Legacy Interface Adapter Design Modeling: Part-I

Lonney J. Head, *Raytheon Company, Dallas, Texas, USA*

doi: 10.22545/2015/00059

Technology is changing at a very rapid rate. Many deployed/fielded information systems are aging and becoming legacy systems that continue to operate and perform as required. However, as time goes by, maintainability of these systems is becoming an increasing issue. Also, the dissemination of information and data from the legacy systems to modernized systems is becoming more prominent within industry. With the technology evolution, there is a need to adapt legacy systems to modern architected systems. The objective of this project is to educate the reader about available design considerations and processes to consider when developing an adaptor type interface with a legacy system. An interface adapter example is utilized throughout this paper to provide the reader with sufficient information to get started on their own design. This paper is intended to visit several design topics and processes; component-oriented axiomatic design, architectural considerations, and project planning. Through the survey of these topics, the reader will have a framework and a model in which to get a head start. Many of these topics are cross disciplinary in nature and may be used with a variety of systems.

Keywords*:* Legacy system, adapter software design, axiomatic design .

1 Introduction

Customers who operate existing legacy data processing systems often have a need to interface those existing legacy systems to new or replacement subsystems. Legacy systems often provide stable low cost processing, but they may need to interface with new/replacement subsystems.

These new or replacement subsystems by their nature utilize more current technology than the legacy system. However, attempting to modify the legacy system directly to interface with the newer subsystem causes an undesirable and often unaffordable ripple of change in the legacy system. Hence, a need arises for an Interface Adapter that fits between the legacy system and the newer subsystem. The adapter can absorb the ripples of change so the newer subsystem can be designed taking advantage of the newest available technology and the legacy system is either not impacted or minimally impacted by this interface modification.

This paper leverages the axiomatic design of the Interface Adapter Software Design (IASD) project. The set of customer needs, Functional Requirements (FRs), Design Parameters (DPs) and Constraints applicable to an IASD were developed and will be utilized to the fullest extent. The full design process, description of customer needs (CNs), Functional Requirements decomposition process, associated Design Parameters and Requirements traceability matrix are described in Chapter III.

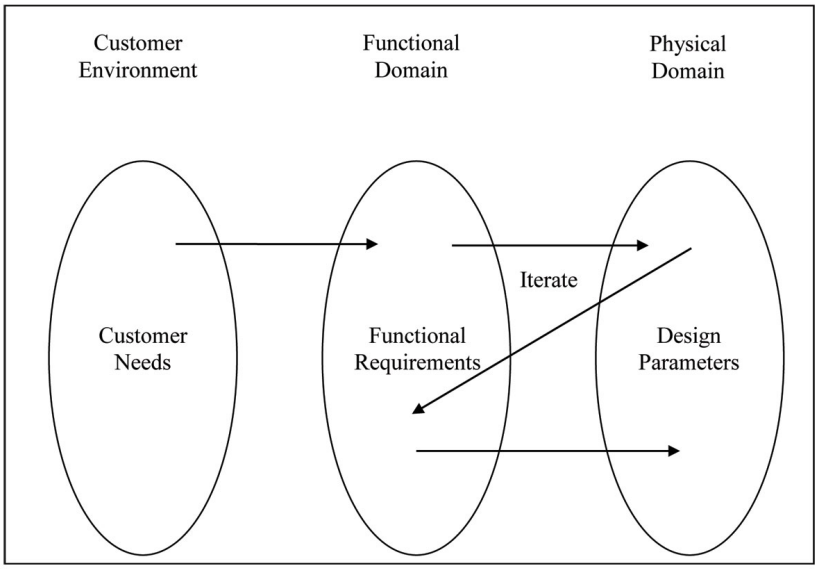

Figure 1: *Axiomatic design domains involved in the IASD design [Tate, 2006].*

The vertical requirement decomposition method/tool is utilized to decompose the system design requirements. The first level requirements are developed to describe the highest level of the design description, and an initial/high level design matrix is developed and decoupled to start the hierarchical vertical decomposition approach. An iterative decomposition analysis approach is utilized to refine the various levels after analyzing coupling in the design matrices.

Although this paper is based around the IASD design using component oriented axiomatic design techniques, architectural considerations are leveraged in order to validate the design and to systematically develop the design to a level that could lead directly to implementation.

2 Interface Adapter Software Design (IASD)

2.1 Component Oriented Axiomatic Design

Component Oriented Axiomatic Design is an approach that utilizes specific processes to develop a system design to the component level while checking for missing components. Identifying missing components early in the design process is crucial to maintain cost, stay on schedule and enhance performance on the project. Also, missing a component may be detrimental to the success of the project. The earlier missing components are found during the design process, the less time and money that are spent later in the project lifecycle. The design team should consider keeping the customer informed throughout the design process or, better yet, have them actively participate on the team.

2.2 Architectural Considerations

The second approach described in this paper utilizes an architectural design approach that combines contextual, operational, logical and physical data to provide a base architecture. This approach, similar to the axiomatic design approach, uses customer needs and requirements as a basis for the architecture of the system. The difference between the two methods is the detailed processes utilized to describe each part of the system. These details allow a smooth transition to system hardware and software development.

2.3 Project Planning

Project planning occurs throughout all stages of project and is considered an evolving process. However, it is important to put together an initial project master plan and integrated master schedule. This is essential to successful project start up and initial project execution. Although it is highly probable that changes will occur during project execution for various reasons, continual adaptation of the master plan and schedule are essential for project comple-

tion. The IASD project Integrated Master Plan (IMP) and Integrated Master Schedule (IMS) are developed in this paper so that the reader may gain an understanding of how to approach project planning in a systematic manner.

3 Component Oriented Axiomatic Design

This paper uses axiomatic design techniques to present such an Interface Adapter Software Design (IASD). The report iteratively develops a list of customer needs, Functional Requirements, Design Parameters and Constraints applicable to an IASD and analyzes the design steps pointing out desirable and undesirable characteristics. The axiomatic design approach is presented in Figure 1.

3.1 Defining Customer Needs

The project objective is to design an interface between an existing legacy data system and multiple modernized data systems with specific interface requirements. This project utilizes the Component-Oriented Axiomatic Design (COAD) process to provide the best possible design decisions based upon customer needs and decomposed functional requirements. The customer vision is to, "provide a reliable, cost effective mechanism to interface a legacy data system to a modernized data system with minimum disruption to ongoing operations."

The customer needs were identified through brainstorming sessions and reviewing past industry project experiences. Customer needs would normally be determined in projects of this category by a consortium of personnel including:

- System design engineers
- Legacy knowledgeable systems engineers
- New subsystem design team
- Software engineers and architects
- Network engineers
- Field personnel who operate or maintain legacy systems
- Program management, scheduling, budgeting, etc.

The scenario under examination is a legacy system in which a subsystem is identified and replaced. This new, replacement subsystem uses current communication technology and protocols but the legacy system operates older technology. The primary desire is to develop a testable adapter to translate the two communication protocols (old and new).

3.2 Customer Needs and Constraints

The following information describes an initial list of customer needs, known or agreed constraints and definitions/declarations.

1. Adapter shall access subsystem replacement.
2. Adapter shall interface legacy system to subsystem replacement using specified communication protocols.
3. Customer needs a subsystem replaced because manufacturers are not supporting aging parts of system and/or customer needs to migrate parts of system to newer and more flexible technology and make software based.
4. Customer needs more functionality but the legacy system contains subsystems that are functionally limited.
5. Limited funding exists for overhauling the entire system so focus is on subsystem replacement which is more cost effective.
6. Customer needs additional functionality added to the system.
7. The customer needs more current technology for the resulting functionality improvement.
8. Various stakeholders need to be able to test the new subsystem in a stand alone configuration (without access to the legacy system).
9. Various stakeholders need to be able to verify requirements of a new subsystem by having visibility to data-flow within the adapter.
10. Customer needs to utilize standardized technology to make development more cost effective and gain more functionality (get more for less).
11. The current legacy system is incompatible with newer COTS products.
12. The Customer needs data flow improvement to take advantage of increased bandwidth and capacity of communication equipment, (i.e. network switches).

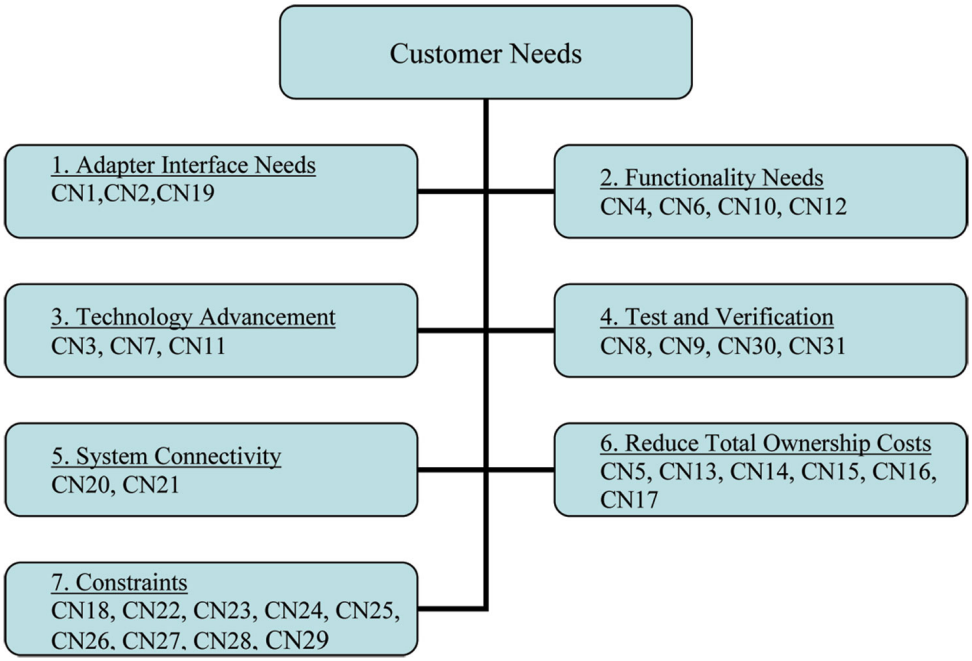

Figure 2: *Grouped customer needs.*

13. Customer needs to reduce equipment footprint size to use less space.
14. Customer needs reduced power requirements.
15. Customer needs reduced heat generation.
16. Customer needs improved reliability.
17. Customer needs improved maintainability.
18. Customer needs system availability to be at least as good as the current system (24x7 with scheduled maintenance periods - 1000 hours or better continual uptime).
19. Adapter shall be considered part of legacy system.
20. If one subsystem channels connectivity is lost, the system considers all are lost.
21. If no response is received from the subsystem or connection is lost on any of the multiple communication channels, all communication with the subsystem is halted.
22. Legacy interface is TCP-IP socket based.
23. Secondary interface shall be Service Oriented Architecture (SOA).
24. Adapter shall be software based running on a standard computer platform.
25. Adapter shall maintain SOA connectivity with Naming Service to resolve single server.
26. Adapter shall provide a sustainable TBD commands per second commanding rate.
27. Adapter design shall scale to match available hardware.
28. For operational mode, adapter does not have to operate if legacy system is not available.
29. Publicly exposed data references in the adapter should be protected (not deleted).
30. Adapter shall trace events to provide diagnostic access to processing.
31. Test scenarios shall utilize the legacy interface of the adapter to test the adapter and subsystem.

The list of customer needs is then grouped into higher level categories. These categories are areas of interest that describe the higher level needs by the customer. This method also helps flush out constraints that may affect the overall design of the final system. Figure 2 shows the KJ diagram related to the Customer Needs described above. It is often helpful to view the relationships between the customer needs and the categories into one diagram for quick identification of customer needs and constraints.

3.3 Mature Domain

Using knowledge from experience and technology used in similar systems, the following diagrams were

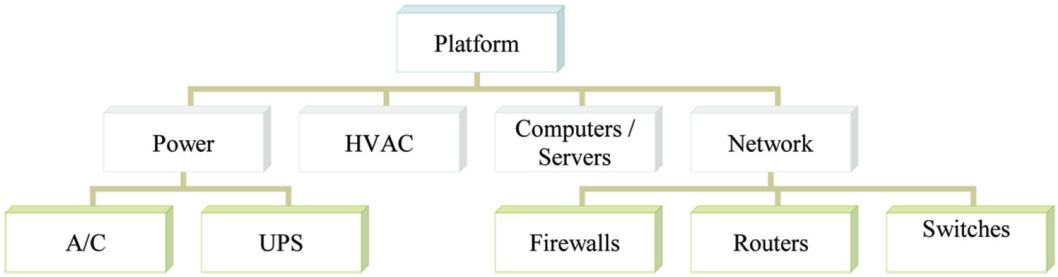

Figure 3: *Mature domains (platform).*

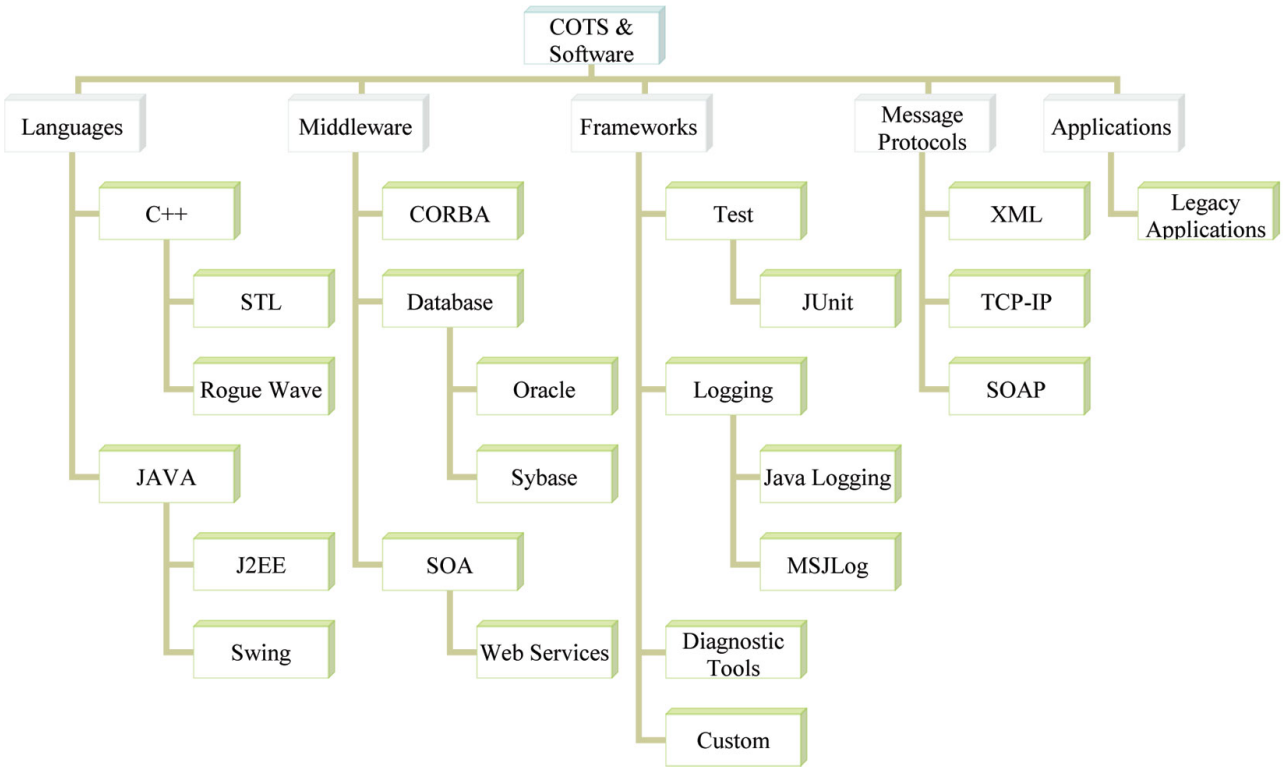

Figure 4: *Mature domains (COTS & Software).*

developed to show possible mature domains that could be incorporated into the design. These domains will be used to further develop requirements and design parameters. The platform technologies listed in Figure 3 and the COTS and Software technologies listed in Figure 4 show mature domain components that map to the IASD requirements. This provides a component relationship of the IASD requirement so existing mature domain components already available in the industry. These mature domains change over time due to the rapid changes in technology, so taking a snapshot in time may lead to changes down the road. As in the description of "legacy system", today's mature domains are tomorrows legacy domains.

The following Constraints emerged during the initial stage of the axiomatic design process. The constraints are derived through the analysis of customer needs and act of mapping them to the mature domain components categorized for this application. Constraint C10 emerged later in the design process.

- C 1 Legacy interface shall be TCP-IP socket based.
- C 2 Secondary interface shall be SOA.
- C 3 Adapter shall be software based running on a standard computer platform.
- C 4 Maintain SOA connectivity with Naming Service.

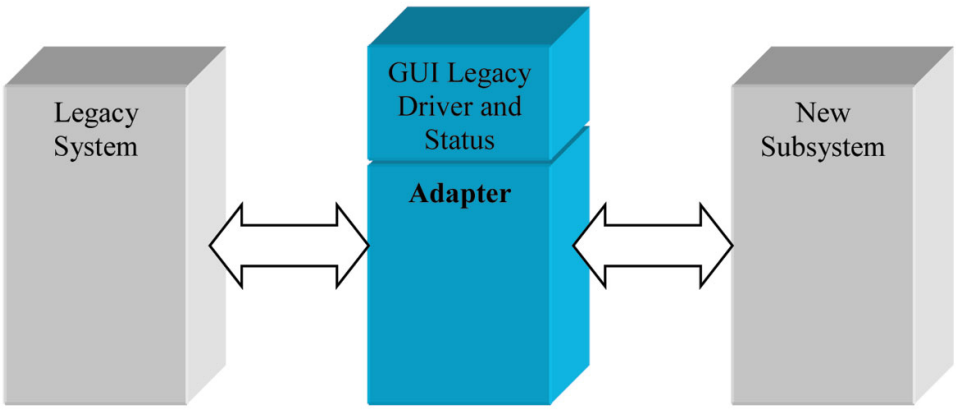

Figure 5: *Base IASD design.*

- C 5 Provide a sustainable TBD commands per second commanding rate. C 6 Provide for 24 hour, 7 day per week duty cycle with a MTBF (Mean Time Between Failure) of 1000 hours or better.
- C 7 Scalable to match available hardware.
- C 8 In operational mode, adapter does not have to operate if legacy system is not available. C 9 Protect (do not delete) publicly exposed data references in adapter.
- C10* Event log service is available to all parts of the adapter for logging trace events.

These system constraints define the adapters functional bounds. They also define the inputs for which the adaptor is responsible.

3.4 Functional Requirements and Design Parameters Decomposition

The adapter is assumed to operate between the legacy and the new subsystem where it can translate data transmitted in either direction. This position also allows a legacy driver to be connected to this adapter for testing and operation without the need for the legacy system. Figure 5 shows the relationship between the legacy system, adapter (IASD) and a new subsystem.

As shown in Figure 1, Functional Requirements (FRs) are developed from Customer Needs (CNs) and Design Parameters (DPs) are developed from the FRs. Each FR captures the concise scope of functions for the given level. Keeping a focus on customer needs insures all system needs are accounted for.

In this case, DP development evolves easily from FRs with the perspective that a DP satisfies everything within the scope of the FR. When the Design Matrix is used to examine coupling, the scope of DPs is made much clearer. This exposes the DP to multiple FR mappings. This process reveals additional information about the FRs (i.e. definition is too broad, functionality is excessive, or that the definition is optimal). By iteratively going over a level of FR/DP development based on Design Matrix analysis, a more sound design is realized as more concise FRs/DPs are realized.

Note: In the next sections, the FR and DP numbering scheme is formed as follows; $< Level\_1 > . < Level\_2 > . < Level\_3 >$. The sub-level numbers are sequential for the entire level to reflect a FR or DP at the given level, not within the higher levels FR/DP. This numbering scheme is preferable when working with design matrices, specifically when a matrix needs to be manipulated into lower triangular form.

3.4.1 Level 1 FRs/DPs

The following FRs are derived for the first level. The primary function of the adapter is to translate data between two interfaces and secondarily provide a user interface for testing the new subsystem.

- FR 1 Interface a legacy system to an upgraded subsystem.
- DP 1 Data Translator Adapter.
- FR 2 Provide a user driven stand-in legacy driver for testing and verifying new subsystem.

Table 1: Level 1 FR/DP Design Matrix

Level 1	DP1	DP2
FR1	X	
FR2	X	X

Table 2: Level 2 FR/DP Initial Design Matrix

Level 2	DP 1.1	DP 1.2	DP 1.3	DP 1.4	DP 2.5
FR 1.1	X		X		
FR 1.2		X	X		
FR 1.3			X		
FR 1.4		X	X	X	
FR 2.5			X		X

Table 3: Level 2 FR/DP Decoupled Design Matrix

Level 2	DP 1.3	DP 1.2	DP 1.1	DP 1.4	DP 2.5
FR 1.3	X				
FR 1.2	X	X			
FR 1.1	X		X		
FR 1.4	X	X		X	
FR 2.5	X	X			X

- DP 2 Legacy Interface Test Driver and Status GUI.

This matrix shown in Table 1 represents the decoupled relationship between FR1/DP1 and FR2/DP2. This could have been a completely uncoupled matrix (Xs only in diagonal locations), except to satisfy the legacy interface test driver requirement, FR2; the GUI must have access to the Adapter.

3.4.2 Level 2 FRs/DPs

The following level 2 FR/DP pairs are derived by analysis of the customer needs. In this case, the foundational functionality emerged from brainstorming the logical separations. Since it is assumed the implementation of the adapter is completed using a standard software language, the level 2 DPs are envisioned as primary functional components in a software design framework.

- FR 1.1 Provide legacy interface to receive and transmit data with legacy system.
- DP 1.1 Legacy Interface.
- FR 1.2 Provide Subsystem interface to receive and transmit data with the new subsystem.
- DP 1.2 Subsystem Interface.
- FR 1.3 Translate/manage entire adapter functionality
- DP 1.3 Adapter Manager
- FR 1.4 Data needs to be inspected and checked against state of data on subsystem interface to determine where a message should be routed (message dependent or independent of current state)
- DP 1.4 Data Marshaller to determine routing to subsystem
- FR 2.5 Operate adapter via legacy driver without legacy system being available
- DP 2.5 Legacy Driver / GUI Portal

Reworking the matrix into lower triangular form in Table 2, the following decoupled version shown in Table 3 is obtained. The Adapter Manager obviously has a lot of coupling to the rest of the system, but this is by intent to try to keep other parts focused on specific functions. This coupling and others may be reduced during the level 3 FR/DP analysis.

3.4.3 Level 3 FRs/DPs

The following level 3 FR/DP pairs are also derived by analysis of the customer needs.

Table 4: Iteration 1 Level 3 FR/DP Initial Design Matrix

Level3	DP1.3.1	DP1.1.2	DP1.2.3	DP1.3.4	DP1.4.5	DP1.2.6	DP1.3.7	DP2.5.8	DP2.5.9	DP1.3.10
FR1.3.1	X	X	X				X			
FR1.1.2		X		X			X			X
FR1.2.3			X	X			X			X
FR1.3.4		X	X	X			X			X
FR1.4.5			X		X	X	X			X
FR1.2.6			X			X	X			X
FR1.3.7	X	X	X	X	X	X	X	X	X	X
FR2.5.8		X					X	X		X
FR2.5.9							X		X	
FR1.3.10		X	X				X			X

Table 5: Iteration 1 Level 3 FR/DP Design Matrix

Level3'	DP1.3.1	DP1.1.2	DP1.2.3	DP1.3.4	DP1.4.5	DP1.2.6	DP2.5.8	DP2.5.9	DP1.3.10
FR1.3.1	X	X	X						
FR1.1.2		X		X					X
FR1.2.3			X	X					X
FR1.3.4		X	X	X					X
FR1.4.5			X		X	X			X
FR1.2.6			X			X			X
FR2.5.8		X					X		X
FR2.5.9								X	
FR1.3.10		X	X						X

- FR 1.3.1 Configure adapter using a configuration input
- DP 1.3.1 Configuration File
- FR 1.1.2 Interface with Legacy System to transfer data to/from adapter
- DP 1.1.2 Legacy Interface
- FR 1.2.3 Interface with Subsystem to transfer data to/from adapter
- DP 1.2.3 Subsystem Interface
- FR 1.3.4 Manage conversion of data in both directions via defined protocols
- DP 1.3.4 Adapter Manager Data Converter
- FR 1.4.5 Route data to/from multiple channels on subsystem interface
- DP 1.4.5 Data Marshaller
- FR 1.2.6 Initialize when legacy interface is triggered to be active against legacy subsystem or test driver
- DP 1.2.6 Legacy Interface Initializer
- FR 1.3.7 Trace events
- DP 1.3.7 Event Log
- FR 1.2.8 Subsystem Interface will input configuration for a given number multiple channels
- DP 1.2.8 Subsystem Interface Initializer
- FR 1.5.9 Manage/coordinate initialization sequence of all adapter functions
- DP 1.5.9 Adapter manager Initializer FR 1.3.10 Sequence/coordinate fault recovery/response
- DP 1.3.10 Adapter Manager Fault Handler

Table 4 shows the Design Matrix realized from the first iteration of FR/DP pairs. Tight coupling between FR1.3.7 and DP 1.3.7 is created because of the event tracing via an event log. This FR/DP is removed from the matrix by defining a new constraint. The new constraint is added as (C10) to ensure this need is covered and to reduce coupling in the matrix. Since the adapter is a software based application, a common software log service can be made available to all parts of the system. See the results below in Table 5.

Table 6: Final Iteration 1 Level 3 FR/DP Design Matrix

Level3''	DP1.3.10	DP1.2.3	DP1.1.2	DP1.3.4	DP1.2.6	DP2.5.9	DP1.3.1	DP1.4.5	DP2.5.8
FR1.2.3	X	X		X					
FR1.1.2	X		X	X					
FR1.3.10	X	X	X						
FR1.3.4	X	X	X	X					
FR1.2.6	X	X			X				
FR2.5.9						X			
FR1.3.1		X	X				X		
FR1.4.5	X	X			X			X	
FR2.5.8	X		X						X

The level 3 design matrix is then manipulated toward lower triangular form to produce Table 6.

Before going further, the design matrix reveals coupling that is difficult to account for in the current level 3 FR/DP definitions. The FR/DPs involving the Legacy Interface and the Subsystem Interface are coupled to other parts of adapter. The single fault handler function is a contributor to this coupling as well as the data conversion being handled in one location.

The analysis shifts back to reexamining the level 3 FR/DPs to more carefully look where functions are performed against the level 2 DPs. The fault handling is uncoupled by breaking out the respective types of fault handling required and assigning them to respective parts that deal with that portion of the adapter functions. Also, the data conversion is assigned to the respective interfaces so other parts of the adapter do not have to be a part of the specific data protocols.

3.4.4 Level 3 FRs/DPs (Revised and Grouped)

Legacy I/F

- FR 1.1.1 Communicate via legacy protocol on a sequential data channel
- DP 1.1.1 Legacy protocol interpreter (built to match spec)
- FR 1.1.2 Initialize when triggered
- DP 1.1.2 Legacy interface initializer
- FR 1.1.3 Provide interface to legacy driver
- DP 1.1.3 Legacy Driver / GUI Portal

SubSystem I/F

- FR 1.2.4 Communicate via subsystem I/F protocol
- DP 1.2.4 Subsystem protocol interpreter (built to match spec)
- FR 1.2.5 Initialize when triggered
- DP 1.2.5 Subsystem interface initializer

Manager

- FR 1.3.6 Input adapter configuration
- DP 1.3.6 Configuration reader
- FR 1.3.7 Manage/coordinate initialization sequence
- DP 1.3.7 Adapter Initializer
- FR 1.3.8 Sequence/coordinate fault recovery/response
- DP 1.3.8 Fault Handler
- FR 1.3.9 Collect system state
- DP 1.3.9 Status Collector

Marshaller

- FR 1.4.10 Manage parallel data channels. Assign work to appropriate channel based on configuration and the previous work handled
- DP 1.4.10 Data Dispatcher Channel Manager
- FR 1.4.11 Store message data while waiting status from subsystem
- DP 1.4.11 Waiting-Response-Queue
- FR 1.4.12 Reject unknown transmission
- DP 1.4.12 Data Dispatcher Xmit Handler

Table 7: Final Level 3 Design Matrix

Level 3	1	2	3	4	5	6	9	11	14	15	16	8	12	13	10	7	17
1	X																
2		X															
3			X														
4				X													
5					X												
6						X											
9							X										
11								X									
14									X								
15										X	X						
16										X	X						
8		X			X							X					
10													X	X	X		
12													X	X	X		
13													X	X	X		
7		X			X									X		X	
17									X	X	X						X

- FR 1.4.13 Reject transmissions when triggered
- DP 1.4.13 Data Dispatcher Xmit Handler

GUI

- FR 2.2.14 GUI interface for legacy driver
- DP 2.2.14 GUI Legacy System Application
- FR 2.2.15 Display status of interface states/configuration
- DP 2.2.15 GUI Application*
- FR 2.2.16 Perform diagnostics
- DP 2.2.16 GUI Application*
- FR 2.2.17 Authenticate GUI user
- DP 2.2.17 User Authentication

* These are defined further in the next section.

The Design Matrix shown in Table 7 represents the new level 3 version after altering the position of the rows and columns to get it into lower triangular form. Note that for readability, the following table has DPs as columns and FRs as rows and since the third level number of the FRs and DPs is unique, only the third level number is used. That is, DP 1.3.9 is listed as column 9 and FR 1.3.9 is listed as row 9.

The tight coupling revealed in this iteration is isolated to two respective parts of system, the Dispatcher and the GUI. The coupling in the Dispatcher's functionality (FRs 10, 12, 13) is expected and understandable since all data routing is done at this point. These FRs may be collapsed into a single FR. The breakout of the fault handling after the first level of FRs/DPs enabled helps to determine where that functionality is best served. The two fault handling functions (FRs 12 and 13) may be collapsed together.

The coupling with the GUI in FRs 15-16 is expected since all FRs are served from a single GUI. Collapsing these FRs into one removes coupling in the diagram, however the best solution is to modify the two respective DPs to clarify which function is for system status and which one is for diagnostic access.

The Adapter Managers FRs (7, 8) that require configuration and initialization are coupled with other components. This coupling should not be intrusive to the overall design of the system. All level 3 FR/DP pairs define the scope of a design that may now begin to transition to functional physical components in a logical software design.

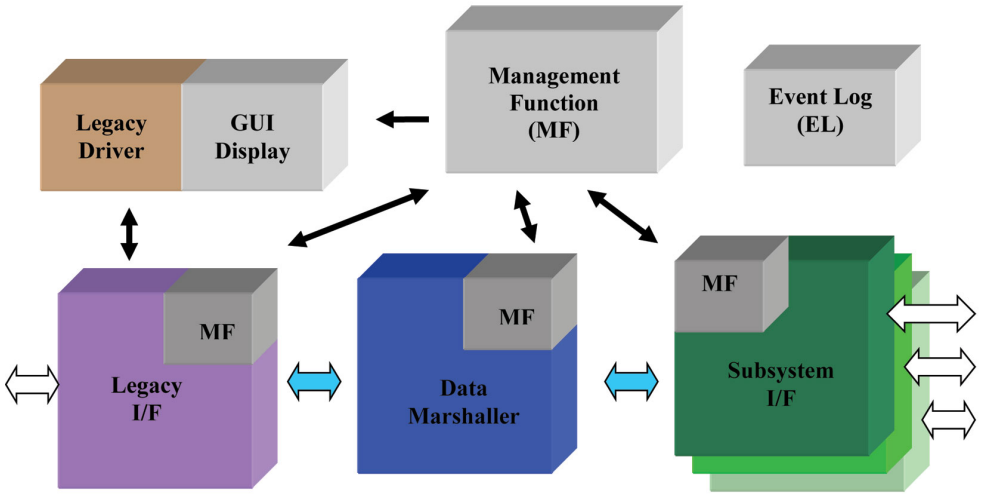

Figure 6: Major design component diagram.

3.5 Resulting Module Definition

Figure 6 represents major components that emerged from the FR/DP and Design Matrix analysis. The level 2 DPs drive the primary functional components in the design. The level 3 DPs reveal more of the functions within and the relationships between the level 2 DPs. The configuration and initialization requirements are mapped to three other components as well as the Adaptor Manager as indicated by the Management Function (MF) subcomponents in the diagram.

3.5.1 Adapter Components

The association of major adapter components identified in Figure 6 may now be associated with more specific mature domains. This consists of using COTS products, middleware, and software languages and libraries as components of the adapter design.

- Management Function
  - o Java package
  - o XML (config file)
- Legacy Interface
  - o Java package
  - o TCP-IP
- Data Marshaller
  - o Java package (algorithms)
- Subsystem Interface
  - o Web Services
- Event Log
  - o Java Logging
- Legacy Driver
  - o Database to store data for test
- GUI Display
  - o Java Swing for GUIs

3.5.2 Adapter Simulations

Simulations may be used to examine the relationships of components and begin to expose the design to the requirements and ensure completeness. The following scenarios are examined using collaboration diagrams to analyze the components and look for functionality completeness.

These scenarios were developed using the steps required for three threads of processing. These threads represent some basic processing paths. The "authentication" component was considered a missing component and is highlighted for discussion later. Figure‘7 shows the collaboration diagram of the IASD initialization process.

All three collaboration diagrams (Figures 7-9) highlight the newly added "Authentication" component to show the relationship of the Authenticator in the respective system processes.

3.5.3 Missing Components (Identification)

The components' relationships are defined as publishers (P) and/or corresponding subscribers (S) in

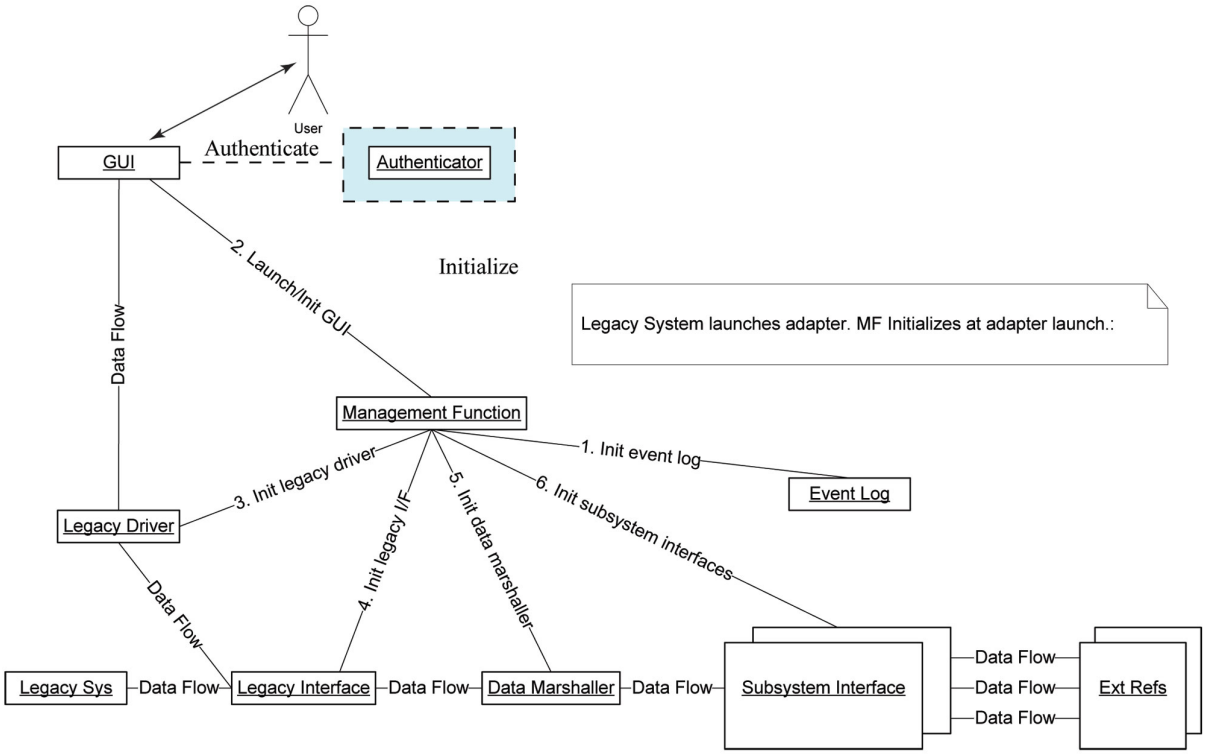

Figure 7: Collaboration diagram (initialization).

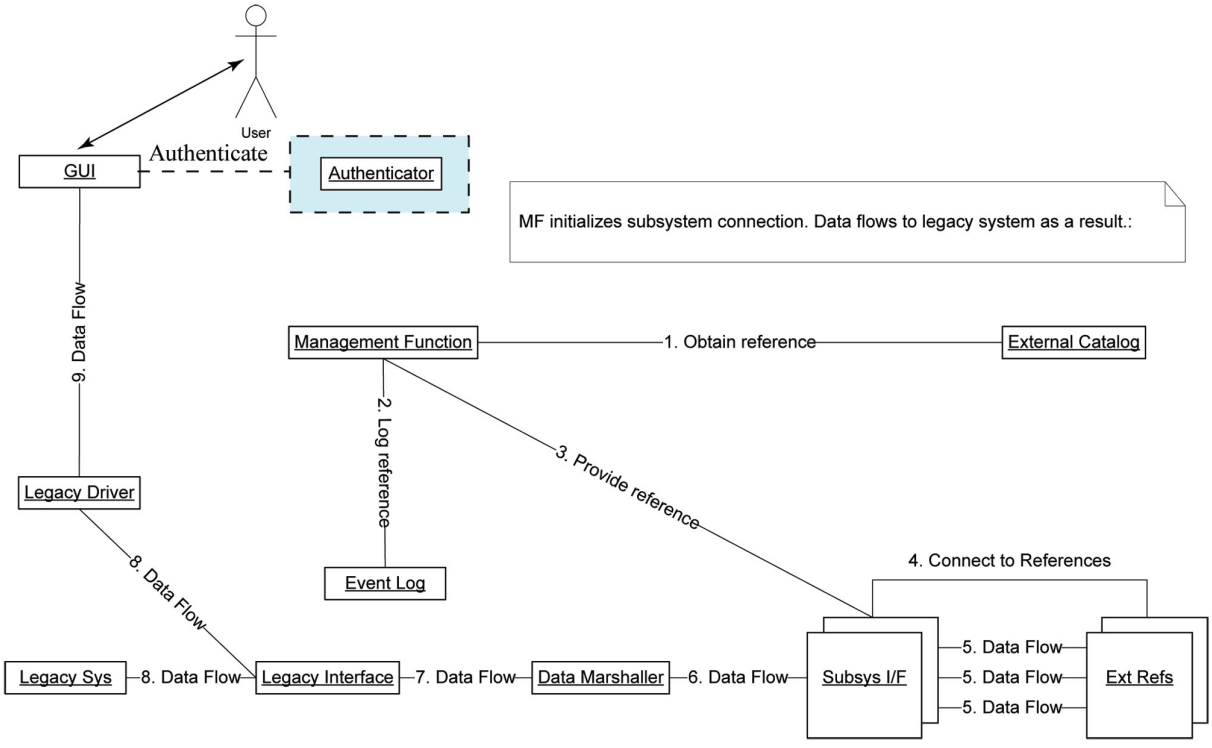

Figure 8: Collaboration diagram (Legacy System Data Process).

Figure 6. The objects are shown as the ordered DPs from the final level design matrix shown in Table 8.

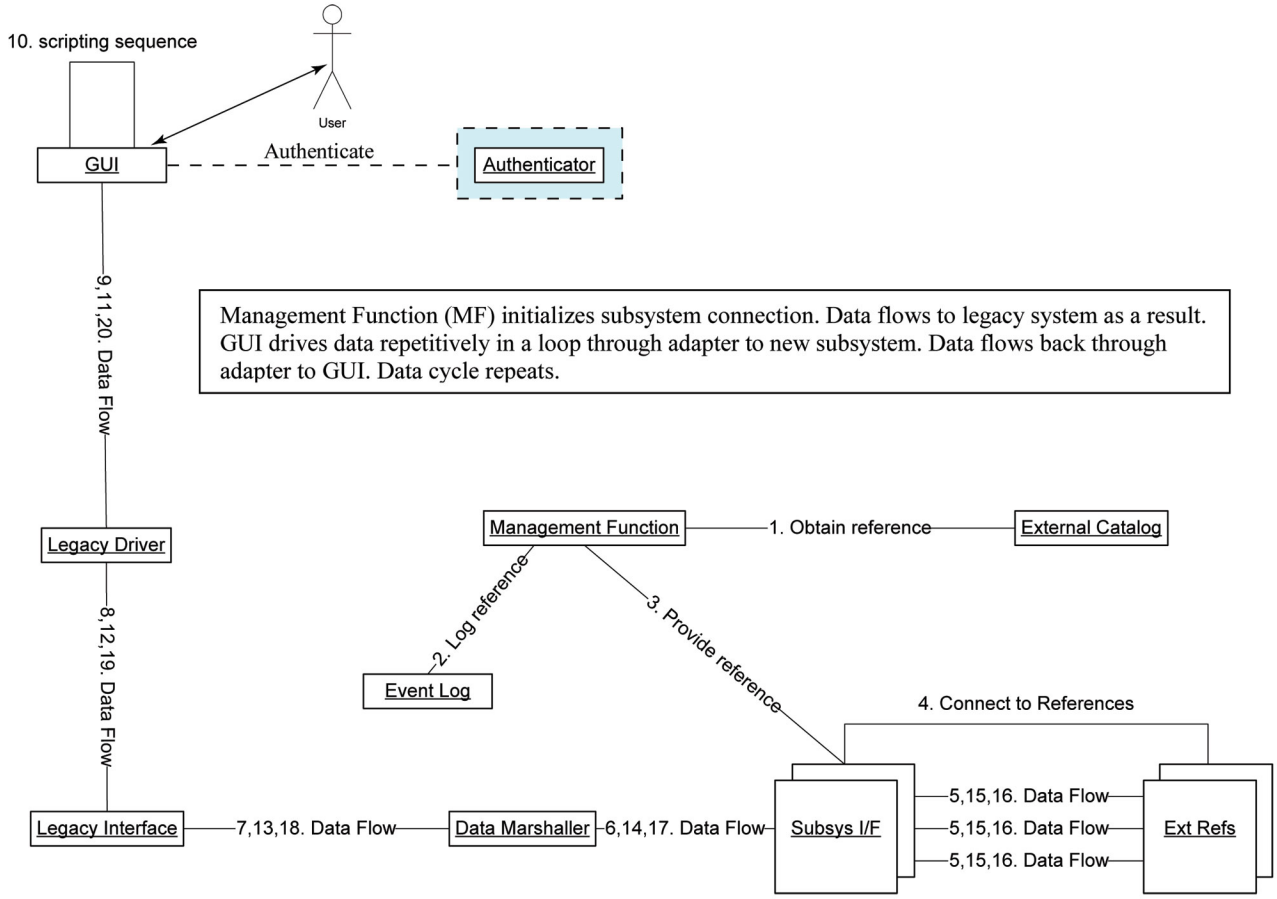

Figure 9: Collaboration diagram (Legacy Driver Guided Processing).

The "Emergent" column captures the DPs that do not have a corresponding component to publish or subscribe to.

At this point, the publish-subscribe condition is broken and one emergent component is exposed. The component needed is an authenticator to validate the user access to the GUI. The missing Authenticator component is added and so the publish-subscribe conditions are satisfied.

3.5.4 Missing Components (Developed)

The missing component provides authentication of users accessing the system via the GUI to operate the legacy data driver and access diagnostic data. The authentication component is attached to the GUI since users will be required to have user ids and passwords entered for authentication. The missing component, "Authentication" is added to the component diagram (Figure 6) and is shown in Figure 10.

The authentication component is integrated into the system and the new functionality as shown in the sequence diagram represented in Figure 11. Since authentication involves validating user access and provides different access rules for testing, administrative, and trouble shooting, it will provide various levels of accessibility. These requirements can be covered using Lightweight Directory Access Protocol (LDAP). This protocol is a mature domain in itself and can be implemented by many different COTS products, typically web servers.

3.5.5 Integration

Integration includes the development of components within the design. The UML diagrams show in Figure 12 and Figure 13 the integration of the java classes that came from the design. These diagrams represent a start to the primary classes, operations, and attributes that will exist in the software system. The integration of these classes to domains like TCP/IP and Database are shown in the UML diagrams.

At this point the test cases may be used to check

Table 8: Checking Integrated Components

Objects	Components						
	Legacy I/F	GUI	Mgmt Function	Legacy Driver	Data Marshaller	Subsystem I/F	Emergent
DP 1.1.1: Legacy Protocol interpreter	S				P		
DP 1.1.2: Legacy Interface initializer	S		P				
DP 1.1.3: Legacy Driver	S	P					
DP 1.2.4: Subsystem protocol interpreter					P	S	
DP 1.3.6: Configuration reader		P	S				
DP 1.3.9: Status collector		S	P				
DP 1.4.11: Waiting – response - queue					S	P	
DP 2.2.14: GUI Legacy system application		S		P			
DP 2.2.15: GUI Application		S	P				
DP 2.2.16: GUI Application		S	P				
DP 2.2.17: User Authentication		S					P
DP 1.3.8: Fault Handler		P	S				
DP 1.4.10: Data Dispatcher Channel Manager			S		P		
DP 1.4.12: Data Dispatcher Xmit Handler			S		P		
DP 1.4.13: Data Dispatcher Xmit Handler			S		P		
DP 1.3.7: Adapter initializer			P		S		

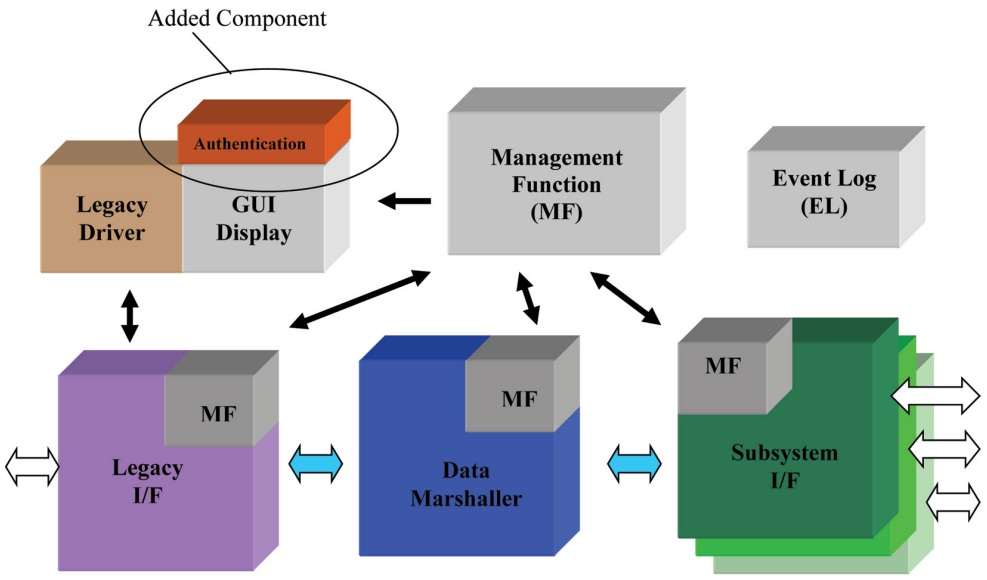

Figure 10: Major design component diagram (Missing Component Added).

out functionality through the integration process. Low level unit tests may be developed to check out component functionality and interfaces. The functional requirements test cases validate overall system functionality as the system is further integrated.

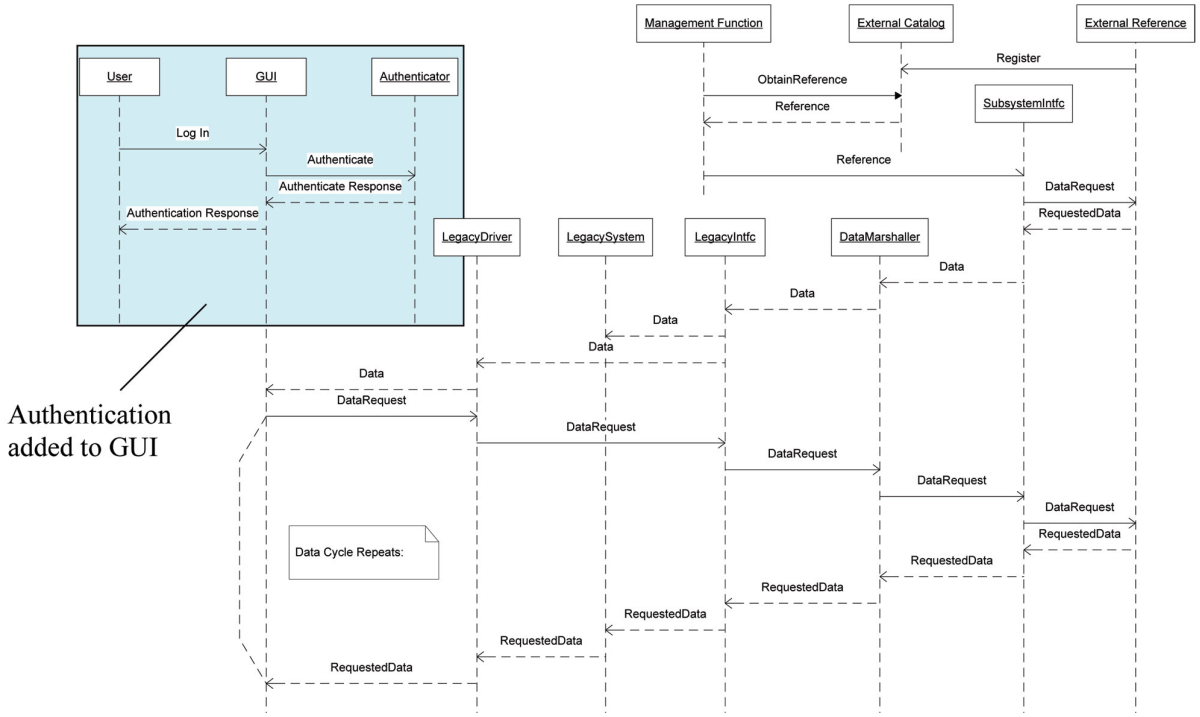

Figure 11: Sequence diagram with user Authentication.

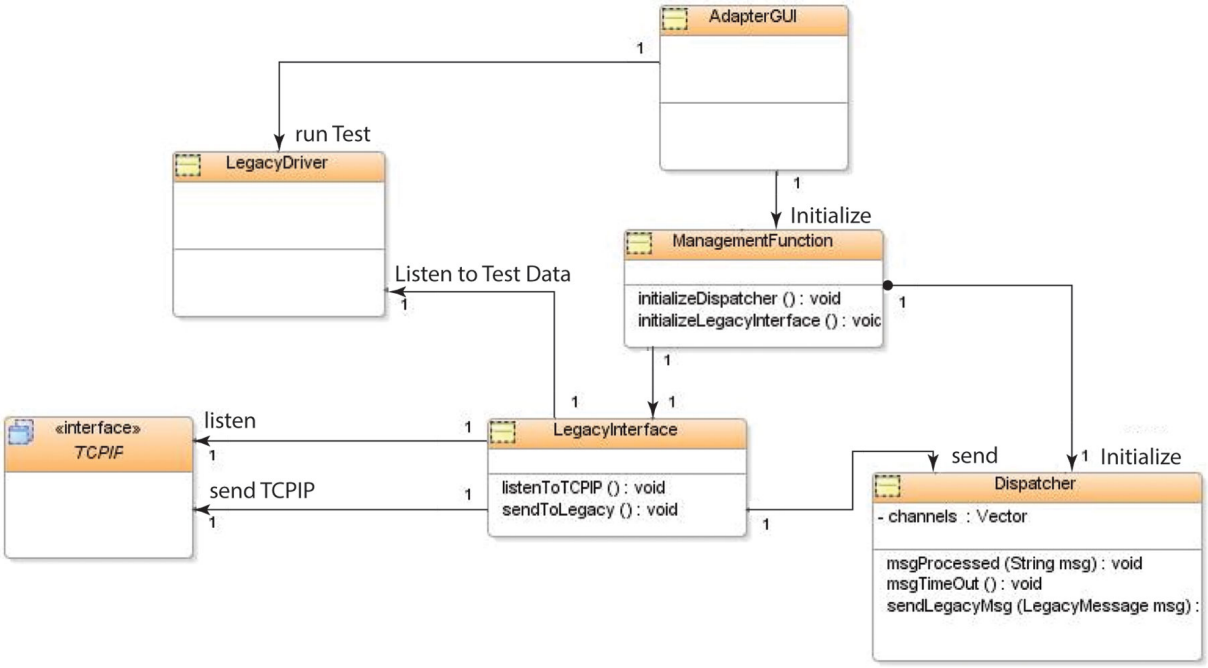

Figure 12: UML diagram (Legacy and GUI).

3.5.6 Adding Components to Mature Domain

The missing authentication component that provides system security is common to both software systems operations and testing functions. The selection of LDAP for this design is a potential component that may be added to the list of mature domains originally identified in Figure 4.

Figure 14 shows an update to Figure 4 (Mature Domains) with security added to the mature domain

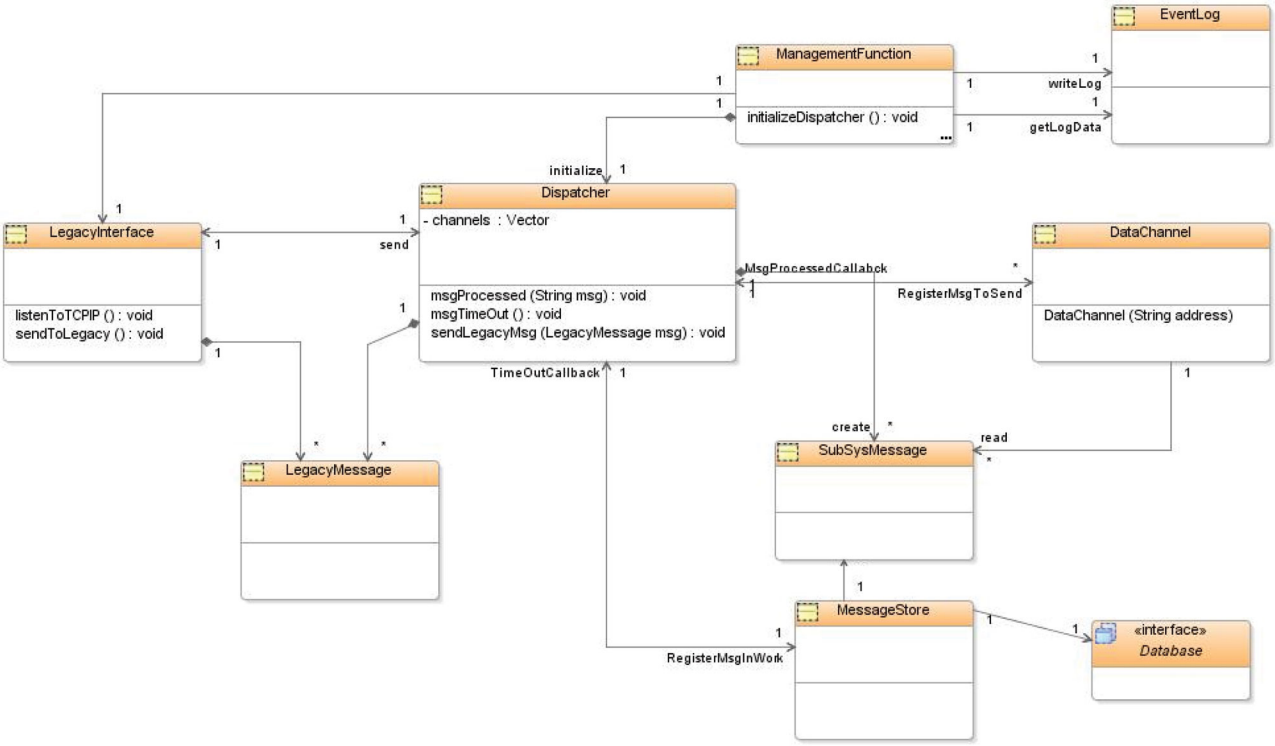

Figure 13: UML diagram (Dispatcher and Subsystem I/F).

components for the missing component, "Authentication".

3.5.7 Software Product (Execution)

The system execution phase of an application such as this Adapter begins when major component development is complete and full system testing may begin with the external interfaces. An application of this type may be deployed when all functional requirement test cases have passed test validation. Customer acceptance should come upon approval of all test results.

4 Axiomatic Design Concluding Comments

The Component-Oriented Axiomatic Design process provides a clear process to identifying design components. This paper, Systematic Component-Oriented Development with Axiomatic Design, [Togay, Dogru, Tanik, Tate] presented the concept with the COAD process of discovering missing components in the design as seen by the missing user authentication component in the Adapter design.

By utilizing the COAD process, perceived customer confidence is heightened because the results of the final design meet the customer needs. The process brings a focus on the important functions of a system and how multiple functions best relate to each other. With the design process steps unique to COAD, the designer may identify missing components that are required to meet defined functional needs without altering the design dramatically.

Bibliography

1. T. Kollman, G. Norby, 2001. Systems Engineering Principles. Tom Kollman, Raytheon Systems Engineering, Greg Norby, Raytheon Systems Engineering, (Presentation Slides for Texas Tech ENG 5000 class, December, 2001).
2. D. Tate, 2006. Fundamentals of Transdisciplinary Design and Process. ATLAS Modul Publication, November 9-11, 2006,
3. Brown, 2006. Elements of Axiomatic Design, a simple and practical approach to engineering design, Draft: 29 March 2006, Christopher A. Brown, Cazenovia, NY 2006.
4. Togay, Dogru, Tanik, Tate, 2006. Systematic Component-Oriented Development with Axiomatic Design,
5. Szyperski, Gruntz, Murer, 2005. Szyperski, C., Gruntz,

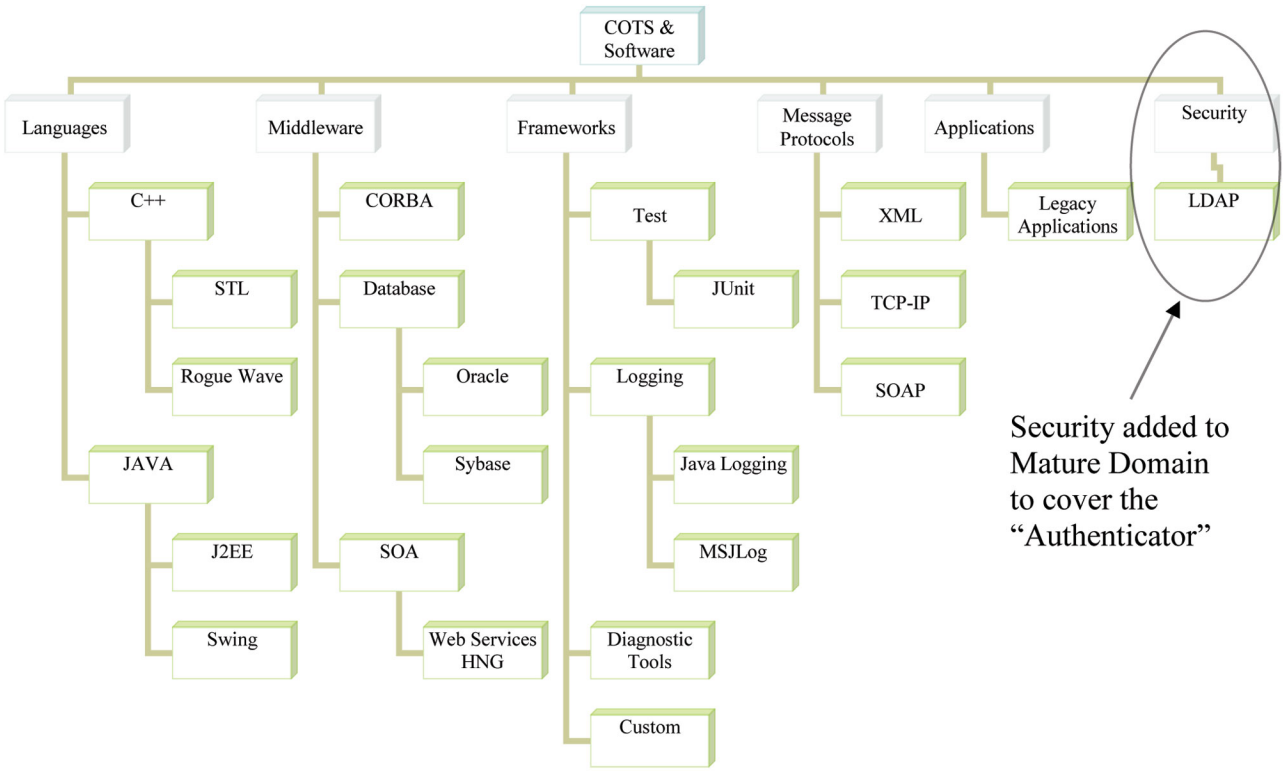

Figure 14: Revised mature domain with security.

D., Murer, S. (2005) Component Software Beyond Object-Oriented Programming. Addison-Wesley, Reading, MA.

6. A. Dogru, 2006. Component Oriented Software Engineering. The ATLAS Publishing, Lubbock, TX.
7. Raytheon IMP_ IMS, 2006. Raytheon Integrated Master Plan and Schedule (IMP-IMS) Guide for Development and Use, Revision 3.0, 30 April 2006
8. Fox, Kirsch, Head, 2006. Axiomatic Design Project Submittal to Dr. D. Tate, December, 2006.

Mr. Lonney Head has BS from University of Colorado in Electrical Engineering, has MS from University of Colorado in Engineering Management and MS from Texas Tech University in Transdisciplinary Enginnering. Currently he is Raytheon Project Engineer and PhDc student at Texas Tech University, has twenty five years of professional industry experience in program management, systems engineering, manufacturing engineering, hardware and software development, integration, and testing. Lonneys expertise includes certifications in Program Management, Six Sigma and Earned Value Management. Throughout his career, Lonney has successfully executed technical programs with multiple United States and foreign governments to include US Air Force, US Navy, NASA, United Kingdom, Kingdom of Saudi Arabia and the Hashemite Kingdom of Jordan.