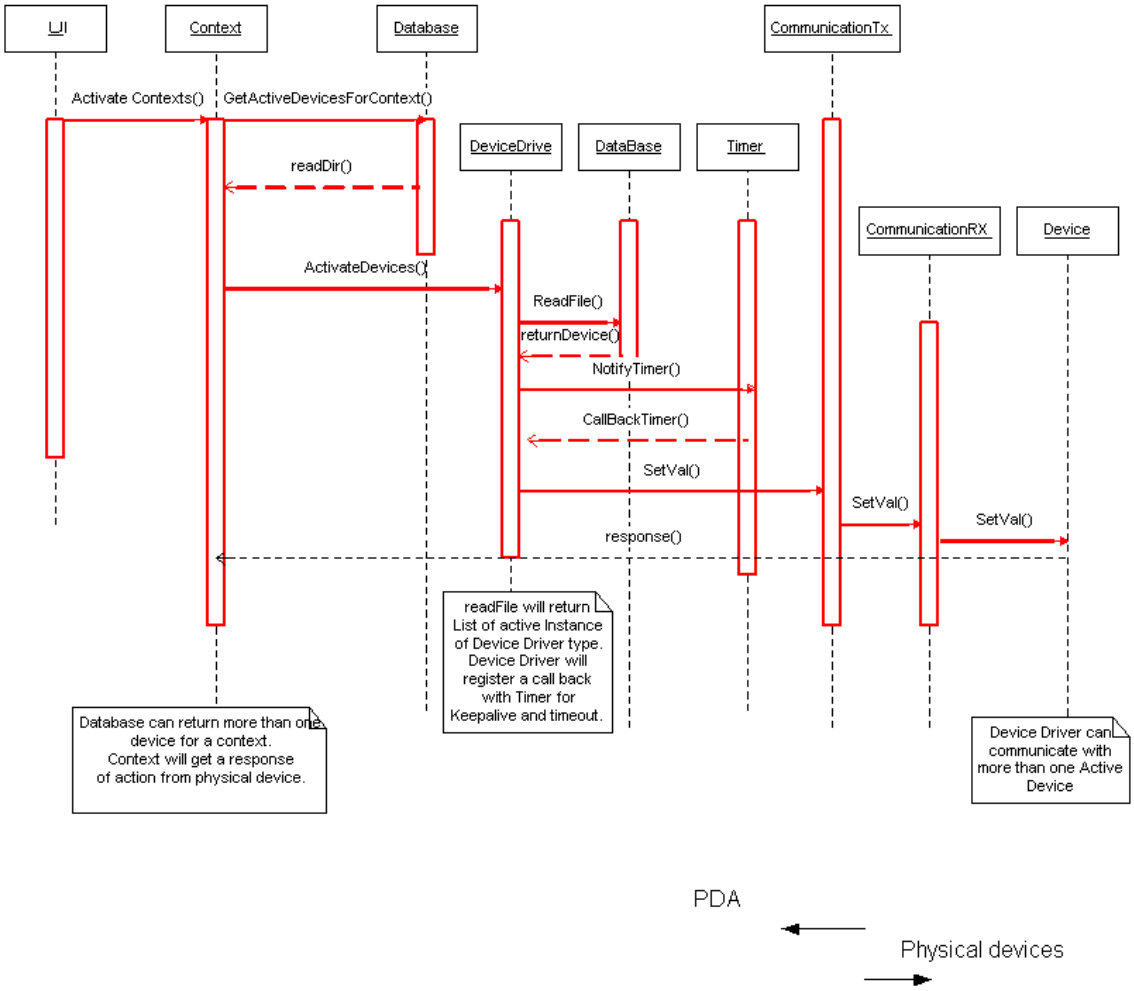


Class Diagram Fig 2



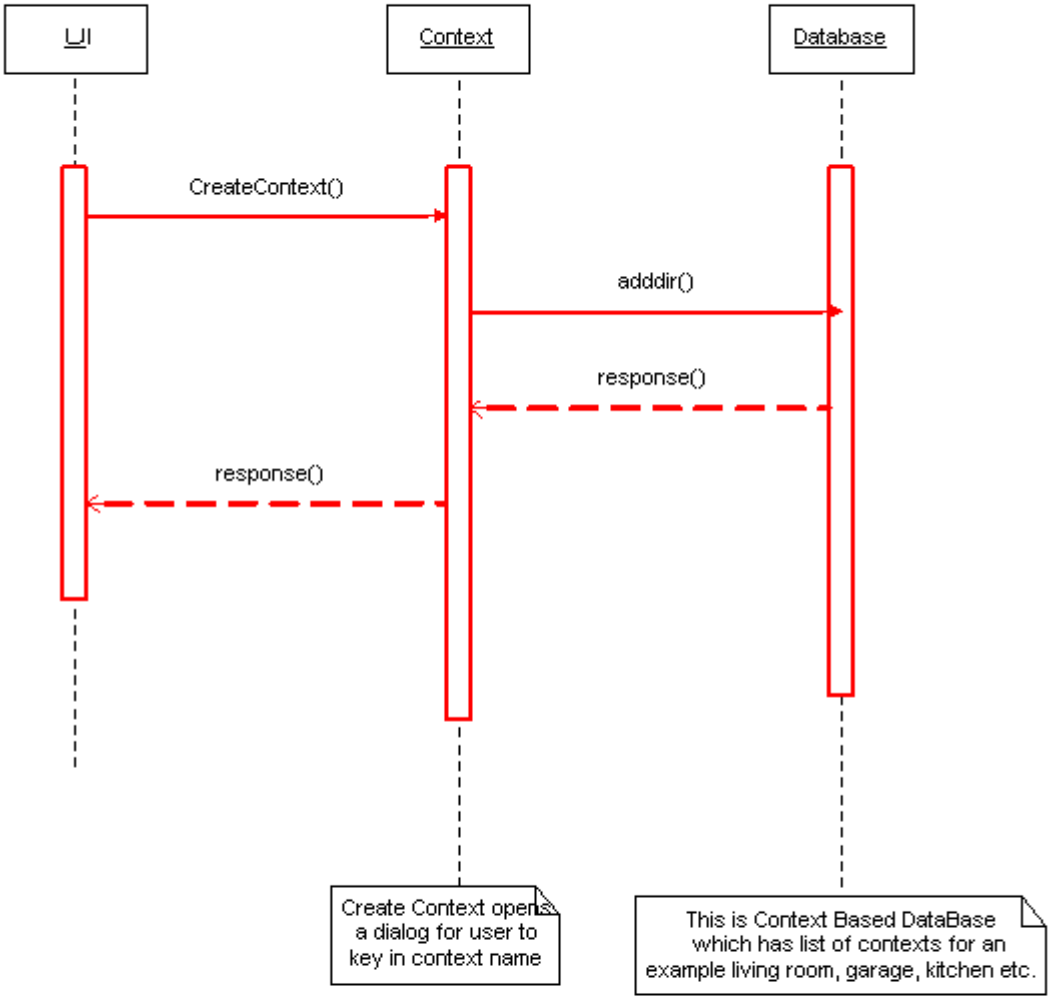
Class Diagram Fig 3

ACTIVATE CONTEXT



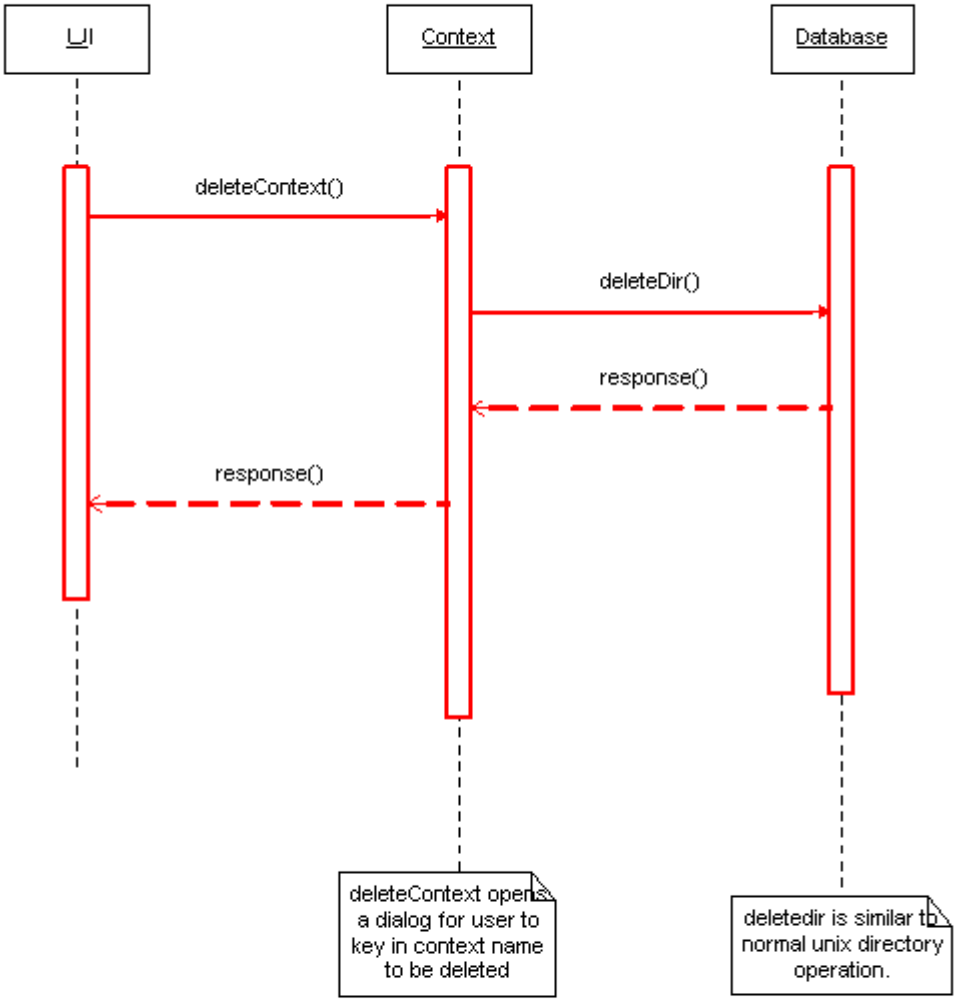
Sequence Diagram 4

ADD NEW CONTEXT



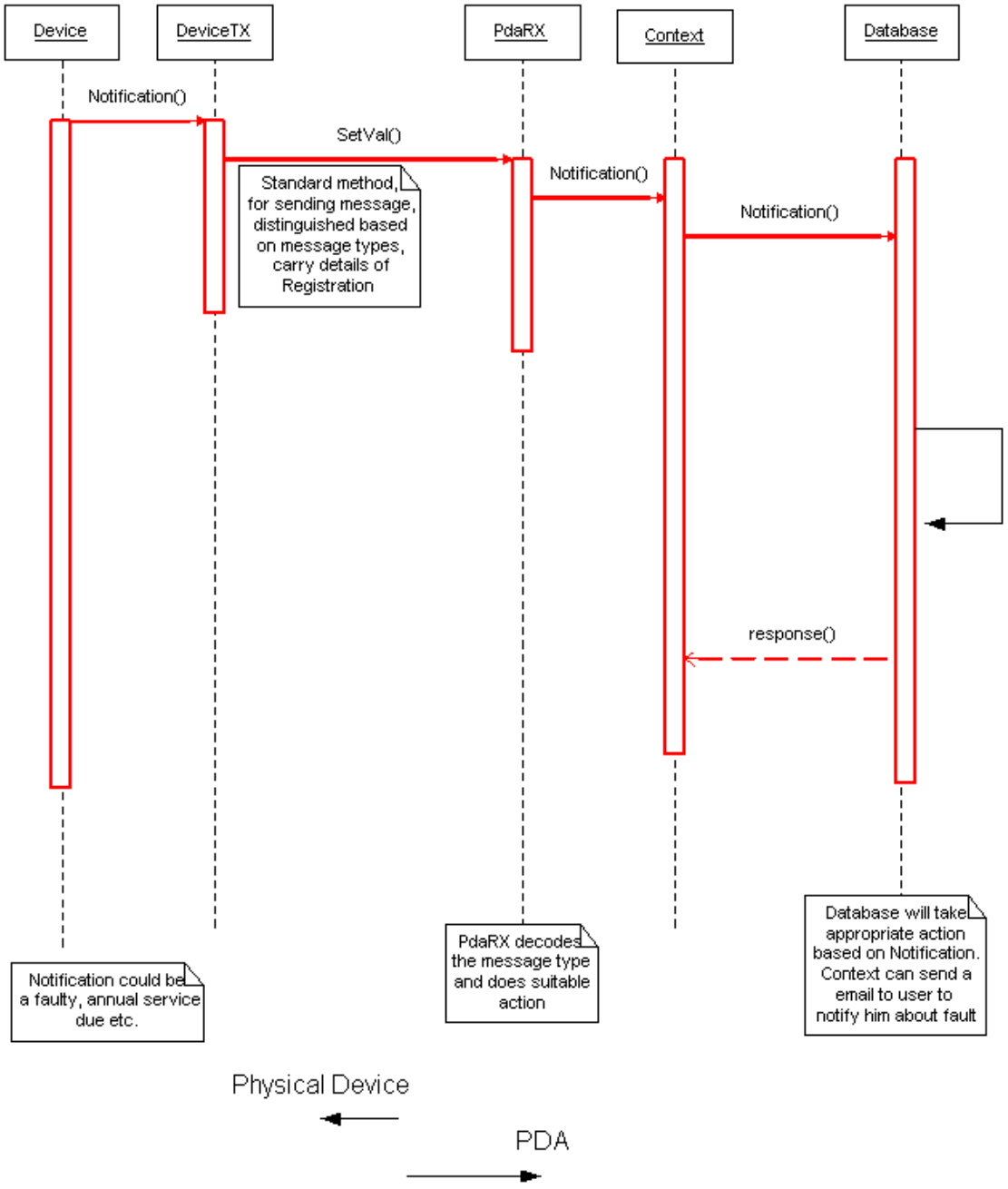
Sequence Diagram 5

DELETE CONTEXT



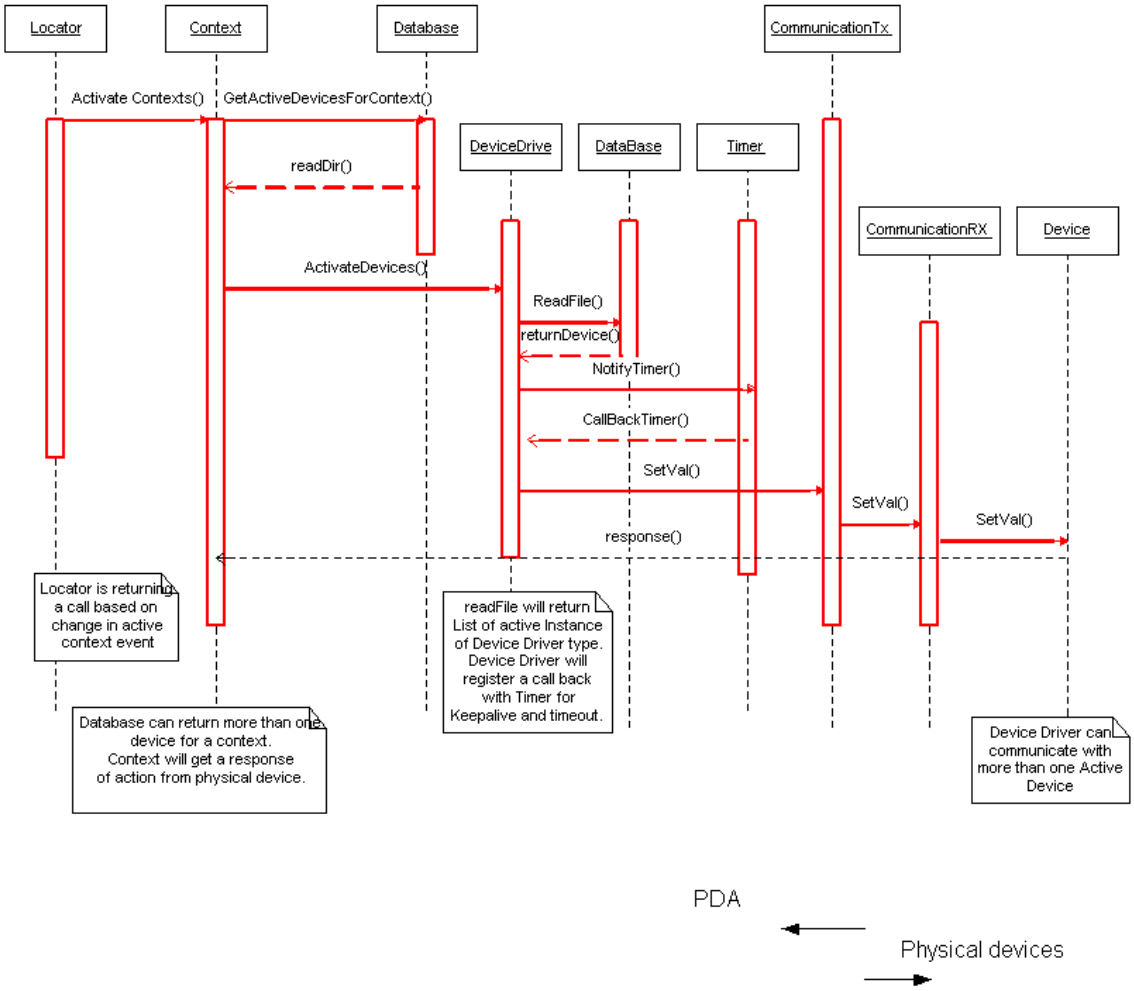
Sequence Diagram Fig 6

DEVICE NOTIFICATION



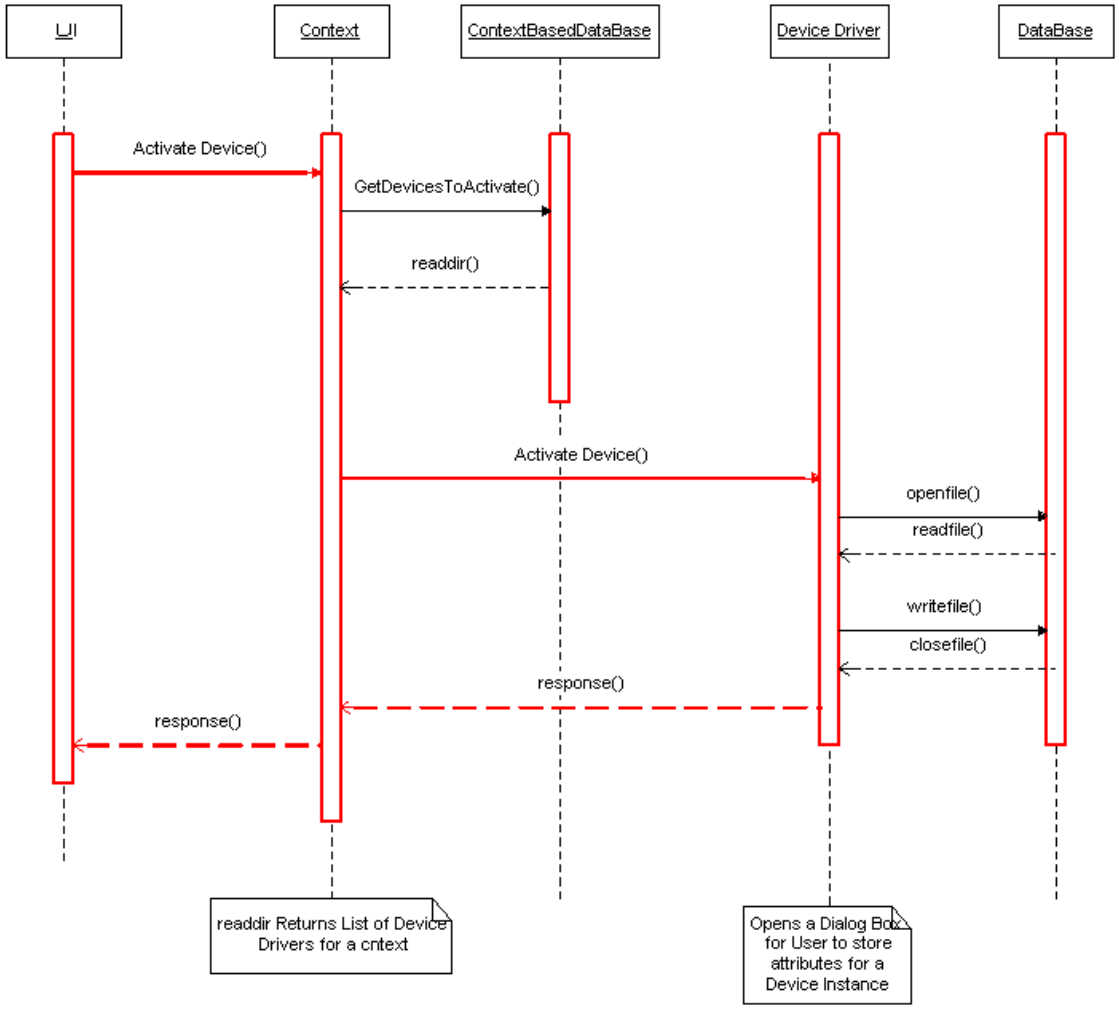
Sequence Diagram Fig 7

LOCATOR BASED ACTIVATE CONTEXT



Sequence Diagram Fig 8

Configure the Device to a context



Sequence Diagram Fig 9