ARCHITECTURE DOCUMENTATION:

Internet-based Collaboration System

|  |  |  |
| --- | --- | --- |
| CLASS  SWENG 587-002 NEW | REVISION | INITIAL RELEASE DATE  06/19/2019 |
|  |  |  |

PREPARED BY

Group 1

CHANGE HISTORY

|  |  |  |
| --- | --- | --- |
| **REVISION** | **DESCRIPTION** | **DATE** |
| NEW | Initial Release of Final Project | 06/19/2019 |

TABLE OF CONTENTS

[1 Problem Statement 6](#_Toc47742)

[1.1 Assignment Goals](#_Toc47744)

[2 System Overview 7](#_Toc47745)

[2.1 First Iteration: Monolithic System Design (Whole System) 7](#_Toc47746)

[2.2 Second Iteration: Low Latency Communication and Scaling (System Service Controller) 8](#_Toc47747)

[2.2.1 Design and Tactics 10](#_Toc47748)

[2.2.2 Trade Off Analysis](#_Toc47749) 11

[2.3 Third Iteration: Creation of Teams and Team Structure (Data Persistence)](#_Toc47750) 11

[2.3.1 Design and Tactics](#_Toc47751) 14

[2.3.2 Trade Off Analysis 15](#_Toc47746)

[2.4 Fourth Iteration: User Input and Output Devices (User and Device Interface) 15](#_Toc47747)

[2.4.1 Design and Tactics 18](#_Toc47748)

[2.4.2 Trade Off Analysis](#_Toc47749) 18

[2.5 Fifth Iteration: Authentication and Authorization (Security)](#_Toc47750) 18

[2.5.1 Design and Tactics 20](#_Toc47751)

[2.5.2 Trade Off Analysis 21](#_Toc47749)

[2.6 Sixth Iteration: Monitoring and Recovery (Monitoring and Recovery) 21](#_Toc47747)

[2.6.1 Design and Tactics 24](#_Toc47748)

[2.6.2 Trade Off Analysis](#_Toc47749) 24

[2.7 Seventh Iteration: Global Support (Localization) 24](#_Toc47750)

[2.7.1 Design and Tactics 27](#_Toc47751)

[2.7.2 Trade Off Analysis 27](#_Toc47749)

[3 Overall System Trade Off Analysis 27](#_Toc47745)

[4 Proposed Architecture 28](#_Toc47745)

TABLE OF FIGURES

Figure 1: Target Architectural Drivers for ADD methodology creating the Internet-Based Collaboration System ........................................................................................................................ 6

Figure 2: Monolithic System Diagram for the Internet-Based Collaboration System...................................................................................................................... 8

Figure 3: Mapping of Architectural Drivers to Monolithic Modules.................................................................................................................. 8

Figure 4: System Decomposition Diagram of the System Service Controller and Arch. Driver #1....................................................................................................... 9

Figure 5: Mapping of Functional Responsibility and Tactics to Second Iteration Modules. ......................................................................... 9

Figure 6: Component Connector Diagram for Second Iteration Modules......................................................... 10

Figure 7: System Decomposition Diagram of the Data Persistence Module and Arch. Driver #2................................................................ 12

Figure 8: Mapping of Functional Responsibility and Tactics to Third Iteration Modules.......................................................... 13

Figure 9: Component Connector Diagram for Third Iteration Modules.............................................. 14

Figure 10: System Decomposition Diagram of the User and Device Interface Module and Arch. Driver #4.................................................... 16

Figure 11: Mapping of Functional Responsibility and Tactics to Fourth Iteration Modules........................................................................ 16

Figure 12: Component Connector Diagram for Fourth Iteration Modules....................................................................................................................... 17

Figure 13: System Decomposition Diagram of the Security Module and Arch. Driver #3................................................... 19

Figure 14: Mapping of Functional Responsibility and Tactics to Fifth Iteration Modules.................................................... 19

Figure 15: Figure 15 - Component Connector Diagram for Fifth Iteration Modules........................................................................ 20

Figure 16: Figure 16 - System Decomposition Diagram of the Monitoring and Recovery Module and Arch. Driver #5....................................................................................................................... 22

Figure 17: Figure 17 - Mapping of Functional Responsibility and Tactics to Sixth Iteration Modules................................................................. 22

Figure 18: Figure 18 - Component Connector Diagram for Sixth Iteration Modules........................................................................ 23

Figure 19: Figure 19 - System Decomposition Diagram of the Localization Module and Arch. Driver #6................................................................. 25

Figure 20: Mapping of Functional Responsibility and Tactics to Seventh Iteration Modules....................................................................................................................... 25

Figure 21: Component Connector Diagram for Seventh Iteration Modules................................................................. 26

Figure 22: System Decomposition Diagram of the Internet-Based Collaboration System........................................................................ 28

Figure 23: Mapping of Functional Responsibility and Tactics for the Internet-Based Collaboration System....................................................................................................................... 29

Figure 24: Component Connector Diagram for the Internet-Based Collaboration System................................................................. 32

# Problem Statement

“You are to develop an Internet-based collaboration system that incorporates speech communication, video conferencing, email, file sharing, and a shared whiteboard that collaborators (such as software engineers) on a virtual team (such as a globally distributed software development project) can use for brainstorming and sketching their ideas (for instance, drawing UML diagrams for software design).

“The system should allow collaborators to use a variety of input devices such as a wireless pen tablet or touch screen and should be capable of sending output to different devices such as a display station or a smart phone. It should be highly responsive for critical operations such as sketching or drawing, speech and video conferencing. The system should recover from unstable network connections, and it should be possible to track progress of execution and troubleshoot any failures at runtime.”

## Assignment Goals

In this assignment, we are asked to use the recursive decomposition technique described by the Attribute Driven Design (ADD) methodology. Through multiple iterations, an architecture addressing the above problem statement will be formed. Using the information generated as part of the previous Project Checkpoints (including Engineering Objectives, Quality Attributes and Scenarios, Tactics, Use Cases, Functional Requirements, among other details), a design fulfilling the Architectural Drivers created in Project Checkpoint #3 should be generated Each section following will present a cycle of the recursive decomposition targeting a specific Architectural Driver, beginning with the creation of a monolithic system design and continuing from there. If the final outcome is desired, please see Section 11 (beginning on page 21). For easier reference, we have included the target Architectural Drivers created in Project Checkpoint #3 in Figure 1, below:

*Figure 1 - Target Architectural Drivers for ADD methodology creating the Internet-Based Collaboration System*

|  |  |  |  |
| --- | --- | --- | --- |
| **Rank** | **Architectural Driver** | **Quality Attribute** | **Priority** |
| **1** | Provide low latency interaction for video, voice, and other team communication, capable of scaling on commercially available hardware | Performance | (H , H) |
| **2** | Allow for the creation and modification of teams and team structure by users | Usability | (H , M) |
| **3** | Provide for user authentication and the authorization of user actions | Security | (H , H) |
| **4** | Interface with a variety of current user input and output devices | Modifiability | (M , H) |
| **5** | Monitor and recover from system connection losses and errors | Availability | (M , H) |
| **6** | Provide language localization and global support | Usability | (M , M) |

# System Overview

## First Iteration: Monolithic System Design (Whole System)

As described in the ADD process outline (appearing in the text, included slides, and accompanying materials), the first step we began with was the creation of a monolithic system design; intended to generally outline the separation of responsibilities in the system and provide for elements that can be further decomposed in later iterations. After some discussion, our group settled on a design pattern based on the ***Model – View – Controller*** pattern (MVC). This is a common pattern applied in applications that contain many user-facing elements that also require the system to process user data and persist that data in some way.

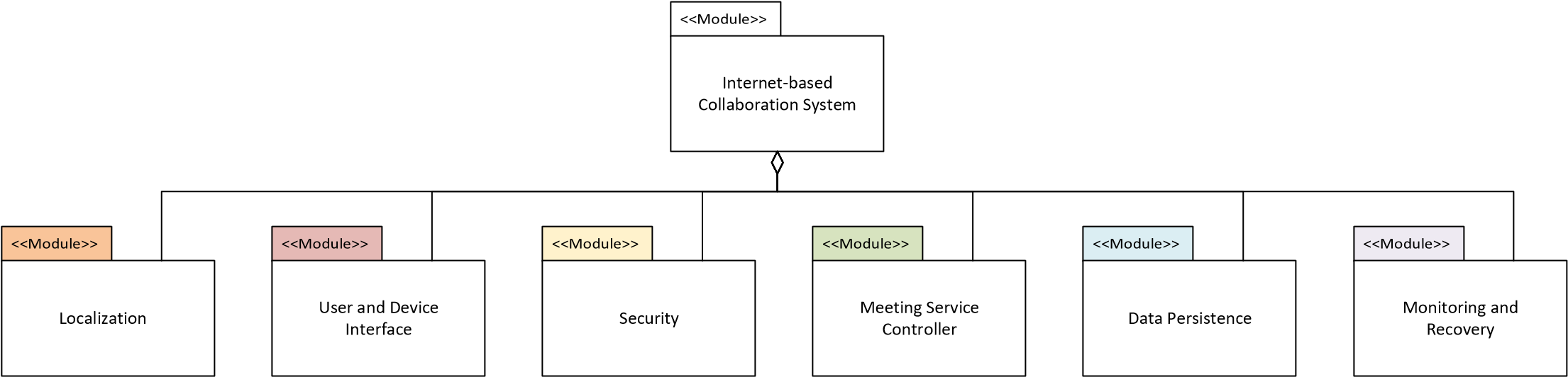
From here, we began to consider the responsibilities that each of these pattern elements would have that are specific to our system. The ***View*** in this pattern should handle the user-facing elements; the presentation of information and the capture of user inputs. Having this element contain the interaction with user devices and the language localization seems to make the most sense (and became the User and Device Interface, and Localization modules respectively). The ***Controller*** in this pattern should handle the processing of the user requests from the View. This element would therefore need to handle the creation of the various meeting services, and ensure the high performance of those services (becoming the System Service Controller). Finally, the ***Model***contains the data persistence of the system. It should be here that the creation and organization of users and teams should take place (and has become the Data Persistence module).

In addition to the traditional MVC pattern, there are two additional elements described by the Architectural

Drivers that are not yet addressed in this iteration. The first is **Security**. By adding an additional element with the responsiblities of authenticating user access as well as authorizing user actions, we can address this driver. The second is **Availability**. With an additional element monitoring meeting services for continued operation, and the ability to restart those services if needed (the monitoring and recovery module), we can also address this driver.

With all the initial considerations above, an initial monolithic system design emerges containing six elements. Two elements dedicated to user interaction (and the handling of user devices) and the localization of that user interaction, an element dedicated to the creation and execution of meeting services, an element dedicated to data storage, an element dedicated to user security, and an element dedicated to system monitoring and recovery. We felt that at the high level of this first iteration, this adequately covers the Architectural Drivers, and allows for a well-encapsulated foundation for further decomposition. A diagram of this design appears in Figure 2, below, and a table showing the mapping between these modules and the applicable architecture in Figure 3.

*Figure 2 - Monolithic System Diagram for the Internet-Based Collaboration System*



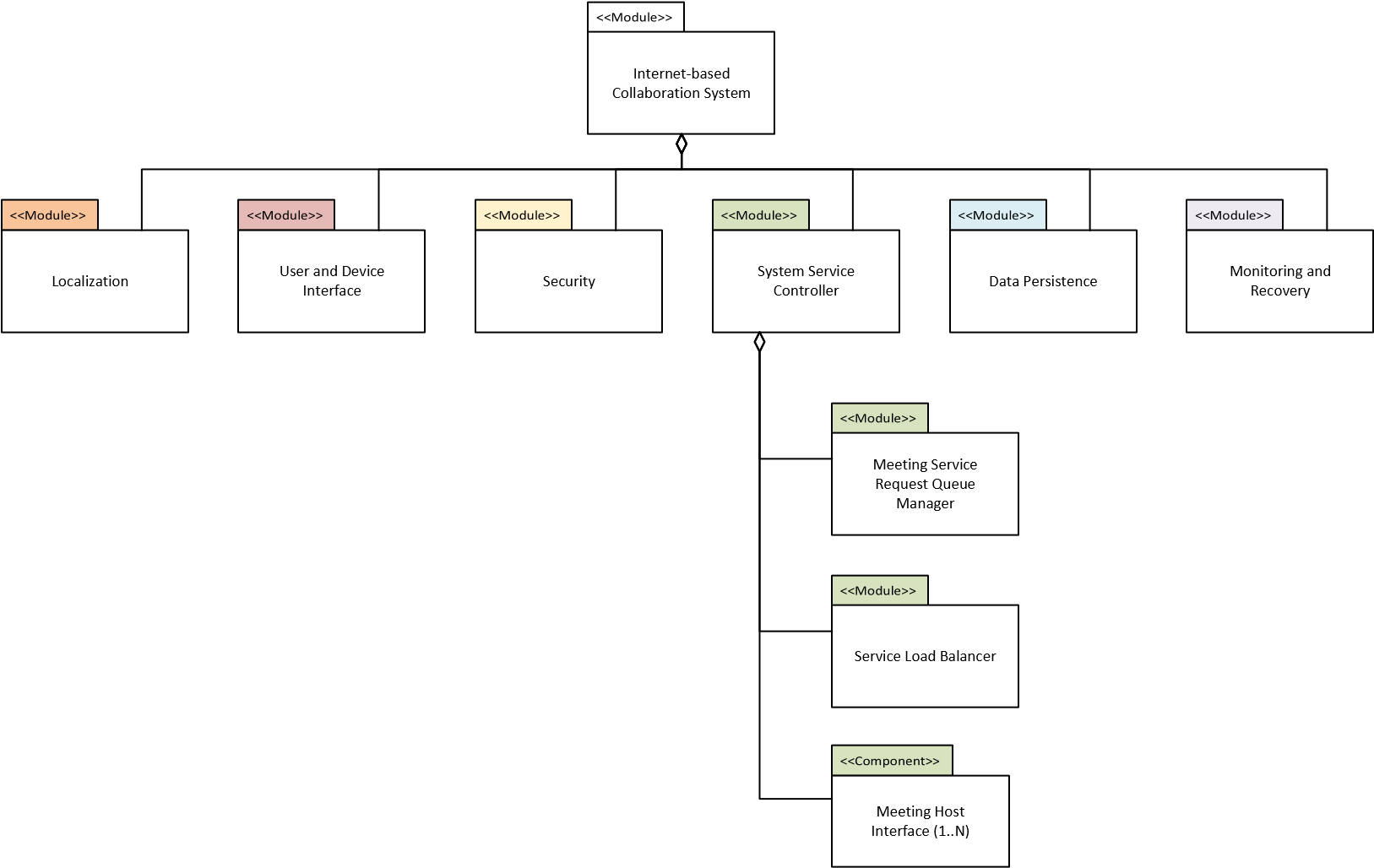
*Figure 3 - Mapping of Architectural Drivers to Monolithic Modules*

|  |  |  |
| --- | --- | --- |
| **Module** | **Architectural Driver** | **Quality Attribute** |
| User and Device Interface | 4. Interface with a variety of current user input ad output devices | Modifiability |
| Security | 3. Provide for user authentication and the authorization of user actions | Security |
| System Service Controller | 1. Provide low latency interaction for video, voice, and other team communication; capable of scaling on commercially available hardware | Performance |
| Data Persistence | 2. Allow for the creation and modification of teams and team structure by users | Usability |
| Monitoring and  Recovery | 5. Monitor and recover from system connection losses and errors | Availability |
| Localization | 6. Provide language localization and global support | Usability |

2.2 Second Iteration: Low Latency Communication & Scaling (System Service Controller)

For this iteration, we selected the **Meeting Service Controller** as the target for decomposition, and “*Provide low latency interaction for video, voice, and other team communication capable of scaling on commercially available hardware*” as the architectural driver. The initial results of this decomposition are shown in Figure 4, below, with a table showing functional responsibilities and a component-connector view following in Figure 5 and Figure 6, respectively. An explanation and analysis of our design follows these figures.

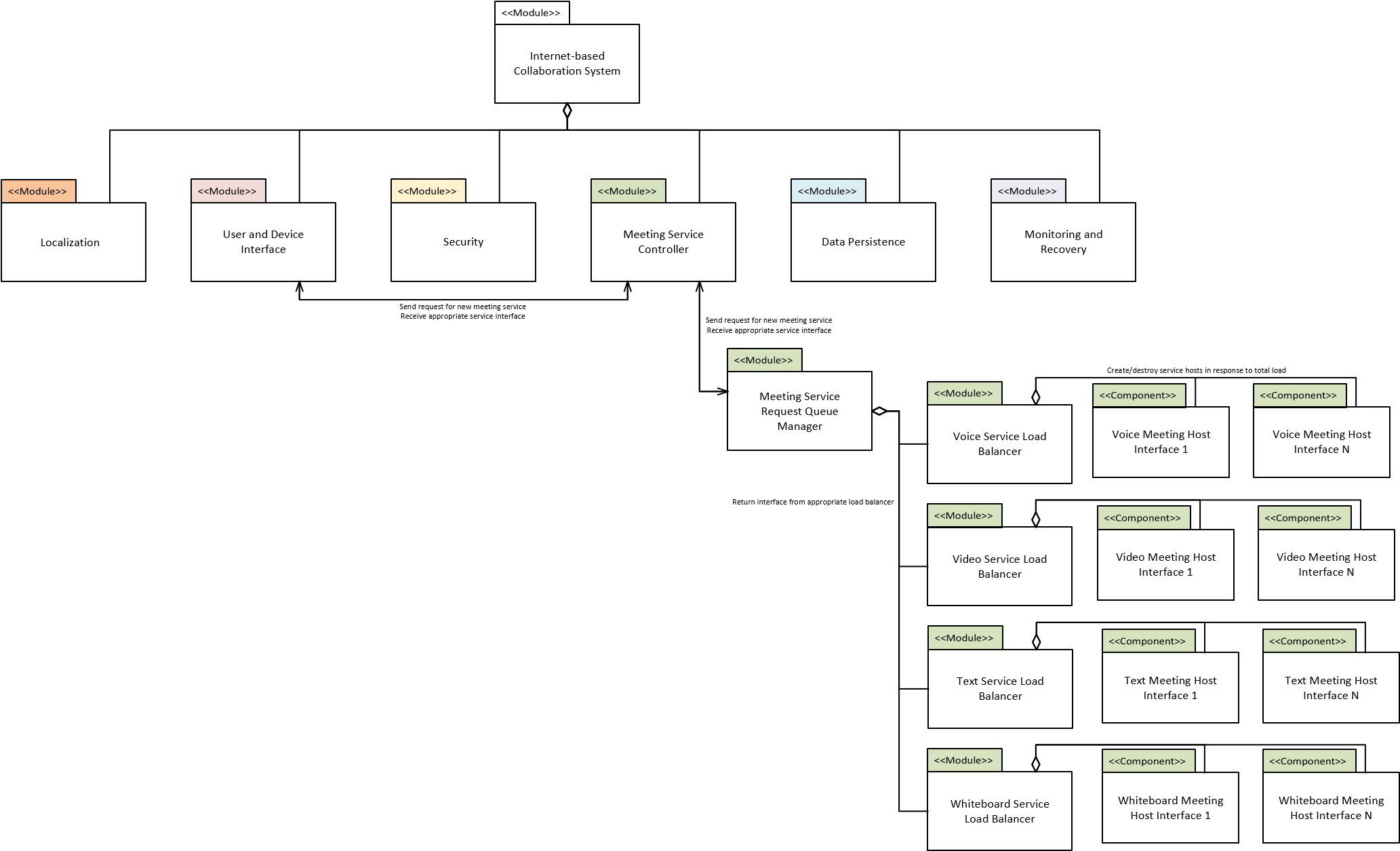
*Figure 4 - System Decomposition Diagram of the System Service Controller and Arch. Driver #1*



*Figure 5 - Mapping of Functional Responsibility and Tactics to Second Iteration Modules*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Sub-component | Architectural Driver | Functional Responsibility | Quality Attribute / Tactics |
| System  Service Controller |  |  | 1. Provide low latency interaction for video, voice, and other team communication; capable of scaling on commercially available hardware | 2. The system shall maintain a responsive UI during user interaction | Performance:  Control Resource  Demand –  Prioritize Events |
| Meeting  Service  Request Queue  Manager | Voice Service Load Balancer | 1a,b,d,e. The system shall provide userexpected communication methods | Performance:  Manage Resources  - Increase  Concurrency,  Increase Resources |
| Voice Meeting Host Interface |
| Video Service Load  Balancer |
| Video Meeting Host  Interface |
| Text Service Load Balancer |
| Text Meeting Host Interface |
| Whiteboard Service Load  Balancer |
| Whiteboard Meeting Host  Interface |

*Figure 6 - Component Connector Diagram for Second Iteration Modules*



# 2.2.1 Design and Tactics

For the design in this decomposition, we have chosen to add a **request queue manager**, as well as individual **service load balancing** in order to provide a solution for the system to be capable of achieving low latency and scalability. New meeting service requests from users (through the User and Device Interface) are processed by the request queue manager first, allowing for the prioritization of low latency services (such as voice and video) over requests where delays would not be noticeable by a user (such as text or email). This design choice is based on the Performance tactic “*Control Resource Demand – Prioritize Events*”. These requests are then passed to an appropriate load balancer, which can return the interface for a particular instance of the service requested back to the User and Device Interface. The load balancers can determine if the new request would overwhelm the current service host and instead create a new host instance for that request. Conversely, they could also determine that the number of current hosts may be too large for the given number of requests, consolidating and destroying extraneous hosts. Separating the load balancers allows for services with differing processing requirements to be managed optimally (i.e. many more text service requests could be handled by one host than video service requests). This design choice is based on the Performance tactics “*Manage Resources – Increase Concurrency*” and “*Manage Resources – Increase Resources*”.

Ideally, the service load balancers would be abstracted in a final design, such that the request queue manager could handle any arbitrary type of service. This would allow for future communication services to be added with minimal impact on the rest of the design. However, the focus of this iteration was the performance of the system, and not its modularity. It would likely still be worth noting this for potential refactoring in the future.

# 2.2.2 Trade Off Analysis

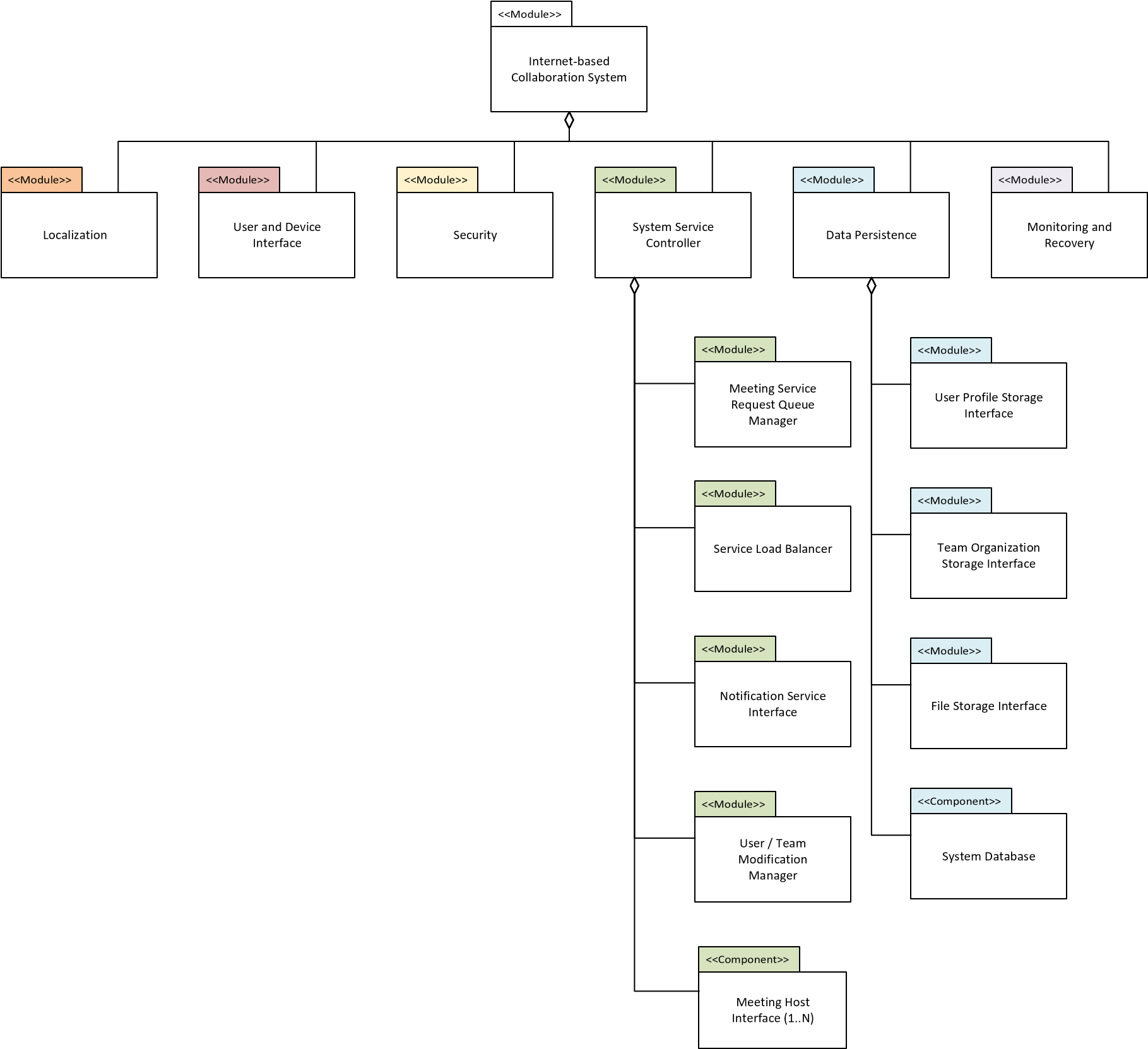
Allowing for the system to expand resources as necessary to meet performance metrics also would lead to a commensurate increase in costs. One of the constraints identified in Project Checkpoint #3 was an overall cost limit. It is possible under this design for the system to expand beyond this limit if left unchecked. A possible way to balance this trade off would be to allow the load balancers to act only to a set upper limit. After this limit has been reached, the hosts themselves could reduce service quality (e.g. video resolution for a video meeting request) in order to remain under hardware limitations.

This design also leads to increased complexity, especially in the case of the system monitoring Architectural Driver. Each new instance created by the load balancers would also need to create a connection to the monitoring and logging service. This in turn creates more network load that would also have to be managed.

## Third Iteration: Creation of Teams and Team Structure (Data Persistence)

In this iteration we have selected the **Data Persistence** module as the target for decomposition, and “*Allow for the creation and modification of teams and team structure by users*” as the architectural driver. The initial results of this decomposition are shown in Figure 7, below, with a table showing functional responsibilities and a component-connector view following in Figure 8 and Figure 9, respectively. An explanation and analysis of our design follows these figures.

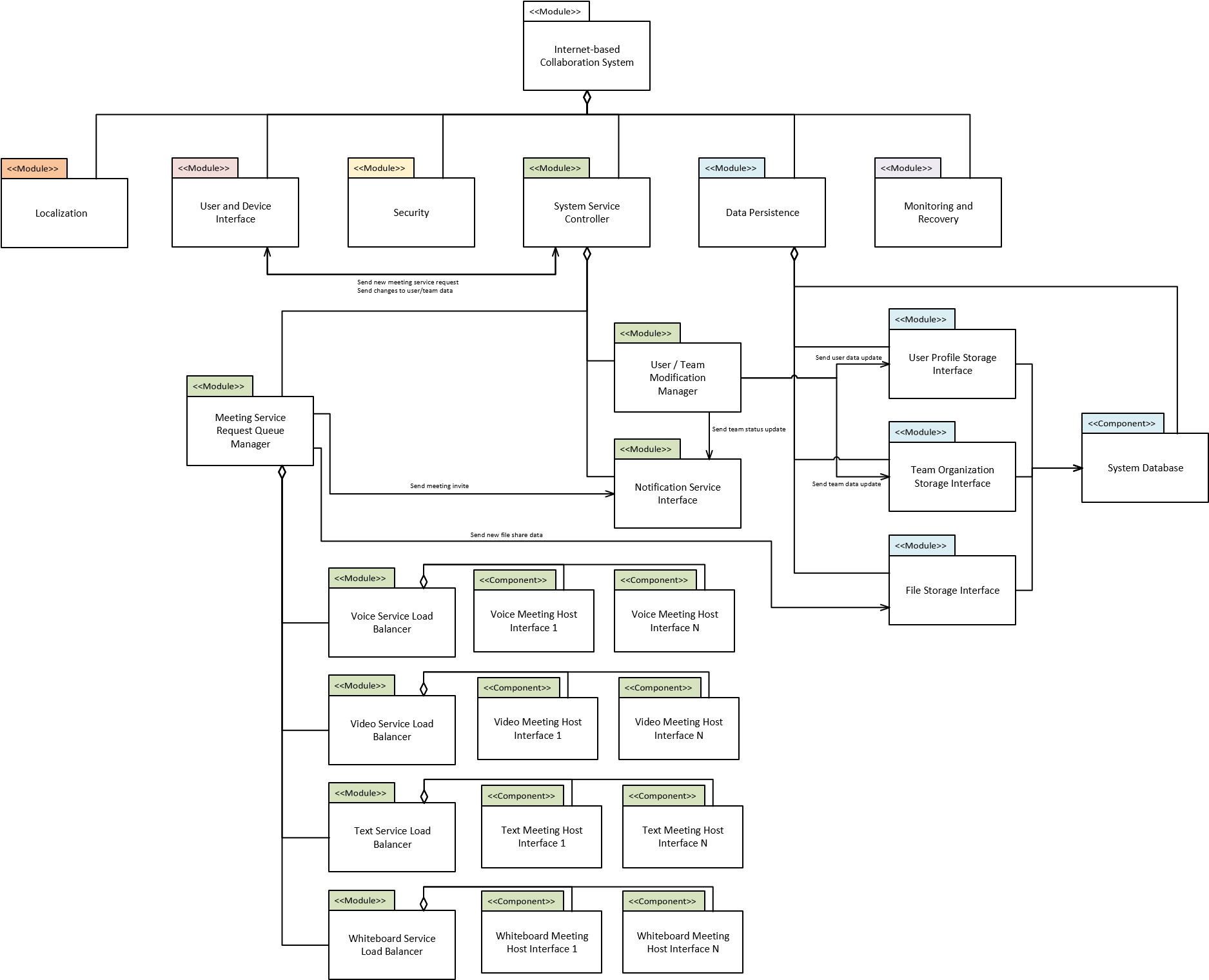
*Figure 7 - System Decomposition Diagram of the Data Persistence Module and Arch. Driver #2*



*Figure 8 - Mapping of Functional Responsibility and Tactics to Third Iteration Modules*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Subcomponent | Architectural Driver | Functional Responsibility | Quality  Attribute /  Tactics |
| System  Service Controller | Notification Service Interface | | 2. Allow for the creation and modification of teams and team structure by users  4. Interface with a variety of current user input and output devices | 1. The system shall allow users to start group meetings with other remote users 2. The system shall allow users to be added to a group meeting | Interoperability:  Manage  Services - Orchestrate |
| User / Team  Modification  Manager | User Profile  Service  Interface | 2. Allow for the creation and modification of teams and team structure by users | 5. The system shall provide users a means to invite new users to the system | Usability:  Support User  Initiative - Aggregate, |
|  |  | Team  Organization  Storage  Interface |  | 1. The system shall provide users a means to create teams 2. The system shall provide users a means to add and remove other users from a team | Support System  Initiative -  Maintain Task  Model |
| Data Persistence | System Database | | 13. The system shall comply with the local laws and regulations of a user | Usability:  Support System  Initiative -  Maintain User Model |
| File Storage Interface | | 1. Provide low latency interaction for video, voice, and other team communication; capable of scaling on commercially available hardware | 1c. The system shall provide user-expected communication methods | Usability:  Support System  Initiative -  Maintain Task  Model |

*Figure 9 - Component Connector Diagram for Third Iteration Modules*



# 2.3.1 Design and Tactics

In this decomposition iteration, we have added the notion of **database and model interface layers** for access to the newly created database, as well as adding two new services for the **aggregation of modifications** to user and team data, as well as for the **notification** of users to changes in team organization or new meeting requests. The System Service Controller now acts as an interface for more than the creation of new meeting services and is instead the center of orchestration for requests through the system, based on the “*Interoperability: Manage Services – Orchestrate*” tactic.

Changes to user or team information (e.g. adding or removing a user from a team) originating from the User Interface are passed from the System Service Controller to a User/Team Modification Manager. This module disaggregates the initial request from the User Interface (which likely contains multiple modifications, such as adding a new user to an existing team) into the appropriate functions in the Storage Interfaces contained in the Data Persistence element. This design is based on the Usability tactic “*Support User Initiative – Aggregate*”. These Storage Interfaces are representational models (ORMs) of the data contained in the System Database, also part of the Data Persistence element. Users and Teams were separated in order to encapsulate associated data in their respective ORM. Maintaining these ORMs also allows for the User Interface (through the Modification Manager Service) to only present options available to a user at the time (e.g. only allowing for a new user to be added to a current user’s teams). This design is based on the Usability tactic “*Support System Initiative – Maintain Task Model*”.

A notification service was also added to the System Service Controller as part of this decomposition, allowing for current users to be updated on changes to their team status, receive new meeting requests, and for new users to be invited to the system. The Notification Service Interface connects with both the User/Team Modification Manager and the Meeting Service Request Queue Manager in order to provide those services with notification ability.

Finally, a File Storage Interface was added to the Data Persistence element. While this new module does not directly support the ASR under analysis for this iteration, it does support the previous ASR by providing one of the user-desired communication methods listed in the functional requirements (file sharing). The Meeting Service Request Queue can access user-stored files in the database through this interface when requested by the User Interface.

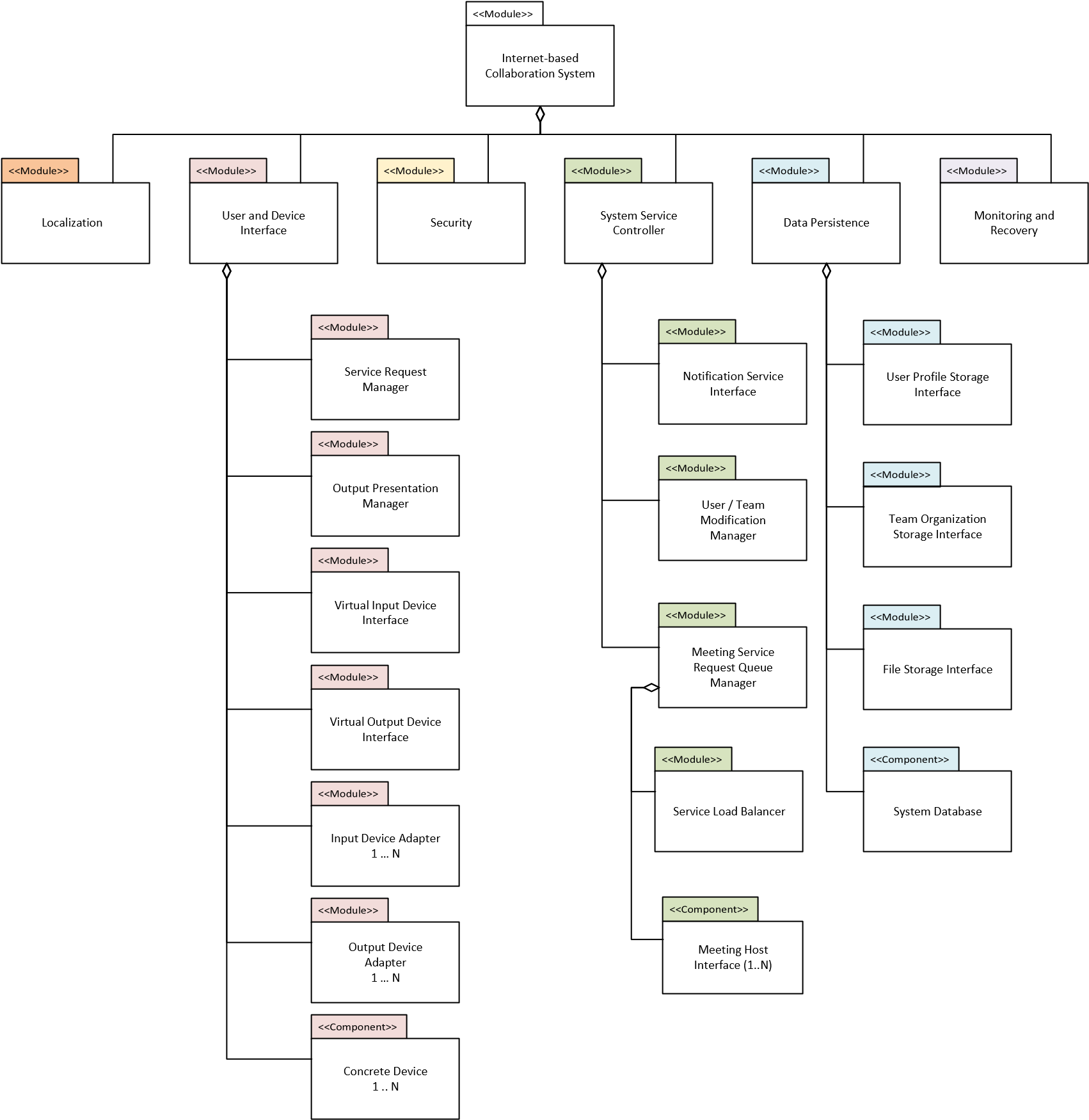
# 2.3.2 Trade Off Analysis

Providing data persistence once again increases the cost of the system, especially for the purposes of user file sharing storage. Containing the Task Model for the system in ORM interfaces for this design also adds complexity in having another interface layer between service interactions with the database. However, these ORMs can provide more flexible access to the database and allows the services to modify database elements in a more abstract way. Another major constraint is the local laws and regulations regarding data storage. Depending on the locale, personal user information may need to be stored in more complicated ways than suggested in this design.

## Fourth Iteration: User Input and Output Devices (User and Device Interface)

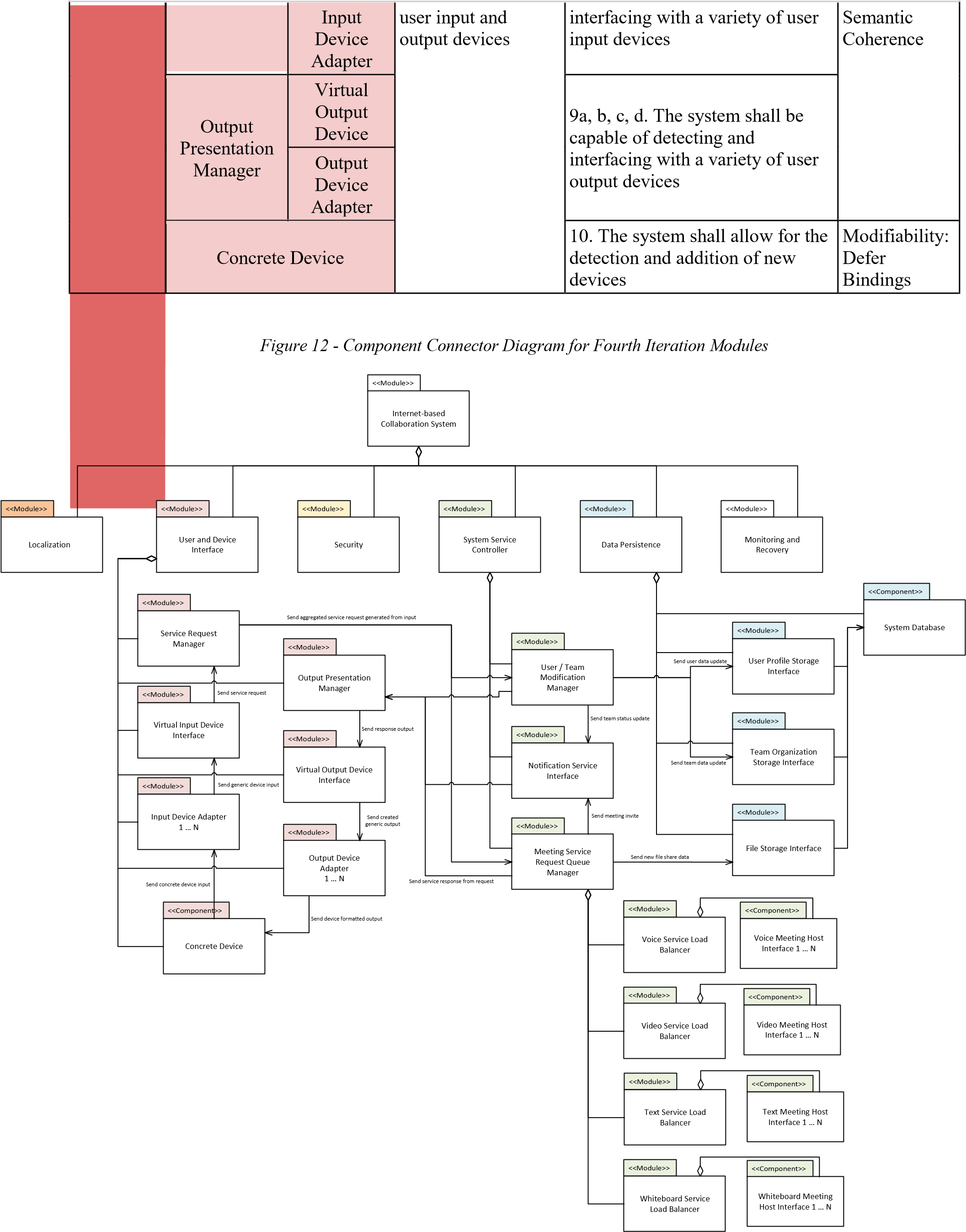
In this iteration, we have selected the **User and Device Interface** element as the target for decomposition, and “*Interface with a variety of current user input and output devices*” as the architectural driver. The initial results of this decomposition are shown in *Figure 10*, below, with a table showing functional responsibilities and a component-connector view following in Figure 11 and Figure 16, respectively. An explanation and analysis of our design follows these figures.

## *Figure 10 – System Decomposition Diagram of the User and Device Interface Module and Arch. Driver #4*



## *Figure 11 - Mapping of Functional Responsibility and Tactics to Fourth Iteration Modules*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Subcomponent | Architectural Driver | Functional Responsibility | Quality  Attribute /  Tactics |
| User and  Device Interface | Service  Request  Manager | Virtual Input  Device | 4. Interface with a variety of current | 8a, b, c, d. The system shall be capable of detecting and | Modifiability:  Encapsulation,  Increase |



# 2.4.1 Design and Tactics

The architectural driver we chose for this decomposition concerns the ability of the system to interface with arbitrary user devices. The ability for developers to quickly add and test new devices to the system as requested by users, without disrupting other system services, is a major design challenge. Therefore, in this iteration much of our design focused on implementing a subset of the Modifiability tactics for the interface with user devices. These tactics were: “*Reduce Coupling – Encapsulate*”, “*Reduce Coupling – Abstract Common Services*”, and “*Defer Bindings*”.

The interface between a user’s device and the rest of the system now includes several steps to provide encapsulation and abstraction of the specific implementation of the device. Instead, most of the system can refer to the devices through **input and output managers**, which encapsulate generic service requests and responses that are agnostic to the type of device they are addressing. These responses are made to **virtual input and output devices**, which serve as abstractions to the underlying real devices such that the input and output managers only process generic events. Finally, **input and output adapters** provide the translation between specific devices and the general interfaces of the virtual devices, where system interaction reaches the user device and is interpreted or displayed as needed.

By creating these layers, the system developers will generally only need to modify input and output adapters as new devices are requested by the user. A new version of a smartphone could use the existing virtual smartphone device interface, since the system would interact with it the same way regardless of the quirks of the specific device. An update to the virtual devices would only be needed when an entirely new type of device interaction is requested (e.g. VR headsets). Even then, if the services required by this device remain the same, the input and output managers could also remain the same.

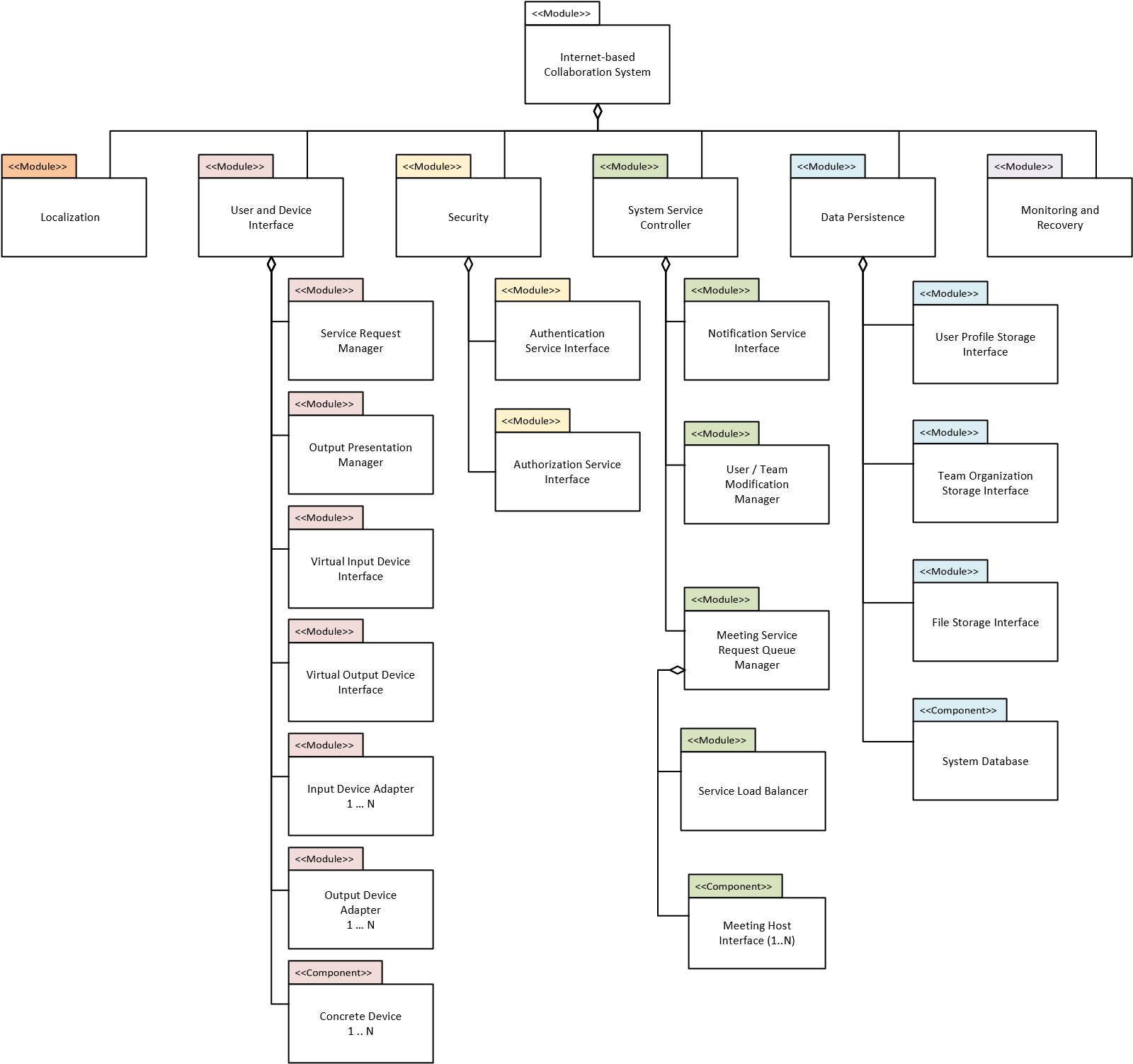
# 2.4.2 Trade Off Analysis

Adding this level of abstraction generates a lot of complexity in the system. Tracking where system failures occur becomes more difficult and will require more logging and monitoring. Errors may bubble up to a level where the logged event is not representative of the real error, making debugging tricky. It also creates an explosion of packages and classes, making it more difficult to find the correct section to modify if a bug is occurring. Multiple interface layers also create more potential for failure. In this design, if any one of the managers, virtual devices, or adapters fails the user device will no longer be reachable.

## Fifth Iteration: Authentication and Authorization (Security)

In this iteration, we have selected the **Security** element as the target for decomposition, and “*Provide for user authentication and the authorization of user actions*” as the architectural driver. The initial results of this decomposition are shown in *Figure 13*, below, with a table showing functional responsibilities and a component-connector view following in Figure 14 and Figure 15, respectively. An explanation and analysis of our design follows these figures.

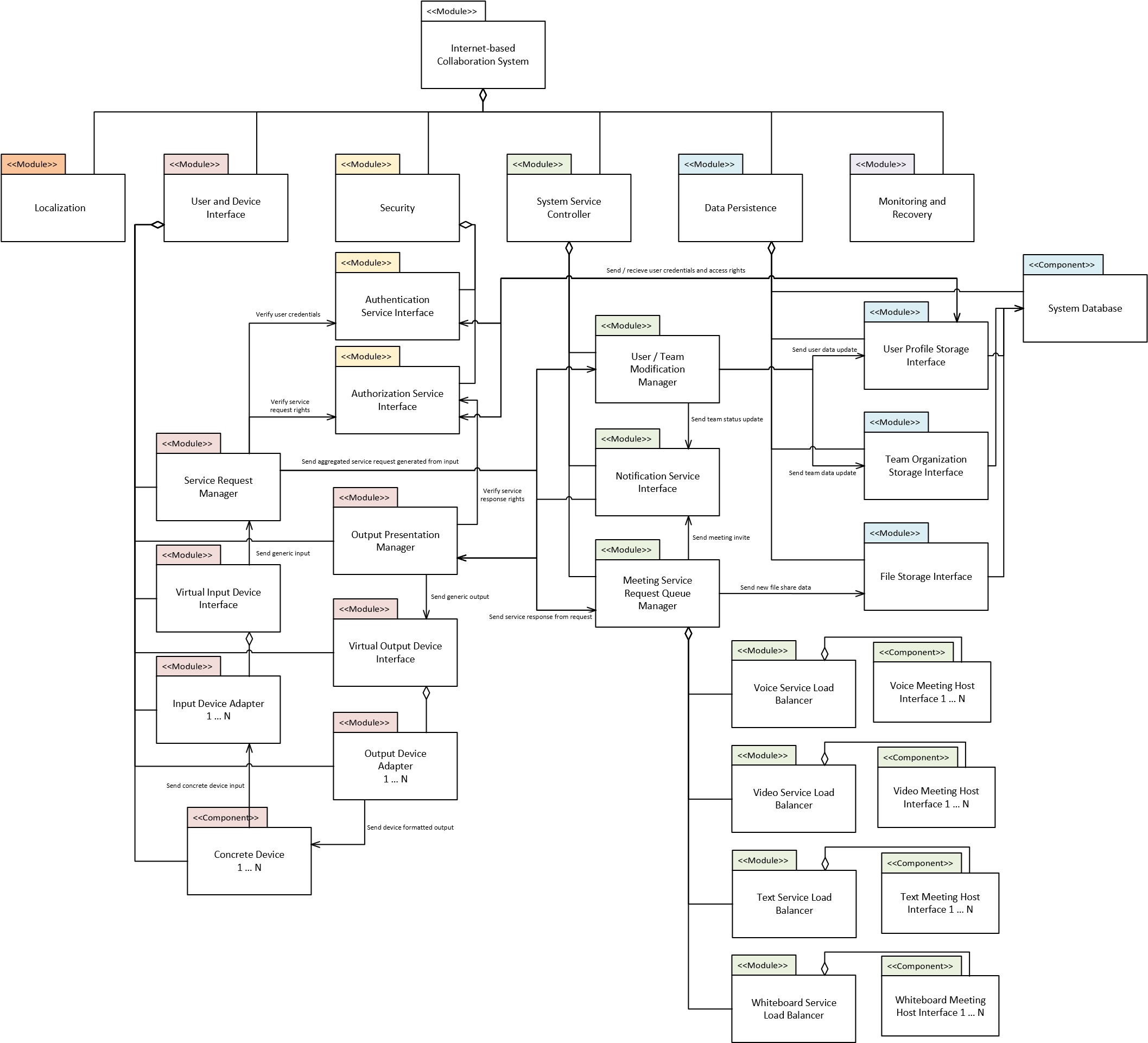
## *Figure 13 – System Decomposition Diagram of the Security Module and Arch. Driver #3*



*Figure 14 - Mapping of Functional Responsibility and Tactics to Fifth Iteration Modules*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Subcomponent | Architectural Driver | Functional Responsibility | Quality Attribute / Tactics |
| Security | Authentication Service Interface | | 3. Provide for user authentication and the authorization of user actions | 14. The system shall provide a means to authenticate user access to the system | Security: Resist  Attacks - Authenticate  Actors, Authorize  Actors, Limit Access |
| Authorization Service Interface | | 15. The system shall provide a means to authorize / deauthorize user actions on the system |

## *Figure 15 - Component Connector Diagram for Fifth Iteration Modules*



# 2.5.1 Design and Tactics

The design and tactics we have used in this iteration are fairly straightforward (perhaps unlike previous iterations). With the input and output managers already in place from the previous iteration, as well as their calls to internal system services, the design tactics we have decided to apply here simply become an intermediate step to system requests and responses. Users making requests to the system must first be authenticated, and then requests must be authorized to verify that users are permitted to make those specific requests. Responses should also be verified such that users are authorized to view the information contained in the response.

These two steps above are performed by two new modules added to the Security element, an **authentication service**, and an accompanying **authorization service**. These services check user credentials and permissions stored in the Data Persistence element before allowing or blocking potential requests or responses from passing out of or into the User and Device Interface element. Our design for this decomposition is based on two Security tactics: “*Resist Attacks – Authenticate Actors*” and “*Resist Attacks – Authorize Actors*”.

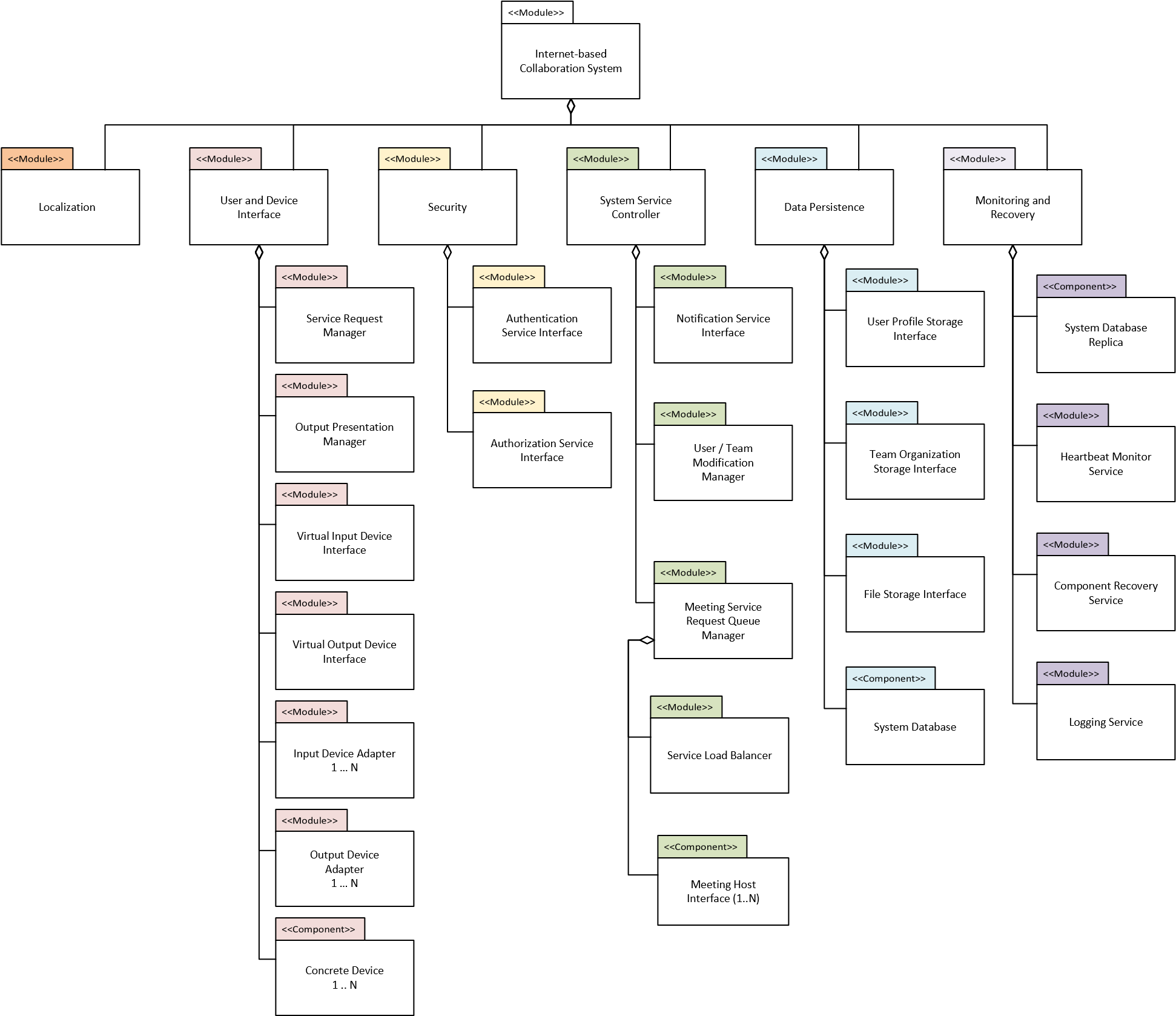
# 2.5.2 Trade Off Analysis

Including these intermediate steps between user actions and the system response will introduce increased latency for service requests and responses. This is in direct opposition to the original goals of the #1 architectural driver of providing users with low latency and responsive communication tools. Some disbenefit can likely be mitigated through the caching of authentication and authorization requests, perhaps allowing for a certain timeframe of requests to pass through the Security element only once instead of each time. The addition of these security modules also increases the complexity of the system without providing any new features to users or system developers. However, the implications of nefarious use of the system may be greater than the performance disbenefits and increased complexity added in this iteration. Additionally, it is possible that not all devices would be compatible with the Authentication or Authorization Services, requiring the system to either not support those devices or provide workarounds which may weaken the security of the system.

## Sixth Iteration: Monitoring and Recovery (Monitoring and Recovery)

In this iteration, we have selected the **Monitoring and Recovery** element as the target for decomposition, and “*Monitor and recover from system connection losses and errors*” as the architectural driver. The initial results of this decomposition are shown in Figure 16, below, with a table showing functional responsibilities and a component-connector view following in Figure 17 and Figure 18, respectively. An explanation and analysis of our design follows these figures.

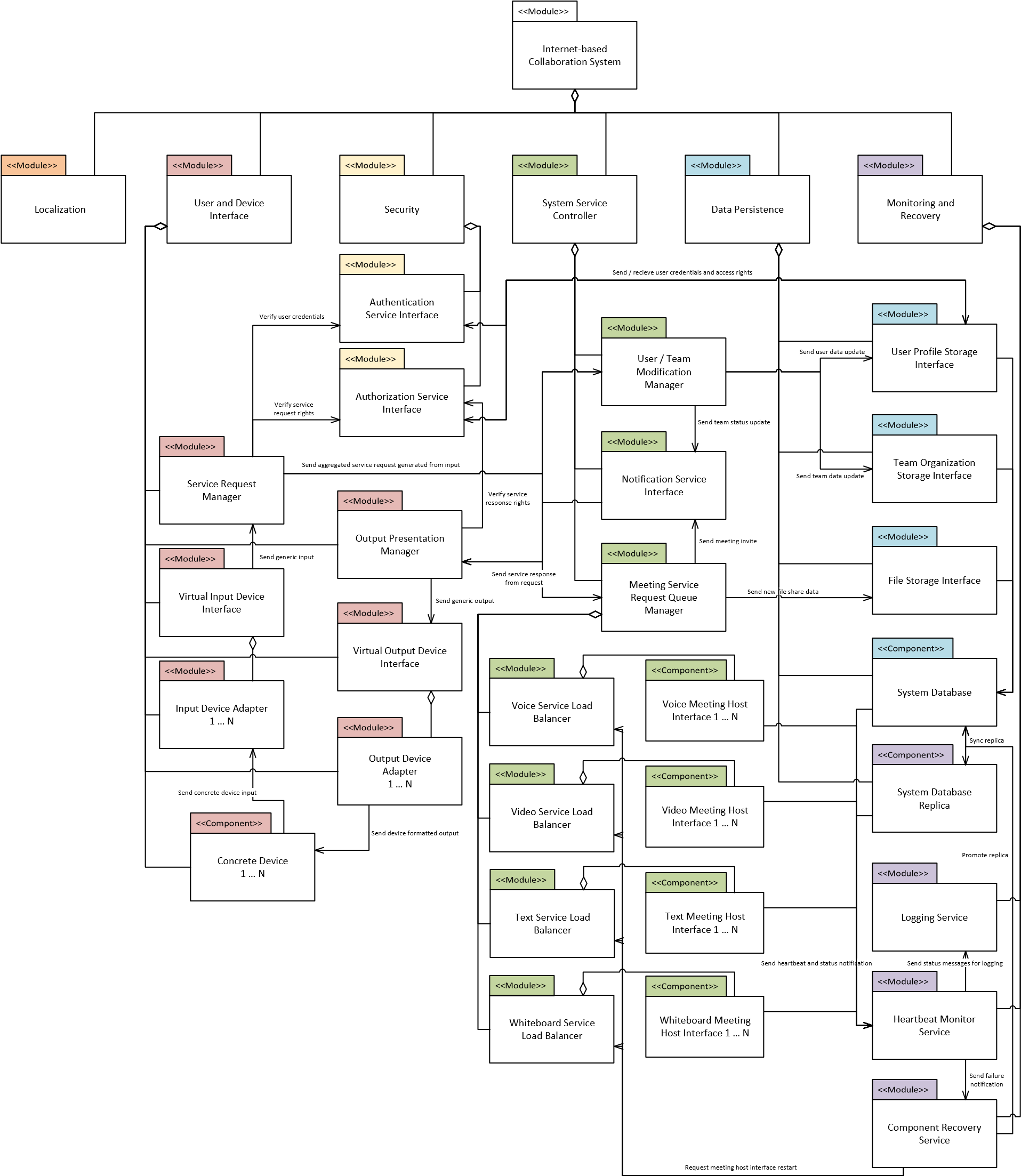
*Figure 16 -* *System Decomposition Diagram of the Monitoring and Recovery Module and Arch. Driver #5*



## Figure 17 - Mapping of Functional Responsibility and Tactics to Sixth Iteration Modules

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Subcomponent | Architectural Driver | Functional Responsibility | Quality Attribute / Tactics |
| Monitoring and Recovery | System Database Replica | | 5. Monitor and recover from system connection losses and errors | 16. The system shall provide a means of data recover / resynchronization during connection losses | Availability:  Spare - Hot Spare |
| Component Recovery Service | | Availability:  Preparation and Repair - Retry |
| Heartbeat Monitor Service | | 17. The system shall provide a means to view its performance and errors in an industry standard way | Availability:  Detect Faults - Heartbeat |
| Logging Service | |

## *Figure 18 - Component Connector Diagram for Sixth Iteration Modules*



# 2.6.1 Design and Tactics

For this iteration, we employed tactics from the Availability quality attribute, namely “*Detect Faults – Heartbeat*”, “*Preparation and Repair – Retry*”, and “*Spare – Hot Spare*”. For the design of this decomposition, we focused on the ability for critical services to be monitoring for connection loss, and the ability to quickly recover from those losses. To that end, we added three new modules and one new component to the Monitoring and Recovery element as part of this design: a **heartbeat monitor service**, a **logging service,** a **component recovery service**, and a **replica database**.

The heartbeat monitor service receives periodic status messages from critical components (in the case of this current design, the meeting host services and the system database are considered critical) that allow it to assess the state of those components. If a message is not received, or the status is not nominal, the heartbeat monitor service notifies the component recovery service to request a restart of that failing component. In the case of the meeting host services, the load balancers created in a previous iteration provide an easy avenue for restarting a service. The component recovery service can simply request the load balancer to start another host and reconnect its interface with the previous host. In the case a database failure, the component recovery service can promote the system database replica (which is being synced as a hot spare) until the main system database can be restarted. The status messages received from the heartbeat monitor service are also passed along to the logging service, where they are recorded for reference or debugging by the system developers.

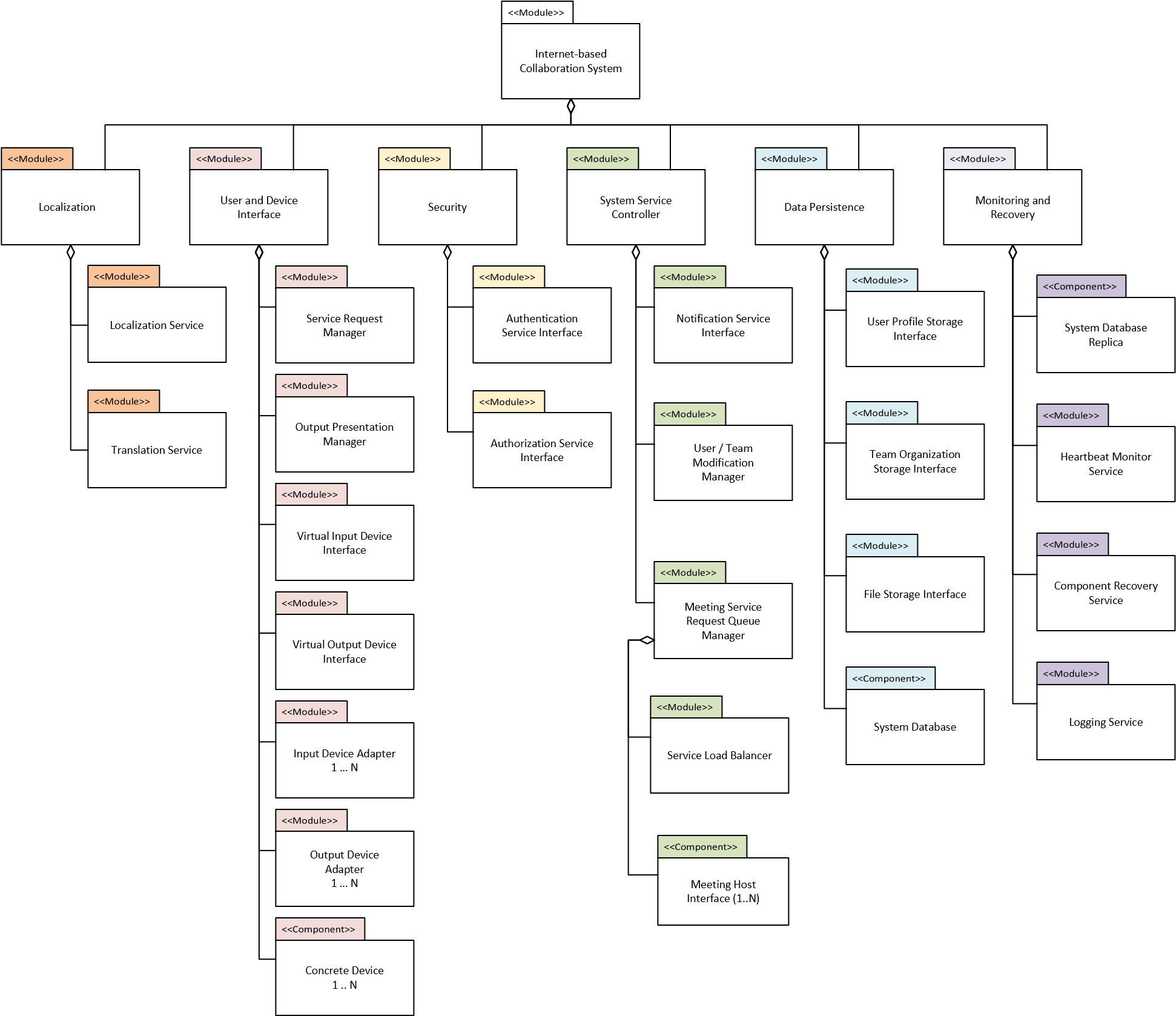
# 2.6.2 Trade Off Analysis

We found it difficult to find trade offs for the heartbeat monitor service, logging service, and component recovery service, since these modules use other modules already created as part of different architectural drivers, and would add a negligible amount of communication between themselves. Perhaps the additional complexity of automatic restarts on services would make for slightly harder maintenance, but that seems entirely worth the small additional hassle. The database replica, however, may have large trade offs to implement. If the system grows large, mirroring the database could become quite costly. Managing the synchronization between the main database and the replica may also prove to be a lot of additional work. This replica database provides no additional features for users or developers until the main database fails, increasing costs and developer time for ‘just sitting there’. However, we feel the ability to quickly recover in the case of a failure is well worth these additional issues.

## Seventh Iteration: Global Support (Localization)

In this iteration, we have selected the **Localization** element as the target for decomposition, and “*Provide language localization and global support*” as the architectural driver. The initial results of this decomposition are shown in Figure 19, below, with a table showing functional responsibilities and a component-connector view following in Figure 20 and Figure 21, respectively. An explanation and analysis of our design follows these figures.

*Figure 19 -* *System Decomposition Diagram of the Localization Module and Arch. Driver #6*



*Figure 20 - Mapping of Functional Responsibility and Tactics to Seventh Iteration Modules*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Subcomponent | Architectural Driver | Functional Responsibility | Quality Attribute / Tactics |
| Localization | Localization Service | | 6. Provide language localization and global support | 1. The system shall provide UI elements in a language requested by the user 2. The system shall accept user input in the language provided by the user | Usability: Support  System Initiative -  Maintain User Model |
| Translation Service | |

*Figure 21 - Component Connector Diagram for Seventh Iteration Modules*



# 2.7.1 Design and Tactics

In this iteration, we employed the Usability tactic “*Support System Initiative – Maintain User Model”*. The goal of using this tactic is the idea that any user, regardless of region or language, can interact with the system in the same way as expected for the English language. This created the need for two additional modules: a **Localization Service**, which can detect and store user location and language preferences; and a **Translation Service**, storing UI element dictionaries in many languages and providing translations for the user specified language.

Similarly to the Security modules, these new modules also act as an intermediary for the Output Presentation Manager, before UI elements are displayed to the user. When a UI element is ready for localization (and before being displayed) the users preferences for language are either detected by location (if no preference has been specified), or are requested from the System Database through the User Profile Storage Interface. If translation is required, the UI element is passed to the Translation Service, where an appropriate translation is generated and sent back to the Localization Service. From here, the translated element is passed back to the Output Presentation Manager where it can be displayed to the user in their requested language.

# 2.7.2 Trade Off Analysis

We considered several trade-offs in this iteration. Firstly, the design will never be able to fully recreate the experience in all languages (the user interface will likely struggle to show certain characters from less common languages). Secondly, the user interface will also likely have to change layout to accommodate the difference in size of words or phrases in different languages. Thirdly is that regulatory laws will likely vary from country to country. Therefore, additional resources will be needed to ensure compliance with laws regarding translations. For example, user manuals might need to be translated to native languages in some countries but not all. Another example would be that data privacy is different around the globe so detecting a device’s location to setup the user interface in a local language might not be legal.

# Overall System Trade Off Analysis

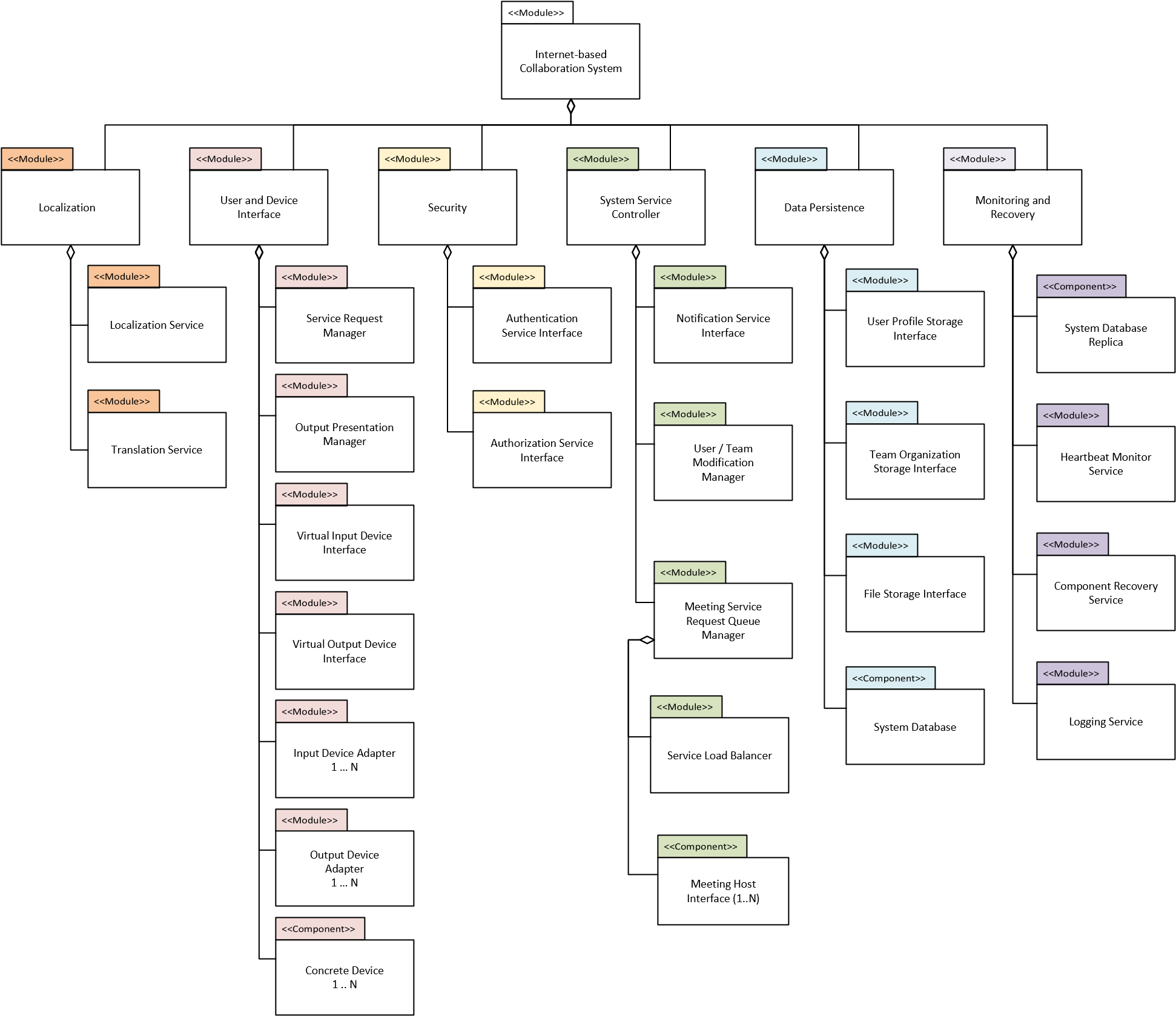
(Please see individual decomposition sections for trade off analysis specific to that decomposition).

This system contains a handful of particularly tricky trade offs between its architectural drivers. The most obvious is the trade off between performance (the #1 architectural driver) and other lower priority drivers like security (#3) and modifiability (#4). Adding layers of process between user actions and system response for abstraction and security concerns will necessarily slow the performance. However, we feel these trade offs are necessary for the long-term stability of the system in general. A security breach or the inability to add a new device to the system will hurt users much more tangibly than a slight drop in performance.

Other trade offs concern cost and complexity. Scaling and replication will add significant cost to the system, and may require limits on the system’s ability to perform these kinds of operations (without business approval). Each decomposition from this point on will likely add complexity as well, making each subsequent update more involved to implement.

# Proposed Architecture

*Figure 22 -* *System Decomposition Diagram of the Internet-Based Collaboration System*



*Figure 23 - Mapping of Functional Responsibility and Tactics for the Internet-Based Collaboration System*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Component | Subcomponent | Architectural Driver | Functional Responsibility | Quality  Attribute /  Tactics |
| User and  Device Interface | Service  Request  Manager | Virtual Input Device | 4. Interface with a variety of current user input and output devices | 8a, b, c, d. The system shall be capable of detecting and interfacing with a variety of user input devices | Modifiability:  Encapsulation,  Increase  Semantic  Coherence |
| Input Device Adapter |
| Output  Presentation  Manager | Virtual  Output  Device | 9a, b, c, d. The system shall be capable of detecting and |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Output  Device  Adapter |  | interfacing with a variety of user output devices |  |
|  | Concrete Device | | 10. The system shall allow for the detection and addition of new devices | Modifiability:  Defer Bindings |
|  | Authentication Service Interface | | 3. Provide for user authentication and the authorization of user actions | 14. The system shall provide a means to authenticate user access to the system | Security: Resist  Attacks - Authenticate  Actors,  Authorize  Actors, Limit Access |
| Security | Authorization Service Interface | | 15. The system shall provide a means to authorize / deauthorize user actions on the system |
| System  Service Controller |  |  | 1. Provide low latency interaction for video, voice, and other team communication; capable of scaling on commercially available hardware | 2. The system shall maintain a responsive UI during user interaction | Performance:  Control  Resource  Demand –  Prioritize  Events |
| Meeting  Service  Request Queue  Manager | Voice Service  Load  Balancer | 1a,b,d,e. The system shall provide user-expected communication methods | Performance:  Manage  Resources -  Increase  Concurrency,  Increase  Resources |
| Voice  Meeting Host  Interface |
| Video Service  Load  Balancer |
| Video  Meeting Host  Interface |
| Text Service  Load  Balancer |
| Text Meeting  Host Interface |
| Whiteboard  Service Load  Balancer |
| Whiteboard  Meeting Host  Interface |
| Notification Service Interface | | 2. Allow for the creation and modification of teams and team structure by users 4. Interface with a variety of current user input and output devices | 1. The system shall allow users to start group meetings with other remote users 2. The system shall allow users to be added to a group meeting | Interoperability:  Manage  Services - Orchestrate |
|  | User / Team  Modification Manager | User Profile  Service  Interface | 2. Allow for the creation and modification of teams and team structure by users | 5. The system shall provide users a means to invite new users to the system | Usability:  Support User  Initiative - Aggregate,  Support System  Initiative -  Maintain Task  Model |
|  | Team  Organization  Storage  Interface | 3. The system shall provide users a means to create teams 4. The system shall provide users a means to add and remove other users from a team |
|  | System Database | | 13. The system shall comply with the local laws and regulations of a user | Usability:  Support System  Initiative -  Maintain User Model |
| Data Persistence |
| File Storage Interface | | 1. Provide low latency interaction for video, voice, and other team communication; capable of scaling on commercially available hardware | 1c. The system shall provide user-expected communication methods | Usability:  Support System  Initiative -  Maintain Task  Model |
| Monitoring and Recovery | System Database Replica | | 5. Monitor and recover from system connection losses and errors | 16. The system shall provide a means of data recover / resynchronization during connection losses | Availability:  Spare - Hot  Spare |
| Component Recovery Service | | Availability:  Preparation and  Repair - Retry |
| Heartbeat Monitor Service | | 17. The system shall provide a means to view its performance and errors in an industry standard way | Availability:  Detect Faults - Heartbeat |
| Logging Service | |
| Localization | Localization Service | | 6. Provide language localization and global support | 11. The system shall provide UI elements in a language requested by the user | Usability:  Support System  Initiative -  Maintain User  Model |
| Translation Service | | 12. The system shall accept user input in the language provided by the user |

*Figure 24 - Component Connector Diagram for the Internet-Based Collaboration System*

