OAM Software Architecture Design
CS 446 – Spring 2001

GROUP #1
Troy Gonsalves 97083748
tgonsalves@student.math.uwaterloo.ca

Chris Mennie 97024327
camennie@student.math.uwaterloo.ca

David Tapuska 97084449
dfapusk@student.math.uwaterloo.ca

May 28, 2001

1.0 Abstract

This document discusses the architectural style used to design the SX4 Operations, Administration and Maintenance (OAM) Software System. Before describing the proposed architecture, this document provides a comparison of several potential architectural styles including: Pipes and Filters, Layered, and Client-Server Styles. An *OAM* Software System based on each of these architectures is described in terms of its modifiability, ability to evolve and feasibility. Additional criteria include: the potential for the use of standardized interfaces; the ease of testing; and, the satisfaction of the system requirements (as specified in the document *CS445 Software Requirements Specification: OAM Software for SX4*).

The conclusion of this report is that a combination of the Client-Server and Layered architectures is best suited for the OAM Software System. Within this hybrid architecture, modules interact in keeping with the Client-Server style; however, decomposing the Client or Server modules reveals an internally layered architecture.

Feasibility studies have been performed to ensure implementation of the proposed architecture can be accomplished with readily available tools and technologies. The client side could be implemented using Tcl/Tk, with the aid of a *GUI* builder. Communication could be handled using *TCP/IP* sockets, and messages formed in *XML*. The database could use a readily available relational database, MySQL.

Evolution requirements have been present throughout the entire architecture stage. Various scenarios of how the system can adapt and react to such changes is presented. It is concluded that since the proposed design is highly modularized and distributed, the system is relatively malleable.

The design team believes that the hybrid architecture proposed will maximize the modifiability, evolvability and feasibility of the OAM Software System.

2.0 Introduction

2.1 Purpose

The purpose of this document is to present an architectural style to the client for approval. The design chosen is argued to meet the system requirements, while allowing modifiability and customizability. The document presents the architecture view from a Top Level, while looking into a few interesting *components* that have sub-architectures of their own. The intended audience is someone who is familiar with architectural styles and has read the requirements and specifications for the system.

2.2 Scope

The main sections to the report are: Architecture, External Interfaces, Ability to Evolve, and Feasibility.

The Architecture section discusses three different approaches to the design of the system. Each approach is presented and then summarized in a table. The section continues on to take a hybrid approach, combining Client-Server and Layered architectures for a proposed system design.

The External Interfaces section presents what external interfaces will be seen to the external applications, such as the *UI* and *CU* interfaces.

The Feasibility section provides an overview of an intended implementation. Technologies used for the various components of the software system are discussed in this section, with reasons for choosing them and advantages they provide.

The Ability to Evolve section discusses how the proposed system can evolve and adapt to future requirements placed on the system. A discussion of the anticipated future requirements of the system is presented. The benefits of the proposed architecture, such as modularization and distribution allow changes to the system to occur in centralized areas and to have relatively small effect on the number of modules.

2.3 Document Conventions

The document has the convention that the first occurrence of technical jargon is italicized, and is inserted into the Data Dictionary. [Section 8]

Architecture diagrams in this document follow the following conventions:

- Events contain all data relevant to the event.
- Data is sent only in response to an event.
- System Components are those components of the system for which the development team is responsible.
- 3rd Party Components are those components that external parties are responsible for.

3.0 Architecture

3.1 Potential Architectural Styles

3.1.1 Pipes and Filters

In a pipe and filter style, each component has a set of inputs and a set of outputs. A component reads streams of data on its inputs and produces streams of data on its outputs, delivering a complete instance of the result in a standard order [Shaw and Garlan].

3.1.1.1 Modularity

The division of the OAM Software System into filters is both logical and simple for data sent to the database or CU. However, the user interface would be difficult to implement as a filter since it is highly interactive.

3.1.1.2 Interfaces

The issue of paramount concern is the difficulty the development team will face when trying to implement an interactive user interface as a filter. The difficultly with implementing interactive components is a known problem with the pipes and filters style.

3.1.1.3 System Evolution

As long as the filters are sufficiently independent, the system would be highly evolvable.

3.1.1.4 Ease of Testing

The independent nature of the filters in this style allow for the individual testing of each filter. Hence, thorough regression and stress tests can be performed on each component.

3.1.1.5 Satisfaction of *SRS*

The SRS states that only one operator needs to be supported; hence, it is possible to create a system using the pipes and filters style.

3.1.1.6 Overall Feasibility

It is the belief of the design team that the versatility of the user interface and the scalability of the backend components are the most important aspect of the OAM Software System. Although the use of a pipes and filters architectural style would yield a highly scalable backend, the incorporation of the user interface is poor.

3.1.2 Layered Architecture

A layered system is organized hierarchically, each *layer* providing service to the layer above it and serving as a client to the layer below. The connectors are defined by protocols that determine how the layers will interact. Topological constraints include limiting interactions to adjacent layers. [Shaw and Garlan]

3.1.2.1 Modularity

It seems natural to divide the OAM Software System into various layers. These layers would include; a user interfaces layer; an event handler; a core processor; and finally, an external interfaces layer for communicating with the database, printer and CU.

3.1.2.2 Interfaces

Standard interfaces would be required between the layers and interactions between non-adjacent layers (*bridging*) would be strongly discouraged.

3.1.2.3 System Evolution

Provided there is little bridging of layers, the system's ability to evolve of the system is excellent. Each layer would only be dependent on its upper and lower layer interfaces to communicate. This reduces the magnitude of the changes required to modify or replace a layer.

3.1.2.4 Ease of Testing

Testing the lower levels of the architecture would not be difficult since each function should be relatively simple. Testing of the higher levels may be difficult since their dependence on the lower layers creates a tree of dependencies; if the tree is sufficiently large, locating bugs becomes difficult.

3.1.2.5 Satisfaction of SRS

The SRS states that only one operator needs to be supported; hence, it is possible to create a system using a layered architecture.

3.1.2.6 Overall feasibility

If the system is constrained to allow at most one Operator then a layered architecture is a very reasonable style.

3.1.3 Client-Server Architecture

In a client server style, components are abstracted into ones that provide black-box services and ones that request those services. Components that provide services are called servers; components that request services are called clients. The two types of components are linked via a connector, a network that allows access to remote servers. [CS 446 Course Notes]

3.1.3.1 Modularity

The division of the system into client and server modules is natural. Access to the database and printers can be placed in one server module. While the portions of the system involve in interfacing with the Operator could be placed in a client module. The only oddity is that the CU both requests and provides services. To resolve this a special component called a Handler is introduced; this component will facilitate requests from the CU and provide services to other components.

3.1.3.2 Interfaces

The use of a standardized protocol would allow components to communicate. Making the components event-driven and using *message-passing* seems like a natural way to of providing generic interfaces between components.

3.1.3.3 System Evolution

The separation of the system into clients and servers, allows each component to be modified with minimal effect on other components.

Even though the SRS calls for at most one operator, a likely enhancement would be to allow multiple operators to use the system. This style naturally lends itself to extension into a multi-user environment.

Another important capability would be the ability to operate as a distributed system. The Client-Server architecture inherently incorporates the fact that components may be on different machines (assuming the links between components can communicate across a network).

3.1.3.4 Ease of Testing

The components of this style can be tested individually because of there independent nature. Hence, thorough regression and stress tests can be performed on each component.

3.1.3.5 Satisfaction of SRS

The SRS states that only one operator needs to be supported. It is possible to create such a system using the client-server style; the system would simply have one Operator Client.

3.1.3.6 Overall Feasibility

The client-server style satisfies the requirements of the SRS while at the same time allowing the components of the system to be both highly adaptive and easily distributed.

3.2 Summary of Potential Architectures

The table below summarized the descriptions of each of the potential architectures. Each style is ranked from first to third in terms each of the attributes described in the previous section; the most import attribute being the "Overall Feasibility" of the style.

Style	**Modularity**	**Interfaces**	**System Evolution**	**Ease of Testing**	**Satisfaction of SRS**	**Overall Feasibility**
Pipes and Filters	3	3	2	1	1	3
Layered	1	2	2	3	1	2
Client-Server	2	1	1	1	1	1

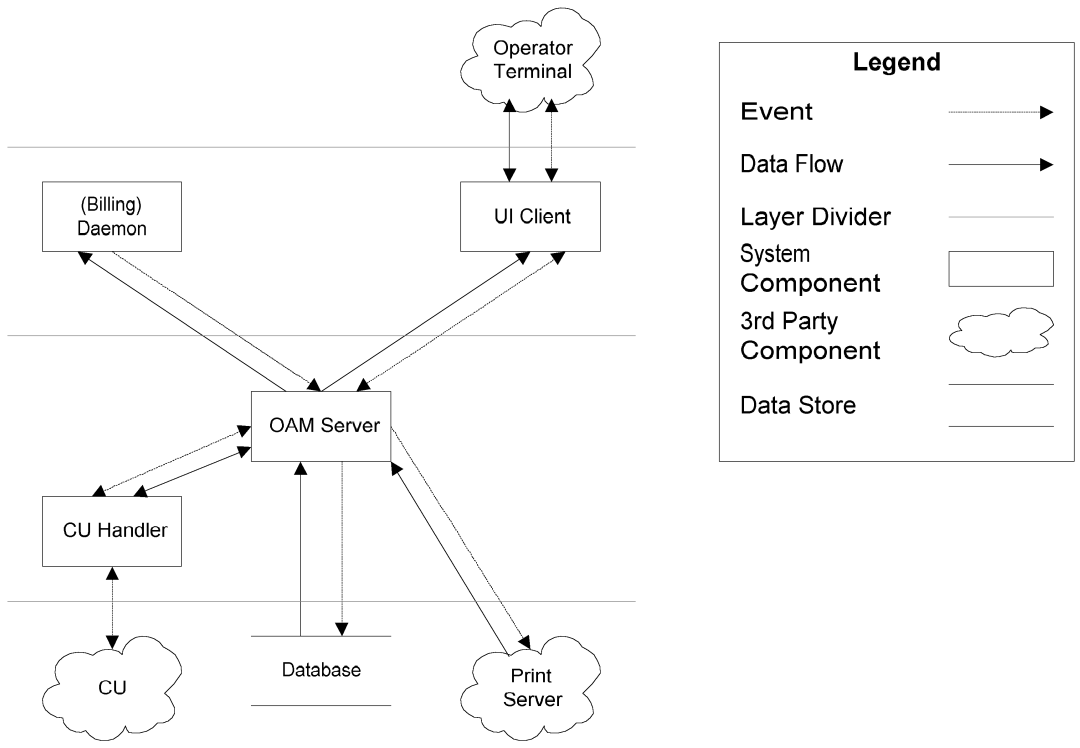
3.3 Analysis

The client-server style placed first in all categories except modularity. The problem with modularizing the OAM Software System into clients and servers was the CU, which could be considered both a client and a server. As mentioned above, a handler will be used to resolve this issue.

The pipes and filters style did poorly overall because of the problems with creating an interactive user interface using a filter.

The layered architecture did well, because it seems naturally to consider the OAM Software System as a series of layers; hence, a layered architecture will be used for the internal structure of some components.

3.4 System-level Architecture



Note: As shown in the system-level view of the architecture the OAM Software System can be considered simultaneously heterogeneous because it is both Client-Server and Layered.

3.4.1 Operator Terminal

The Operator Terminal is an abstraction of the physical machine into which the Operator would enter information (e.g. a personal computer or a workstation). This module is responsible for displaying data to the Operator and receiving input from the Operator.

The Operator Terminal can send events to and receive responses (data) from the UI Client (e.g. send an Add Customer event with the corresponding data, and eventually receive a response containing the new customer's ID). The Operator Terminal can also receive events from the UI Client (e.g. login prompt, error message) and send data to the UI Client in response to these events.

3.4.2 UI Client

The UI Client is responsible facilitating communication between the Operator Terminal the OAM Server.

3.4.3 (Billing) Daemon

The (Billing) *Daemon* is another type of client and thus it is similar to the UI Client. The significant difference is that this module is a daemon, meaning that is an automated process that executes a set of commands. Currently this modules only responsibility is to periodically send an event to the OAM Server telling it to print the current bills for all customers. Note however, that the responsibilities of this module may be increased to assist in the automated testing of the system or in any other automated or periodic functions the system may require as it evolves (this is why the word "Billing" is in brackets in this modules name).

3.4.4 OAM Server

The OAM Server's responsibility is to carryout all client requests. This may require accessing the Database, Print Server or sending an event to the CU Handler, the UI Client or the (Billing) Daemon. The OAM Server may also need to return a response (in the form of data or an event) to the requesting client.

It is the OAM Server's responsibility to maintain the consistence of the exchange information stored in the Database and cached on each CU.

When an error event is received, the server is responsible for sending an error event to all active UI Clients (and hence to all Operators currently logged into the system) The error event may also be sent to the (Billing) Daemon if necessary. If there are no active UI Clients at the time of the error and the error must be viewed by an Operator to be resolved then the OAM Server must record the error and send it to the next UI Client that becomes active (i.e. to the next Operator that logs in)

3.4.5 Database

The Database is a Third Party Component responsible for storing all persistent information required by the OAM Software System. The database must support atomic operations, also known as transactions.

3.4.6 Print Server

The Print Server is a Third Party Component. The Print Server receives requests to output customer bills; typically, this will involve printing the bills. Note that this module in addition to printing the information it receives could also output information via email (for example). In fact, despite its name, this module does not necessarily *print* the information it receives at all; this would of course imply that the information was output by some other means (e.g. email).

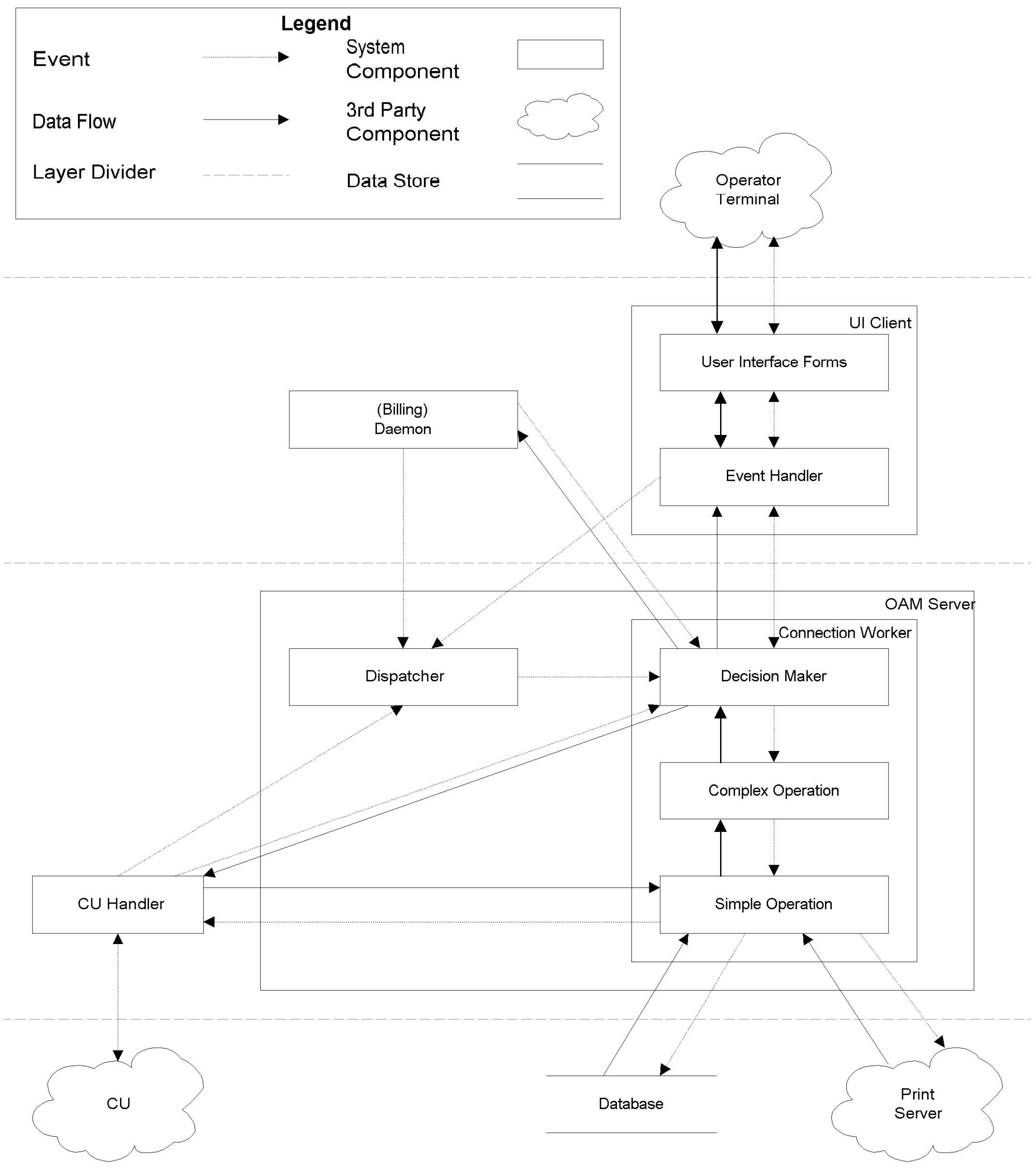
3.4.7 CU Handler

The CU Handler is the proxy between the OAM Server, and the CU that allows the CU to act as both a client and a server. All messages to and from the CU must go through that a handler.

3.4.8 CU

The CU is a Third Party Component that provides an interface to the hardware components of the SX4 System. For a more detailed description of the CU see [CS 445 Project Introduction, CS 445 Software Interface Description]

3.5 Component-level Architecture



Note: The Communication Subsystem, Operator Terminal, UI Client, (Billing) Daemon, OAM Server, Database, Print Server, CU Handler, and CU are described in the previous section on the [Section 3.4] System-level Architecture.

3.5.1 UI Client

The UI Client can be decomposed two (layered) components, User Interface Forms and an Event Handler.

3.5.1.1 User Interface Forms

The User Interface Forms module acts as an interface between the UI Client's Event Handler and the Operator Terminal. This involves formatting the display that is output by the Operator Terminal and altering this display when events and data are received from the Event Handler. The module must also passes events and data from the Operator Terminal to the Event Handler.

3.5.1.2 Event Handler

The Event Handler decides what type of message is sent to the OAM Server in response to an event received from a User Interface Form. For example, pressing a button to add a customer results in an "Add Customer" event being sent to the OAM Server. The Event Handler then waits for the OAM Server to respond to this event with either an ID number for the new customer or and error message.

The Event Handler may also receive unsolicited events from the OAM Server such as error messages. Upon receipt of such messages, the Event Handler should display the event on the Operator Terminal by using User Interface Forms.

3.5.2 OAM Server

The OAM Server contains two main modules, the Dispatcher and the Connection Worker.

The Worker is active from the point the Operator begins their session and until he/she ends their session.

3.5.2.1 Dispatcher

The Dispatcher is a lightweight module whose primary function is to act as an administrator or a nexus for client processes.

The Dispatcher is responsible for receiving initial requests from the clients and for allocating a Connection Worker to service that particular client. The key idea behind the Dispatcher is to delegate work, so that it does not become a bottleneck.

The Dispatcher is also responsible for handling any error messages received from the CU Handler. These error messages are then sent to all active Connection Workers. If there are no active Connection Workers then the Dispatcher records the error and sends it to the Connection Worker created when a login request is received from a UI Client.

3.5.2.2 Connection Worker

The internal architecture of the Connection Worker contains three layers, Decision Maker, Complex Operations, and Simple Operations.

3.5.2.2.1 Decision Maker

The Decision Maker handles all communication between the OAM Server and a client. Once created a Connection Worker's Decision Maker is the only component that can communicate with the client.

The Decision Maker is also responsible for deciding which Complex Operations are performed in response to an event from the Connection Worker's client or the Dispatcher.

3.5.2.2.2 Complex Operations

A Complex Operation is a sequence of one or more Simple Operations. These Simple Operations may be wrapped in a transaction. For example, consider Adding a Subscription, this could be implemented as a Complex Operations involving three Simple Operations, namely Allocate Phone Number, Allocate Line Card and Update Customer.

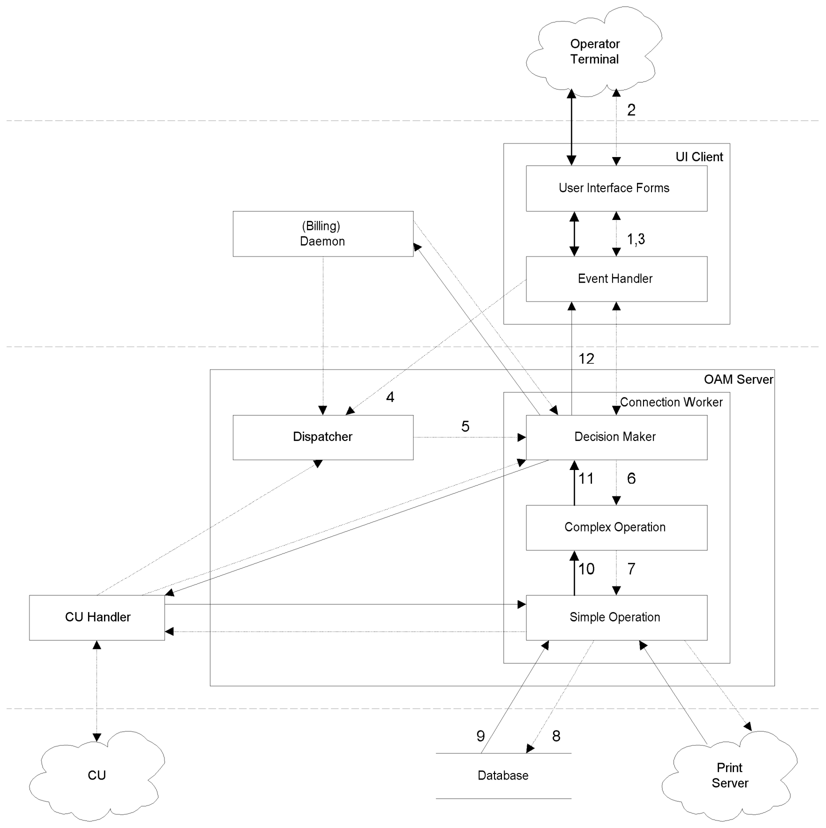
3.5.2.2.3 Simple Operations

The Simple Operations layer implements basic operations and provides an abstract interface to the Database, Print Server, and CU Handler.

3.6 Examples of Component Interaction

This section will trace through a few realistic examples to demonstrate how the components of the OAM Software System interact.

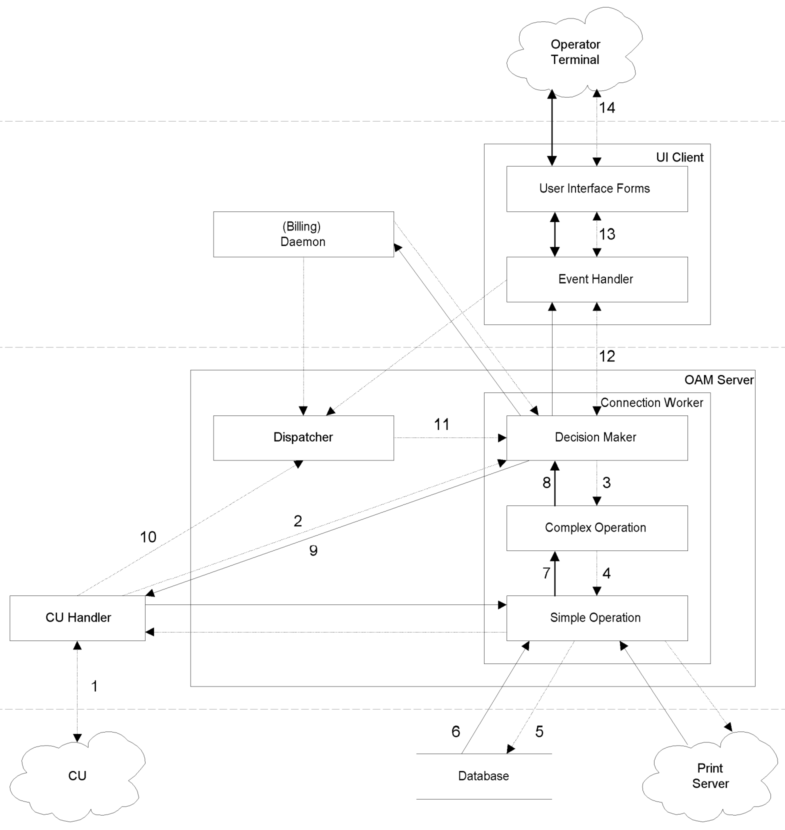
3.6.1 Adding a Customer



Operator Login
Assume that the OAM Server and Database have already been started.

1) The UI Client starts; its Event Handler sends an event to the User Interface Form to display the login dialog.
2) User Interface Forms formats the display and sends it to the Operator Terminal, which displays the login prompt; the Operator enters his/her user name and password into the given form and presses the "Login" button.
3) A login request is passed to the Event Handler from the User Interface Form.
4) The Event Handler contacts the Dispatcher with a login request.
5) Dispatcher creates a Connection Worker for the client and forwards the login request.
6) Connection Worker sends a validate request to Complex Operations.
7) A Complex Operation results in three Simple Operations: connect, validate and disconnect.
8) Simple Operations connect to, validate the user's name and password and disconnect from the Database
9) The Database responds with success.
10) A success of login message is returned to the calling Complex Operation
11) A success of login message is returned to the calling Decision Maker.
12) A success of login message is returned to the UI Client's Event Handler.

3.6.2 Displaying an Error Message Sent from the CU



"UpdateDB" message sent from CU with an error code set
Assume the CU already has acquired a Connection Worker.

1) CU sends an updateDB event to its CU Handler
2) CU Handler sends updateDB message its Connection Worker
3) Decision Maker then sends the message to the Complex Operations layer
4) The Complex Operation results in three Simple Operations: connect, update and disconnect.
5) Simple Operations connect to, update and disconnect from the Database
6) Database responds with success
7) Simple Operations then responds to Complex Operations with success
8) Complex Operations then responds to the Decision Maker with success
9) The Connection Worker then responds to the CU Handler with a success message for updateDB event
10) The CU Handler detects that an error bit has been set in the updateDB message from the CU and sends an error message is sent to the Dispatcher
11) The error message is propagated to each Connection Worker
12) For each Connection Worker, the Decision Maker then sends an error message to each Client Event Handler
13) The error message is send to the User Interface Forms to format the display of the error
14) The error message is displayed on the Operator Terminal

4.0 External Interfaces

4.1 The Communication Subsystem

Components use message-passing over the communication subsystem to interact with one another. This subsystem can be separated into two pieces, the channel and the massage format. The implementation details of both of these pieces should be hidden from components using the communication subsystem.

The communication subsystem is responsible for performing validity checks on all messages. This check can be performed either before of after the message is transferred as long as the check is performed before making the message available to its recipient.

4.1.1 The Channel

The abstraction of the channel increases the ease with which the system can be ported to different platforms. That is, to port the communication channel to a specific platform the development team need only provide an implementation of the channel's interface that will work on that platform.

One desirable property of the channel would be that it allows components to communicate over a network so that the OAM Software System can function as a distributed system.

4.1.2 Message Format

The message format must be generic enough to encode both data and events, yet it should be concise enough to allow for the efficient validation of a given message.

The message format should also be designed such that components can communicate over a channel despite differences in the programming language in which each component is written or the format in which local data is stored (e.g. little endian vs. big endian representation of a binary number)

4.2 Database

The OAM Server communicates with the Database through the Simple Operations layer of a Connection Worker. When the Decision Maker layer of a Connection Worker receives an event it decides which Complex Operation(s) to perform and then it executes them. If a Complex Operation requires a connection to the database, it makes three or more calls to the Simple Operation layer, namely, one call to establish a connection, one or more calls to execute various queries, and finally one call to close the connection.

4.3 CU

The CU interacts with the OAM through the CU Handler module. This module maintains a persistent connection with the CU using message queues, as specified in CS 445 Software Interface Description. All communication to and from the CU is handled through the CU Handler.

4.4 Print Server

The OAM Server communicates with the Print Server through the Simple Operations layer. When something needs to be output, a Complex Operation will be executed. This Complex Operation will begin by calling a Simple Operations to format the information to be output, then on to connect to the Print Server and send the information to be output. The information sent will be a printable data type such as text or a file (for example a PDF). The Print Server will send a response to the Simple Operation layer in response to the output request. Once this response is received, the Simple Operation can assume that the output was successfully completed. Control will then return to the Complex Operation which will make one more call to the Simple Operations layer to disconnect from the Printer Server.

5.0 Feasibility Analysis

Some of the components outlined in the previous sections are dependent on the certain requirements being implemented. This section will describe a number of technologies that can be used to implement these requirements.

5.1 MySQL Database

The Database is a vital component to the OAM Software System. To allow easy storage and retrieval of the OAM information, the use of MySQL as database was explored.

MySQL supports the *SQL* standard, which is one of the most popular and common database standards. Should the need arise to change to a different database, the use of SQL for the Database in the OAM Software System will make the transition simple.

MySQL is available in both binary and source forms, for a number of common operating systems. This gives the OAM Software System a large degree of flexibility in terms of the platforms on which it can execute. There is also, abundant documentation on http://www.mysql.com which has aided the development team in installing and configuring MySQL.

There are a number of APIs available in multiple languages that can be used to communicate with a MySQL server. For example, there is an API for C++ and one for Java.

Finally, like most database systems, MySQL is thread safe, supports transactions and has a number of security features.

5.2 Communication

5.2.1 TCP/IP Sockets as Channels

If TCP/IP sockets are used to implement channels then the development team will have relatively easy time creating concrete implementations of the channel's interface. This is because most modern operating systems include TCP/IP and network support.

In addition, because TCP/IP sockets will allow the components to communicate over a network the OAM Software System can operate as a distributed system

5.2.2 Using XML as the Message Format

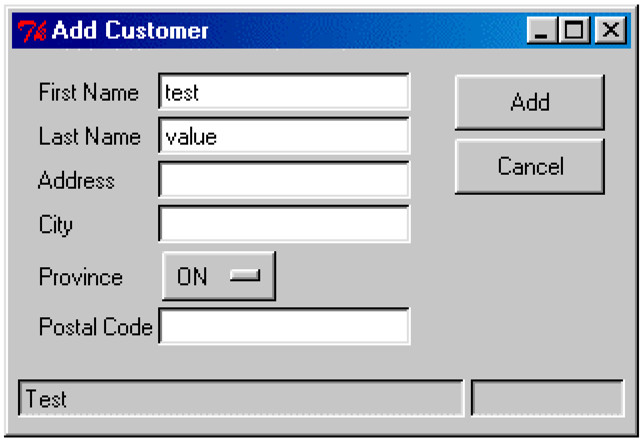
XML is fully capable of meeting all the requirements specified for the message format. It can be used to encode both events and data and there are a number of high-speed validators and parsers available in a wide varity of languages (including C++, Java and even Tcl/Tk).

An additional advantage to using XML that the language is text-based, this will simplify debugging and may assist the testing team in automating their tests.

5.3 Using Tcl/Tk for the GUI

The *UI* Client could be implemented using Tcl/Tk. This language was chosen due to its wide availability on a variety of platforms. The language is also relatively simple to learn, yet highly versatile. This should allow the development team to create a robust user interface.

Below is a prototype of the Add Customer form.



First Name	test
Last Name	value
Address	
City	
Province	ON
Postal Code	

Test

The SpecTcl *GUI* builder will be used to assist in the initial layout of Tcl/Tk forms in the hope that this will speed up the development of the user interface.

6.0 Ability to Evolve

6.1 Performance Enhancements

Performance is arguably the most important aspect of a system. The following is a list of methods for improving the speed of the system:

- Compress messages; which will decrease data transfer time, but increase data processing time (i.e. the messages will have to be decompressed at some point).
- Simplify the message format; this may decrease data transfer time (assuming the messages get smaller). However, this may compromise the system's ability to evolve.
- Collapse layers (e.g. in the Connection Worker); by combining two or more layers into one, developers may be able to increase the efficiency of their code thus improving the speed of the system.
- Distribute System-level Components to different machines; this may increase speed if the system is overburdened with requests (see Section 6.5 Distribution of Components for details).

6.2 Scalability

The proposed architecture defines a client-server relationship between the UI Client and the OAM Server. This allows the system to easily accommodate multiple Operators; despite the fact that the SRS states that there is at most one Operator. It is possible to maintain the appearance of high performance in a multi-user environment by using any of the alterations mentioned in the [Section 6.1] Performance Enhancements section.

6.2.1 Upgrading the Database, Print Server or CU

There is a direct relationship between a system's scalability and its low-level mechanism(s). Hence, it is important that a system is able to upgrade and/or replace these mechanisms with relative ease. In the OAM Software System, all communication with the Database, Print Server and CU is performed through a layer of abstraction (see Section 3.4.7 CU Handler and Section 3.5.2.2.3 Simple Operation). Therefore, if it becomes necessary to alter the Database, Print Server or CU these changes will have a minimal effect on the rest of system.

6.3 Adding and Enhancing Features

Adding or enhancing a feature would naturally require the alteration of at least one component (possibly more, depending on the type of feature). However, because of the structure for the event communication subsystem these alterations will have no effect that any component's external interface.

6.4 Internationalization

Multiple language support would be isolated to the User Interface Forms layer and the Simple Operation layer and would not affect the other components. Changing bill calculations (i.e. different tax structure, different currency) would also be isolated to the Simple Operations layer.

Note: these changes would also require the storage of some additional information about Customers and Operators (e.g. Preferred Language, Preferred Currency, Tax Structure)

6.5 Distribution of Components

The abstract nature of the communication used by the System Components would allow them to function in a distributed environment. That is, each System Component could be placed on a different physical machine. In addition, with minor enhancements, the OAM Server could delegate requests to other OAM Servers running on different machines.

6.6 Multiple UI Formats

The generic nature of event communication allows multiple versions of the UI Client to operate simultaneously. Thus, the system could support windowed, web-enabled, even command-line versions of the UI Client.

7.0 References

CS 445 Project Introduction, August 2000.
    http://www.student.math.uwaterloo.ca/~cs445/project/intro.pdf

CS 445 Software Interface Description, August 2000.
    http://www.student.math.uwaterloo.ca/~cs445/project/soft.pdf

CS 445 SRS Specification December 1, 2000.
    [Hard copy enclosed with submission, soft copy email dftapusk@uwaterloo.ca]

MySQL Manual
    http://www.mysql.com/Downloads/Manual/manual.pdf

Shaw and Garlan, "An Introduction to Software Architecture"
http://www.swen.uwaterloo.ca/~dasiewic/courses/ece452/local/slides/intro_softarch.ps

Shaw and Garlan, Software Architectures: perspectives on an emerging discipline, Prentice-Hall, 1996

R. Pressman, Software Engineering, McGraw-Hill

TCL/TK SpecTcl GUI Interface builder
    http://dev.scriptics.com/software/spectcl/

8.0 Data Dictionary

Word	**Meaning**
Bridging	In a layered architecture bridging occurs when interact is allowed between non-adjacent layers.
Component or Module	An isolated part of the software system that performs an isolated set of functions
CU	The Control Unit
CS 445 SRS	Refers to the requirements document for the OAM Software System (*CS445 Software Requirements Specification: OAM Software for SX4*)
Daemon	A server process that services periodic requests, or performs periodic functions
DB	Database
GUI	Graphical User Interface
IP	Internet Protocol
Layer	A set of functions or routines within a component that performs a certain task.
Message-passing	A protocol in which components interact by exchanging messages. These messages constitute either events or data.
O&A	Operations and Administration
OAM	Operating, Administration and Maintenance
SRS	Software Requirements Specification
SQL	Structured Query Language
TCP	Transport Control Protocol
UI	User Interface
XML	Extensible Markup Language