



GAMMA-J

Connecting Retailers with the World

GAMMA-J USB Web Store, Attribute Driven Design (ADD) Document

Group Name GAMMA-J

Group Project Subtitle GAMMA-J Web Store Project II

Team Members:

Ashwini Chavate

Minesh Dhyani

Glenn Hollander

Avishek Shrestha

Jose Tellez

Marc Weber

Table of Contents	
Table of Contents	ii
Table of Figures	iv
Table of Tables	iv
Abstract	v
1. Introduction.....	1
2. System Definition	1
2.1 Functional Requirements	1
2.2 Design Constraints	2
2.3 Quality Attribute Requirements	2
3 Applying ADD	3
3.1 (Step 1) Confirm there is sufficient Requirements Information	3
3.2 (Step 2) Choose Element of the system to Decompose.....	4
3.3 (Step 3) Identify Candidate Architectural Drivers.....	4
3.4 (Step 4) Choose a design concept that satisfies the architectural Drivers.....	4
3.4.1 Identify Design Concerns.....	4
3.4.2 List Alternatives Patterns for Subordinate Concerns	4
3.4.3 Select Patterns from list	5
3.4.4 Determine relationship between patterns and drivers.....	6
3.4.5 Capture Preliminary Architecture Views	7
3.4.6 Evaluate and resolve Inconsistencies.....	8
3.5 (Step 5) Instantiate architectural elements and allocate responsibilities	8
3.5.1 Instantiate Instance and Assign Responsibilities to Child Elements	8
3.5.2 Allocate Responsibilities among Children	8
3.5.3 Additional Instances of Element Types	8
3.6 (Step 6) Define interfaces for instantiated elements	9
3.6.1 Functional Requirements for Elements.....	9
3.6.2 Produced and Consumed Information	9
3.7 (Step 7) Verify and refine requirements and make them constraints for instantiated elements.....	9
3.8 (Step 8) repeat steps 2 through 7 as necessary.....	9
4. ADD Second Iteration	9
4.1 (Step 1) Confirm there is sufficient Requirements Information.....	9

4.2 (Step 2) Choose Element of the system to Decompose.....	9
4.3 (Step 3) Identify Candidate Architectural Drivers.....	10
4.4 (Step 4) Choose a design concept that satisfies the architectural Drivers.....	10
4.4.1 Identify Design Concerns.....	10
4.4.2 List Alternatives Patterns for Subordinate Concerns.....	10
4.4.3 Select Patterns from list	12
4.4.4 Determine relationship between patterns and drivers.....	13
4.4.5 Capture Preliminary Architecture Views.....	13
4.4.6 Evaluate and resolve Inconsistencies.....	14
4.5 (Step 5) Instantiate architectural elements and allocate responsibilities	14
4.6 (Step 6) Define interfaces for instantiated elements	15
4.7 (Step 7) Verify and refine requirements and make them constraints for instantiated elements....	15
4.8 (Step 8) repeat steps 2 through 7 as necessary.....	15
5 Summary of architecture	15
<i>5.3 Remaining Design issues</i>	16
6 Comments on the method.....	16
7 Conclusion.....	17
8 Glossary.....	17
References	19

Table of Figures

Figure 1: Users.....	1
Figure 2: Users.....	7
Figure 3: Expanded Architecture Views.....	14

Table of Tables

Table 1: Quality Attribute Requirements.....	2
Table 2: Required Information.....	3
Table 3: Candidate Architectural Drivers.....	4
Table 4: Alternatives for Subordinate Concerns	4
Table 5: Patterns from List.....	5
Table 6: Restart Patterns	6
Table 7: Single VS Multiple Processors.....	6
Table 8: Replication	6
Table 9: Account Durability.....	6
Table 10: Error Detection Security	6
Table 11: Error Detetction Security Account Recovery	7
Table 12: Account Recovery.....	7
Table 13: Child Instances	8
Table 14: Ping Component Responsibilities.....	8
Table 15: Echo Component Responsibilities.....	8
Table 16: Heartbeat Component Responsibilities.....	8
Table 17: Voting Component Responsibilities	8
Table 18: Removal of Service Responsibilities	8
Table 19: Listening Element for Ping	8
Table 20: Candidate Architectural Drivers.....	10
Table 21: Design Concerns.....	10
Table 22: Restart Patterns	10
Table 23: Deployment Patterns	11
Table 24: Data Integrity Patterns	11
Table 25: Fault Detection Patterns.....	11
Table 26: Transparency Patterns.....	11
Table 27: Replication Patterns	11
Table 28: Patterns For Updating Client Behavior For System Failure	12
Table 29: Patterns and Associated Drivers	13
Table 30: Elements And ADD Iteration	13
Table 31: Interfaces	15
Table 32: Glossary	17

Abstract

This document uses the Attribute-Driven Design (ADD) method to define software for the GAMMA-J USB Web Store. Armed with the ADD's step by step method, functional requirements, quality attribute requirements, and constraints this document will define the system architecture for the GAMMA-J USB Web Store.

According to SEI, "the ADD method is an approach to defining a software architecture in which the design process is based on the software quality attribute requirements. ADD follows a recursive process that decomposes a system or system element by applying architectural tactics and patterns that satisfy its driving quality attribute requirements" [1].

1. Introduction

This document is a definition of the GAMMA-J Web Store USB key. An Attribute-Driven Design (ADD) method is used to define the software for this key. The ADD method is a recursive method used to flush out all the details in a software project.

2. System Definition

The GAMMA-J Web Store is designed to allow new online store owners a quick and easy means to setup and perform sales and other core business over the internet. This document will outline all the functions, capabilities and requirements for Version 1 of GAMMA-J's Web Store. Version 1 is planned for implementation on a "plug and play" USB Key. Future versions will be based on other network appliances.

2.1 Functional Requirements

- The system shall allow three types of users to log on via the internet (Administrators, Sales, and Customer)
- The system shall allow Administrators to perform all maintenance on the store as required such as: editing user properties, installing plug-ins, and installing patches.
- The system shall allow Sales personnel to create and manage store inventory
- The system shall allow customers to buy items from the store
- The system shall allow all users to review their own account information
- The system shall allow all users to edit their own personal information such as their address and password
- The system shall have a virtual "shopping cart" for customers to store items in for future purchases
- The system shall send email to sales and customer confirming order

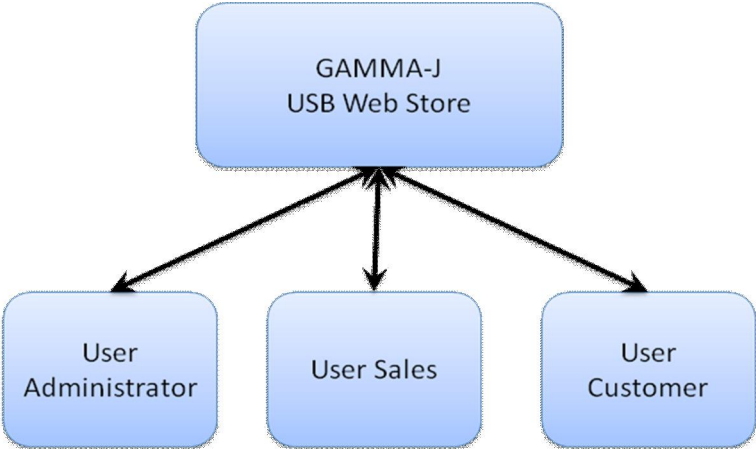


Figure 1: Users

The GAMMA-J web store provides services for three types of users:

- **Administrator:** This user will perform all maintenance on the store as required such as: editing user properties, installing plug-ins, and installing patches.
- **Sales:** The Sales user will create and manage the inventory in the store.
- **Customer:** The customer is anyone who buys inventory from the store.

2.2 Design Constraints

- The system shall be compatible with the following internet browsers: Microsoft Internet Explorer version 6 and 7, Netscape Communicator Version 4 and 5.
- The system shall operate on an Intel based system with Slackware Linux 2.6 and Apache Web Server. The operating system is designed by Yoggie Corporation. Although maintenance documentation will be supplied and the operating system will be tested, the developers of this Web Store are not responsible for functionality of the operating system.
- The system shall use MySQL 5.0 a SQL based database to store inventory information.
- USB interface and divers are provided by Yoggie Corporation.

2.3 Quality Attribute Requirements

Table 1: Quality Attribute Requirements

Element	Statement
Security	The system shall use a Secure Socket Layer (SSL) or equivalent to protect users personal information from being compromised over the internet
Availability	- Assuming infinite availability from the host system, the system shall have an availability of 99.9999% with one USB device - Additional USB web store devices shall raise the availability directly proportional to the number of devices added
Reliability	The system shall allow an optional mirror site for backups reliability
Portability	- The system shall be 100% portable to any Windows XP system or above - The system may be 100% portable to any Windows system below XP via USB driver installation - The system may be portable to other operating systems depending on independent drivers
Performance	- Except for internet connectivity the system shall operate independently from host system - Assuming infinite internet bandwidth from the host system, the system shall be able to service 1000 users simultaneously with one USB device - Multiple devices shall improve performance directly proportional to the number of devices added
Capability	- The system shall have an initial inventory capable of holding 100 items - The system shall be capably of holding a maximum of 100,000 items via administrator release codes
Upgradability	The system shall feature interface for future software enhancement via "Plug-ins"
Usability	- The system shall have an unambiguous initial interface ("store front") to assist in browsing the categories and products

	<ul style="list-style-type: none"> - The system shall have the ability to create customized user store fronts - The system shall have the ability to download predefined store fronts
--	---

3 Applying ADD

This is the first of two iterations. Further decomposition of this system and there functions will be detailed in the second iteration (Part 4)

3.1 (Step 1) Confirm there is sufficient Requirements Information

After thorough review of documentation all initial stakeholders (Yoggie Corp, developers, and investors) have decided there is sufficient requirement information.

Requirement Prioritization (L = Low, M = Medium, H = High)

Table 2: Required Information

Functional Requirement	Priority	System Requirements	Priority	Quality Requirement	Priority
Administrator Account	M	Apache Web Server	H	Security	H
Sales Account	H	MySQL Server	H	Availability 99.9999%	M
Customer Account	H	IE 6+7 Compatibility	H	Availability Additional USB devices	L
Edit account properties	M	Netscape Compatibility	L	Reliability	M
Review Account info	M	USB Drivers	L	Portability Window XP+	H
Inventory Management	H			Portability below Window XP	L
Shopping cart	H			Portability other systems	L
Order confirmation	H			Performance independently	L
				Performance 1000 users simultaneously	M
				Performance Additional USB devices	L
				Capability 100 items	H
				Capability 100,000 items	M
				Upgradability	M
				Usability default store front	H
				Usability custom store front	L
				Usability download store front	M

3.2 (Step 2) Choose Element of the system to Decompose

This is the first iteration of the ADD process. By default the only element that can be decomposed is the system itself.

3.3 (Step 3) Identify Candidate Architectural Drivers

Selected requirements for the system are the Administrator Account, Sales Account, and Customer Account. The Apache server and the SQL server will not be focused on due to being Commercial off the Shelf (COTS) product and are readily available. The Administrator account has only a medium impact because all administrator functionality could be assigned to the sales account.

Relative Impact (L = Low, M = Medium, H = High)

Table 3: Candidate Architectural Drivers

#	Architectural drivers	Importance	Impact	Difficulty
1	Administrator Account	H	M	M
2	Sales Account	H	H	M
3	Customer Account	H	H	M
4	Apache Web Server	H	H	COTS
5	MySQL Server	H	H	COTS

3.4 (Step 4) Choose a design concept that satisfies the architectural Drivers

3.4.1 Identify Design Concerns

Some of the major design concerns regarding the availability attribute are fault detection, fault recovery, and fault prevention.

3.4.2 List Alternatives Patterns for Subordinate Concerns

Table 4: Alternatives for Subordinate Concerns

Design Concern	Tactic
Fault Detection	<p>Ping/Echo: component sends signal and expects to receive a signal back within an allotted time.</p> <p>Heartbeat: component continuously sends signals and receives them.</p> <p>Exceptions: raised exception based on error handling.</p>
Fault Recovery	Voting: voter mechanism on which component gets usage

	<p>priority.</p> <p>Active redundancy: components running in parallel.</p> <p>Passive redundancy: one component listens and responds to events and relays the other components that are on standby.</p>
Fault Prevention:	<p>Removal from service: removes components from system.</p> <p>Transactions: all in one execution with error rollback feature.</p> <p>Process Monitor: monitors and removes processes and initiates new instances of processes.</p>

3.4.3 Select Patterns from list

Table 5: Patterns from List

Design Concern	Discriminating parameters
Fault Detection	<p>The period of time it takes for a restart.</p> <p>The events that cause the type of restart.</p> <p>The events that cause it to loose service.</p>
Fault Recovery	<p>The type of event that designates the type of module to handle failures.</p> <p>The type of event that continues to make it transparent.</p> <p>The type of replication processes available.</p> <p>The number of replication process patterns.</p> <p>The type of scenario to use when updating client behavior.</p>

	The consequence behind each updating scenario.
Fault Prevention	The type of system status monitoring tactic.

3.4.4 Determine relationship between patterns and drivers

Table 6: Restart Patterns

	Cold Restart	Warm Standby	Load Sharing
Availability	<p>Pros: Minimal resource usage.</p> <p>Cons: Not highly available. Highest downtime and causes loss of service.</p>	<p>Pros: Lower downtime. Medium usage of resources.</p> <p>Cons: Causes loss of service.</p>	<p>Pros: Lowest downtime and no loss of services.</p> <p>Cons: Highest usage of resources.</p>

Table 7: Single VS Multiple Processors

	One Processor	Two Processor
Availability	<p>Pros: Minimal resource usage.</p> <p>Cons: Slow Recovery Time.</p>	<p>Pros: Faster recovery time. Ability to off load recovery.</p> <p>Cons: High usage of resources.</p>

Table 8: Replication

	Primary Replication	Secondary Replication
Availability	<p>Pros: Faster replication duration.</p> <p>Cons: Low resource usage.</p>	<p>Pros: High resource usage.</p> <p>Cons: Slower replication duration.</p>

Table 9: Account Durability

	Slow Checkpoint	Fast Checkpoint	Checkpoint + Log Changes
Account Durability	<p>Pros: Low resource usage.</p> <p>Cons: Slow loading.</p>	<p>Pros: Fast loading. Ideal for large stores.</p> <p>Cons: not ideal for small stores.</p>	<p>Pros: Ideal for small stores.</p> <p>Cons: not ideal for large stores. Not good for greater than 100 messages.</p>

Table 10: Error Detection Security

	Poll	Ping
Error Detection Security	Pros: Less complex, requires less messages, and faster. Cons: Less message capacity.	Pros: Can take more messages. Cons: More complex and requires more messages.

Table 11: Error Detetction Security Account Recovery

	Continue Sending Data	Stop Sending Data	Load Log Data
Error Detection Security Account Recovery	Pros: no downtime. Cons: unusable data is sent.	Pros: saved to log file during downtime. Cons: experiences downtime.	Pros: Cons: loading time.

Table 12: Account Recovery

	Client Handles Failure	Proxy Handles Failure	System Handles Failure
Account Recovery	Pros: Handling offloads. Cons: Not transparent.	Pros: Transparent, allows recovery methods and faster. Cons: experiences downtime.	Pros: Transparent. Cons: loading time. Other processes and priorities.

3.4.5 Capture Preliminary Architecture Views

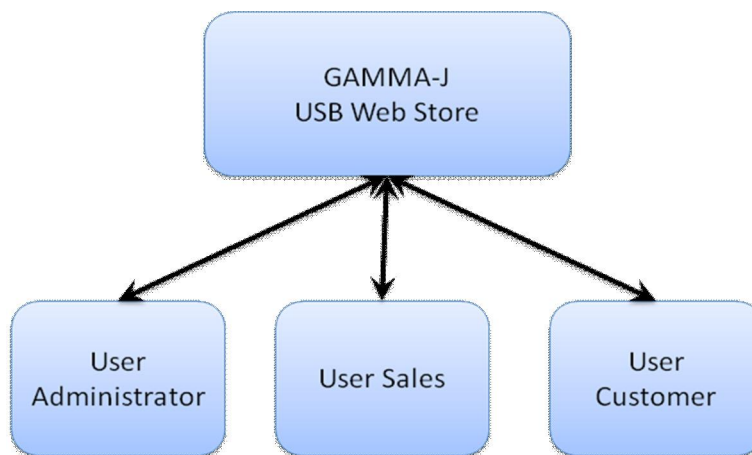


Figure 2: Users

3.4.6 Evaluate and resolve Inconsistencies

Various patterns require the usage of resources and therefore these additional patterns should be included resource management, resource arbitration and resource demand. The security architectural driver should also be considered against resisting attacks, detecting attacks, and recovering from attacks.

3.5 (Step 5) Instantiate architectural elements and allocate responsibilities

Since the only element that was identified in step 4 that could be decomposed was the system itself. The availability attribute was identified as the main component in ensuring the system is stable and availability attributes will be discussed.

3.5.1 Instantiate Instance and Assign Responsibilities to Child Elements

Table 13: Child Instances

Child Elements
Ping Component
Echo Component
Heartbeat Component
Voting Component
Removal of Service

3.5.2 Allocate Responsibilities among Children

Table 14: Ping Component Responsibilities

Child Elements	Responsibilities
Ping Component	Send a ping signal

Table 15: Echo Component Responsibilities

Child Elements	Responsibilities
Echo Component	Upon receipt of ping signal, send a ping response.

Table 16: Heartbeat Component Responsibilities

Child Elements	Responsibilities
Heartbeat Component	Continuously send and receive ping signals.

Table 17: Voting Component Responsibilities

Child Elements	Responsibilities
Voting Component	Determine priority given votes.

Table 18: Removal of Service Responsibilities

Child Elements	Responsibilities
Removal of Component	Remove a Component from the system

3.5.3 Additional Instances of Element Types

Table 19: Listening Element for Ping

Child Elements	Responsibilities
Listening Element for Ping	Listen for ping element. This is part of the Ping element and Echo element.

3.6 (Step 6) Define interfaces for instantiated elements

3.6.1 Functional Requirements for Elements

Element	Requirements
Ping Component	Send a ping signal
Echo Component	Receive a ping signal
Heartbeat Component	Send and Receive ping signals continuously
Voting Component	Determine Priority
Removal of Components	Remove Components

3.6.2 Produced and Consumed Information

Element	Produced	Consumed
Ping Component	Ping Source	n/a
Echo Component	n/a	Ping Source
Heartbeat Component	Ping Source	Ping Source
Voting Component	n/a	Voting Information
Removal of Components	List of Removed Components	n/a

3.7 (Step 7) Verify and refine requirements and make them constraints for instantiated elements

Identified software elements do meet all functional requirements, quality attribute requirements, and design constraints.

3.8 (Step 8) repeat steps 2 through 7 as necessary

After steps 1-7 are completed, we have a decomposition of the fault-tolerance service of the Customer Account element. Steps 2 through 7 can be completed as necessary for the other elements.

4. ADD Second Iteration

4.1 (Step 1) Confirm there is sufficient Requirements Information

This step was already accomplished in the first iteration of the process.

4.2 (Step 2) Choose Element of the system to Decompose

The fault-tolerance service of the Customer Account element is chosen to be decomposed because we want to ensure that our system can detect faults, recover from faults, and prepare itself from faults in future occurrences. Most importantly, the Customer Account must be fault tolerant, as it is priority to ensure the customers' data remains confidential. However, the other components must also be fault tolerant.

4.3 (Step 3) Identify Candidate Architectural Drivers

Relative Impact (L = Low, M = Medium, H = High)

Table 20: Candidate Architectural Drivers

#	Architectural drivers	Importance	Impact	Difficulty
1	Account Durability	M	M	H
2	Account Recovery	M	M	M
3	Error Detection	H	H	H
4	Availability	H	H	L
5	Security	H	M	M

4.4 (Step 4) Choose a design concept that satisfies the architectural Drivers

4.4.1 Identify Design Concerns

The three design concerns associated with the fault-tolerance services are:

Fault Preparation: This concern is the methods taken to ensure that when a failure occurs, a recovery is able.

Fault Detection: This concern is the methods taken to detect the failure and notify the system of the failure.

Fault Recovery: This concern is the methods taken to restore normal operation.

Table 21: Design Concerns

#	Design Concern	Subordinate Concerns	Alternative Patterns Section	Selecting Design Patterns Section
1	Fault Preparation	Restart	4.4.2.1	4.4.3.1
		Deployment	4.4.2.2	4.4.3.2
		Data Integrity	4.4.2.3	4.4.3.3
2	Fault Detection	System Status	4.4.2.4	4.4.3.4
3	Fault Recovery	Transparency	4.4.2.5	4.4.3.5
		Replication	4.4.2.6	4.4.3.6
		Update Client Behavior	4.4.2.7	4.4.3.7

4.4.2 List Alternatives Patterns for Subordinate Concerns

4.4.2.1 Restart

The four design alternatives for restarting a failed component are displayed with two associated parameters in Table 6:

Table 22: Restart Patterns

#	Pattern	Downtime Estimates	Loss of Service
1	Cold Restart	> 2 Minutes	Yes
2	Warm Standby	> 0.5 seconds	Yes

3	Load Sharing	> 0.05 seconds	No
---	--------------	----------------	----

4.4.2.2 Deployment

The two components are deployed with two scenarios:

- 1) Both primary components on one processor
- 2) Each primary component on a separate processor

The primaries are denoted by A and B, and the secondaries are denoted by A` and B`.

Table 23: Deployment Patterns

#	Pattern	Processor 1	Processor 2	A Failure	Recovery Time (seconds)
1	One Processor	A, B	A`, B`	A`, B`	2
2	Separate Processor	A, B`	A`, B	A`, B	0.5

4.4.2.3 Data Integrity

The data integrity provided a method that when a failure occurs, the secondary has information to proceed. The patterns are shown in Table 8.

Table 24: Data Integrity Patterns

#	Pattern	Communication Loading
1	Slow Checkpoint	1.2 seconds per every minute
2	Fast Checkpoint	1.2 seconds per every 5 minutes
3	Checkpoint + Log Changes	1.2 seconds per every minute + 50 messages per second

4.4.2.4 System Status

A single system status monitoring tactic is sufficient for fault detection. Table 9 lists patterns to consider and the associated parameters

Table 25: Fault Detection Patterns

#	Pattern	Communication Loading
1	Poll	2 messages (for A, B)
2	Ping	4 messages (for ping of A, A`, B, B`) 4 messages (for echo of A, A`, B, B`)

4.4.2.5 Transparency

The three alternatives to making the faults transparent to the client are displayed in Table 10 below.

Table 26: Transparency Patterns

#	Pattern	Location	Transparency
1	Client Handles Failure	Client	None
2	Proxy Handles Failure	Proxy	Yes
3	System Handles Failure	System	Yes

4.4.2.6 Replication

Replication is necessary when a fault is detected and the element is replicated to ensure transparency for the client. The two alternatives for replication methods and associated parameters are displayed in Table 11 below. The primary process is denoted by A. The secondary process is denoted by B. The replicated processes are denoted by A` and B`.

Table 27: Replication Patterns

#	Pattern	Original Process	Replicated Process	Duration
---	---------	------------------	--------------------	----------

1	Primary Replication	A	A`	> 0.5 seconds
2	Secondary Replication	B	B`	> 2 seconds

4.4.2.7 Update Client Behavior

There are two primary scenarios for updating the client behavior.

- 1) The poll system notifies the proxy of a failure, and the proxy handles the failure. When the proxy acknowledges the failure, the system promotes the secondary to the primary. Then the system discards the failed secondary, and creates a replication of the primary.
- 2) System handled failures are shown below in Table 12.

Table 28: Patterns For Updating Client Behavior For System Failure

#	Pattern	Consequence
1	Continue Sending Data	Unusable data is sent to the proxy and client
2	Stop Sending Data	The communication during downtime is saved to a log
3	Load Log Data	The communication log is loaded

4.4.3 Select Patterns from list

4.4.3.1 Restart

Rationale: The Cold Restart scenario does not meet the quality attribute requirements of high availability. Consequently the Warm Standby scenario is chosen.

Decision: Use the Warm Standby

4.4.3.2 Deployment

Rationale: The Two Processor scenario is chosen because the duration is shorter. If a Two Processor scenario is not available, the system will resort to a One Processor scenario.

Decision: Use the Two Processor

4.4.3.3 Data Integrity

Rationale: The Slow Checkpoint requires 1.2 seconds of processing time for every minute, while the Fast Checkpoint requires 1.2 seconds of processing time for every 5 minutes. However, the Checkpoint + Log Changes method can be faster if less than 100 messages are processed in more than 5 minutes. This is the best scenario for small stores with infrequent messages.

Decision: Checkpoint + Log Changes scenario for small stores with fewer than 100 messages for 5 minutes or more. Fast Checkpoint scenario for large stores with many transactions.

4.4.3.4 System Status

Rationale: The ping and echo method is more complex than the polling method. And requires four times the messages.

Decision: Poll method with each poll set to 0.25 seconds.

4.4.3.5 Transparency

Rationale: It is preferred for the client to not handle the failures. Additionally, offloading the transparency to the proxy allows the proxy to incorporate recovery methods of its own, usually faster than the system since the system has other processes and priorities.

Decision: Proxy

4.4.3.6 Replication

Rationale: Primary replication scenario is faster than secondary replication and is consequently used.

Decision: Primary replication scenario.

4.4.3.7 Update Client Behavior

Rationale: A failure is detected when the polling system detects a failure and informs the proxy. The proxy then restores communication after the system promotes the secondary. The system then discards the secondary and creates a replication of the primary.

Decision: Poll and proxy scenario.

4.4.4 Determine relationship between patterns and drivers

The relationship between patterns and drivers are shown in Table 13 below.

Table 29: Patterns and Associated Drivers

#	Pattern	Pattern Chosen	Architectural Driver
1	Restart	Warm Standby	Availability
2	Deployment	Two Processors	Availability
3	Data Integrity	Checkpoint + Log Changes	Account Durability
4	System Status	Poll	Error Detection Security
5	Transparency	Proxy	Account Recovery
6	Replication	Primary	Availability
7	Update Client Behavior	Poll and Proxy	Error Detection Security Account Recovery

4.4.5 Capture Preliminary Architecture Views

Table 14 shows the elements of the ADD iteration and which they are developed.

Table 30: Elements And ADD Iteration

#	Element	ADD Iteration
1	Administrator Account	1
2	Sales Account	1
3	Customer Account	1
4	Apache Web Server	Requirement
5	MySQL Server	Requirement

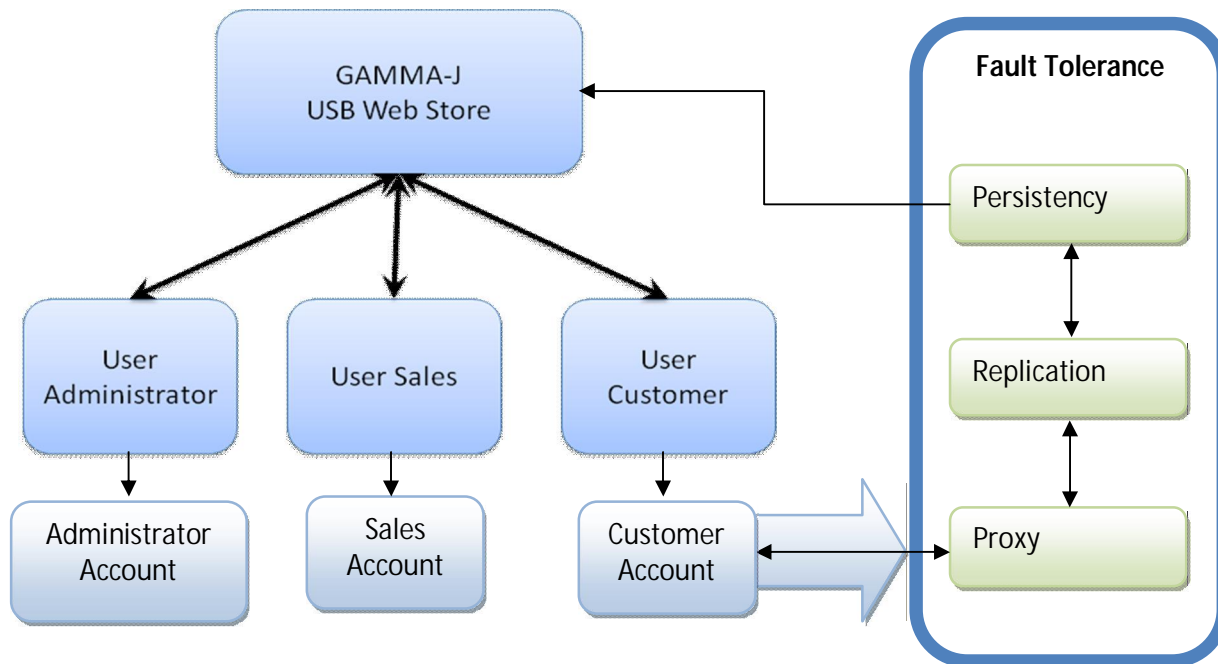


Figure 3: Expanded Architecture Views

4.4.6 Evaluate and resolve Inconsistencies

Inconsistencies may occur when replicating instances of A and B. However, the replication service will be designed to ensure that A and A` are on two separate processes, and B and B` are on two separate processes. A and B` will be together, and A` and B will be together.

Furthermore, the proxy will work in conjunction with the replication service to provide the client with a seamless transition from normal operation to fault recovery. In the event of a fault detection, the persistent state of A and B will be recovered using checkpoints and log files.

4.5 (Step 5) Instantiate architectural elements and allocate responsibilities

4.5.1 Primary A and B

The primary A and replication of B occupy the same processor. In the event of a fault, the secondary of A` is promoted. Now, A is demoted and killed. A` is replicated and stored as the secondary A. A poll detects the error, and the proxy minimizing downtime to the client by allowing the connection to A` while A is being replicated and stored as the secondary.

4.5.2 Persistency

There are four storage files: CheckpointA, CheckpointB, LogA, and LogB.

4.5.3 System Status

The poll uses a timer to check if it has received an acknowledgement for A and B. If any of the acknowledgements fail to be received, the system will notify the proxy.

4.5.4 Proxy

The proxy ensures that the client experiences minimal downtime when the system transfers from the primary to the secondary.

4.6 (Step 6) Define interfaces for instantiated elements

Table 15 displays the interfaces for the instantiated elements

Table 31: Interfaces

#	From Element	To Element
1	Customer Account	Proxy
2	Proxy	Replication
3	Proxy	Customer Account
4	Replication	Persistency
5	Replication	Proxy
6	Persistency	GAMMA-J USB Web Store API
7	Persistency	Replication
8	GAMMA-J USB Web Store API	User Customer
9	GAMMA-J USB Web Store API	User Sales
10	GAMMA-J USB Web Store API	User Administrator
11	User Customer	Customer Account

4.7 (Step 7) Verify and refine requirements and make them constraints for instantiated elements

We verify that the decomposition of the fault-tolerance service supporting the Customer Account meet the functional requirements, quality attribute requirements, and design constraints.

In 4.4.3 we show how we determine which patterns are chosen for the fault-tolerance element supporting the Customer Account architectural driver shown in Table 3 and determined in 4.2.

4.8 (Step 8) repeat steps 2 through 7 as necessary

After steps 1-7 are completed, we have a decomposition of the fault-tolerance service of the Customer Account element. Steps 2 through 7 can be completed as necessary for the other elements defined in Table 3.

5 Summary of architecture

5.1 Architecture Summary

The summary of design is given below.

- | The two components are deployed with two scenarios. Both primary components on one processor. Each primary component on a separate processor
- | The primaries are denoted by A and B, and the secondaries are denoted by A' and B'.
- | The data integrity provided a method that when a failure occurs, the secondary has information to proceed.
- | A single system status monitoring tactic is sufficient for fault detection.

- | The three alternatives to making the faults transparent to the client occurs when "Client Handles Failure", "Proxy Handles Failure", or "System Handles Failure".
- | Replication is necessary when a fault is detected and the element is replicated to ensure transparency for the client. The two alternatives for replication methods and associated parameters are displayed in Table 11 below. The primary process is denoted by A. The secondary process is denoted by B. The replicated processes are denoted by A` and B`.
- | The poll system notifies the proxy of a failure, and the proxy handles the failure. When the proxy acknowledges the failure, the system promotes the secondary to the primary. Then the system discards the failed secondary, and creates a replication of the primary.
- | The Cold Restart scenario does not meet the quality attribute requirements of high availability. Consequently the Warm Standby scenario is chosen.
- | The Two Processor scenario is chosen because the duration is shorter. If a Two Processor scenario is not available, the system will resort to a One Processor scenario.
- | The Slow Checkpoint requires 1.2 seconds of processing time for every minute, while the Fast Checkpoint requires 1.2 seconds of processing time for every 5 minutes. However, the Checkpoint + Log Changes method can be faster if less than 100 messages are processed in more than 5 minutes. This is the best scenario for small stores with infrequent messages.
- | The ping and echo method is more complex than the polling method. And requires four times the messages.
- | It is preferred for the client to not handle the failures. Additionally, offloading the transparency to the proxy allows the proxy to incorporate recovery methods of its own, usually faster than the system since the system has other processes and priorities.
- | A failure is detected when the polling system detects a failure and informs the proxy. The proxy then restores communication after the system promotes the secondary. The system then discards the secondary and creates a replication of the primary.
- | Inconsistencies may occur when replicating instances of A and B. However, the replication service will be designed to ensure that A and A` are on two separate processes, and B and B` are on two separate processes. A and B` will be together, and A` and B will be together.
- | The poll uses a timer to check if it has received an acknowledgment for A and B. If any of the acknowledgments fail to be received, the system will notify the proxy.

5.2 Design Issues Being Resolved Elsewhere

Some designs are being resolved elsewhere:

- | the mechanisms for the persistent storage
- | the start-up procedures for A and B and their coordination

5.3 Remaining Design issues

- | How does the system respond to a failure in a secondary component?
- | How do proxy elements recover from failures in the software or hardware, and how they are distributed?

6 Comments on the method

The ADD method is an approach to defining a software architecture in which the design process is based on the software quality attribute requirements. ADD follows a recursive process that decomposes a system or system element by applying architectural tactics and patterns that satisfy its driving quality attribute requirements.

We made the following observations while writing this report:

1. A good understanding was established among topics like fault tolerance, alternative patterns which were compatible with timelines effective for the particular choice of model.
2. This documentation of software architecture description is not dependent on the development method (ADD) but rather on the most effective way of capturing the views developed.
3. The developer chose to develop the architecture iterations, which were resulted in alternative patterns to be represented in a matrix.

ADD weakly implies that the development of an architecture is done sequentially—at each iteration, an element is chosen for design elaboration, all architectural drivers are known before starting the design of the element, and this iteration’s results are then used in the next iteration

7 Conclusion

This document uses the Attribute-Driven Design (ADD) method to define software for the GAMMA-J USB Web Store. Armed with the ADD’s step by step method, functional requirements, quality attribute requirements, and constraints this document will define the system architecture for the GAMMA-J USB Web. This report also clearly distinguishes between choosing well-known patterns during the first iteration and the alternatives for improvement during the second iteration. In the first iteration, straight forward choices were made from predefined patterns; whereas in the second, alternatives included changing a number of design aspects to achieve the improved qualities.

8 Glossary

Table 32: Glossary

Design constraints	Design constraints are decisions about the design of a system that must be incorporated into any final design of the system. They represent a design decision with a predetermined outcome
Property	A property is additional information about a software element such as name, type, quality attribute characteristic, protocol, and so forth
Quality Attribute	A quality attribute is a property of a work product or goods by which its quality will be judged by stakeholders. Quality attribute requirements such as those for performance, security, modifiability, reliability, and usability have a significant influence on the software architecture of a system.
Quality Attribute Requirements	Quality attribute requirements are requirements that indicate the degrees to which a system must exhibit various properties
Functional Requirements	Functional requirements specify what functions a system must provide to meet stated and implied stakeholder needs

	when the software is used under specific conditions.
Design Constraints	Design constraints are decisions about the design of a system that must be incorporated into any final design of the system. They represent a design decision with a predetermined outcome.

References

[1] Wojcik, Rob et al. *Attribute-Driven Design (ADD) Version 2.0*. Carnegie Mellon: Software Engineering Institute. 2006.

[2] [SEI 2007]

Software Engineering Institute. *Software Architecture Glossary*.
<<http://www.sei.cmu.edu/architecture/glossary.html> (2007)>.