CS 487 – SOFTWARE ENGINEERING

Homework – 2

Suhas Palani **A20548277** spalani3@hawk.iit.edu

Title: AI Audio System

The AI Audio System, an advanced in-car entertainment solution that uses artificial intelligence to provide a personalized audio experience. Key features include mood-based content selection, multi-modal interaction (voice, touch, gesture), and emotional awareness through sensors. The system adapts to user preferences over time, integrating multiple audio sources (satellite, terrestrial radio) for a seamless experience. By responding to occupants' emotional states and preferences, the AI Audio System creates a more engaging and personalized environment.

1. Context Model

This context diagram shows the AI Audio System at the centre, with its various partners:

- **Occupants:** They interact with the system by making requests and receiving audio output.
- **Emotion Sensors:** These provide emotion data to the system.
- **Satellite Radio:** These provide content to the system.
- **User Preferences Database:** This stores and provides user preference data.
- **Terrestrial Radio:** This also provides content to the system, offering an alternative source to satellite radio.

➢ **External Systems:**

- Satellite Radio (SR)
- Terrestrial Radio (TR)
- User Preferences Database (UP)

These are considered external systems because they are separate, complex systems that the AI Audio System interacts with but doesn't control directly. They provide services (content or data storage/retrieval) to our system.

➢ **External Input Devices:**

- Emotion Sensors (ES)

Emotion Sensors are classified as external input devices because they only provide input to the AI Audio System. They capture data about the occupants' emotional states and feed this information into the system, but don't receive any output from it.

➢ **External I/O (Input/Output) Devices:**

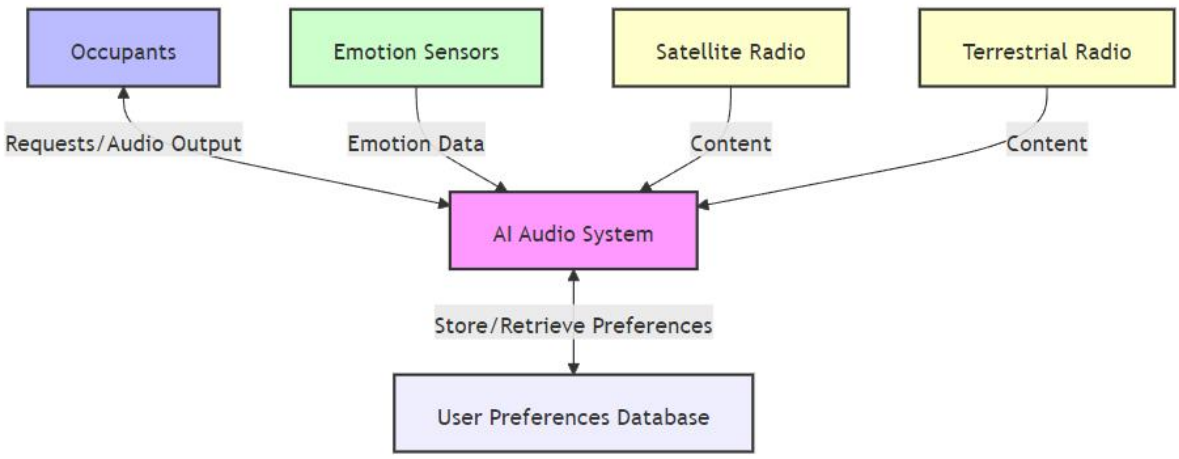
- Occupants (OC)

The Occupants are considered an external I/O because they both provide input to the system (in the form of requests) and receive output from it (in the form of audio). This two-way interaction makes it an I/O component rather than just an input or output device.

```
graph TD
    AS[AI Audio System]
    OC[Occupants]
    ES[Emotion Sensors]
    SR[Satellite Radio]
    TR[Terrestrial Radio]
    UP[User Preferences Database]

    OC <--> |Requests/Audio Output| AS
    ES --> |Emotion Data| AS
    SR --> |Content| AS
    TR --> |Content| AS
    AS <--> |Store/Retrieve Preferences| UP

    style AS fill:#f9f,stroke:#333,stroke-width:2px
    style OC fill:#bbf,stroke:#333,stroke-width:2px
    style ES fill:#cfc,stroke:#333,stroke-width:2px
    style SR fill:#ffc,stroke:#333,stroke-width:2px
    style TR fill:#ffc,stroke:#333,stroke-width:2px
    style UP fill:#eef,stroke:#333,stroke-width:2px
```



2. Human-Computer Interaction (HCI) Protocols

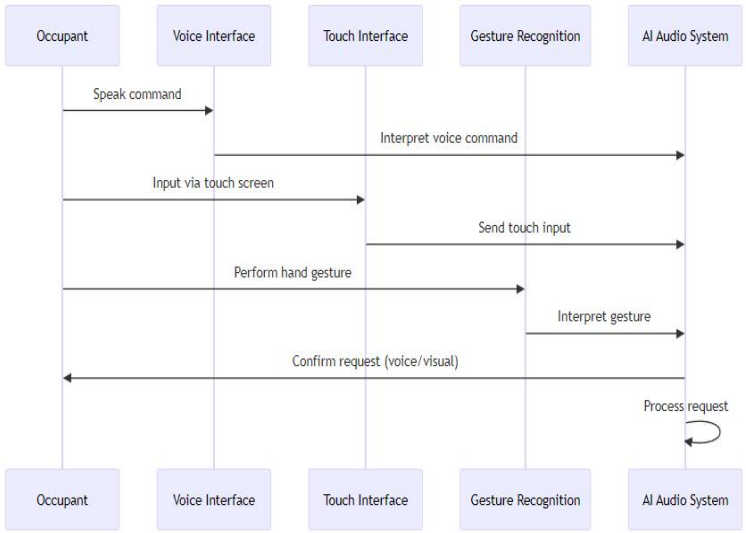
Let's specify the HCI protocols for each of the relationships mentioned:

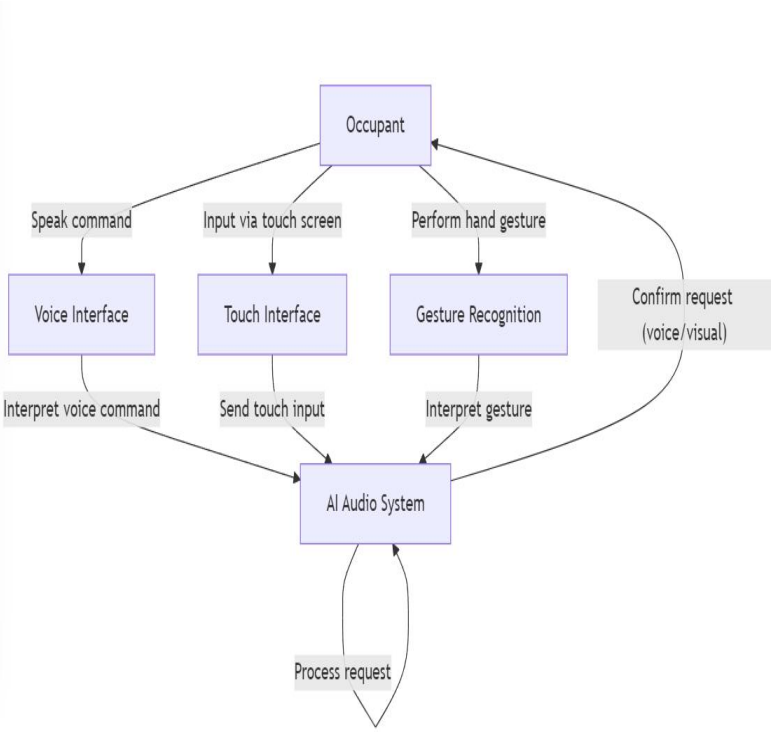
a) **Accepting requests from occupants:**

- Voice Interface: The system will use natural language processing to understand voice commands from occupants.
- Touch Interface: A touchscreen displays in the car dashboard for manual input.
- Gesture Recognition: Camera-based system to recognize simple hand gestures for basic controls.

```
sequenceDiagram
    participant O as Occupant
    participant VI as Voice Interface
    participant TI as Touch Interface
    participant GR as Gesture Recognition
    participant AS as AI Audio System

    O->>VI: Speak command
    VI->>AS: Interpret voice command
    O->>TI: Input via touch screen
    TI->>AS: Send touch input
    O->>GR: Perform hand gesture
    GR->>AS: Interpret gesture
    AS->>O: Confirm request (voice/visual)
    AS->>AS: Process request
```





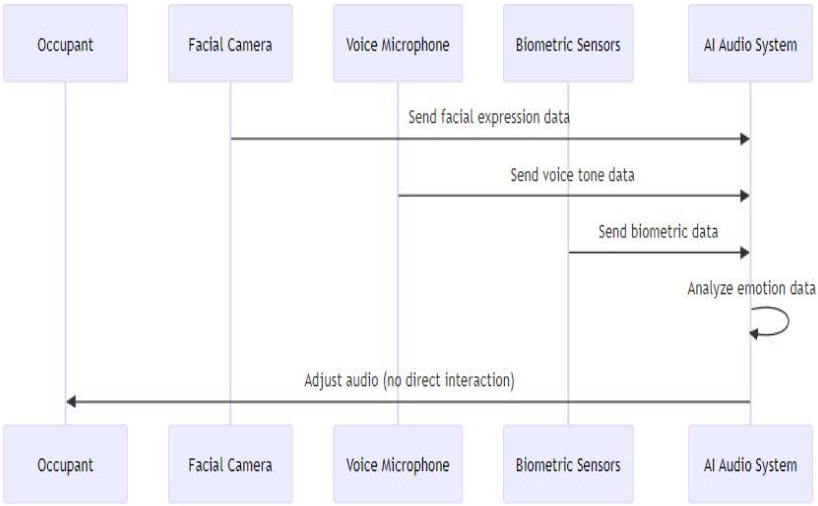

```
flowchart TD
    A[Occupant] -->|Speak command| B[Voice Interface]
    B -->|Interpret voice command| C[AI Audio System]
    A -->|Input via touch screen| D[Touch Interface]
    D -->|Send touch input| C
```

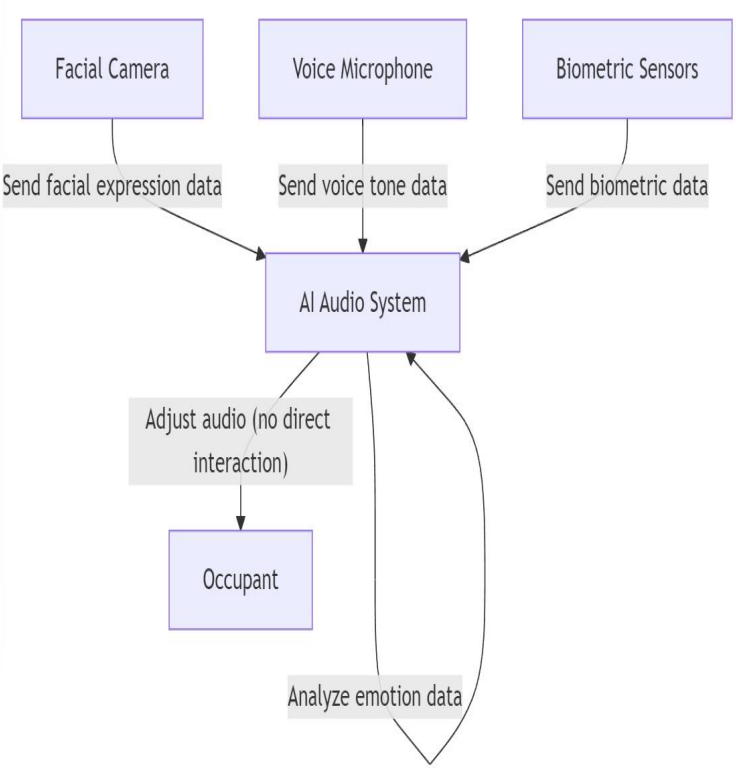
b) **Interfacing with emotion sensors:**

- Facial Expression Analysis: Cameras will capture facial expressions and analyse them for emotional cues.
- Voice Tone Analysis: The system will analyse the tone and pitch of occupants' voices to detect emotions.
- Biometric Sensors: Sensors in the steering wheel and seats can detect heart rate, skin conductivity, and other physiological indicators of emotion.

```
sequenceDiagram
    participant O as Occupant
    participant FC as Facial Camera
    participant VM as Voice Microphone
    participant BS as Biometric Sensors
    participant AS as AI Audio System

    FC->>AS: Send facial expression data
    VM->>AS: Send voice tone data
    BS->>AS: Send biometric data
    AS->>AS: Analyze emotion data
    AS->>O: Adjust audio (no direct interaction)
```





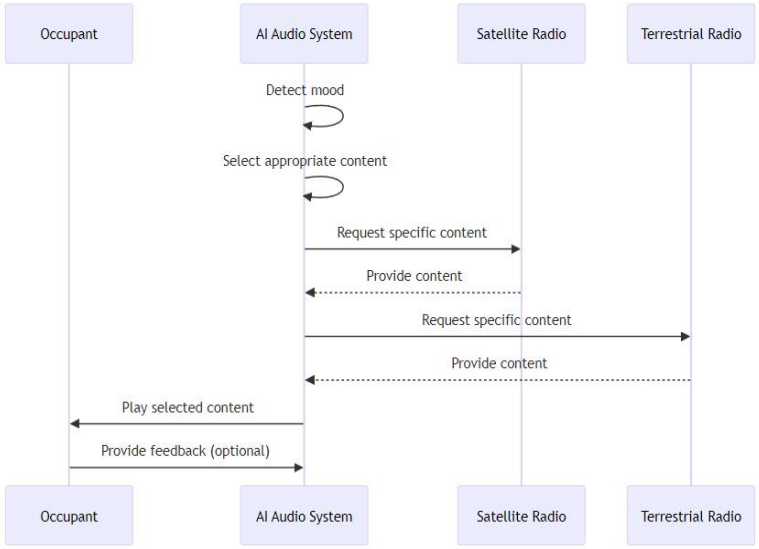

```
flowchart TD
    A[Facial Camera] -->|Send facial expression data| B[AI Audio System]
    C[Voice Microphone] -->|Send voice tone data| B
    D[Biometric Sensors] -->|Send biometric data| B
    B -->|Analyze emotion data| B
    B -->|"Adjust audio (no direct interaction)"| E[Occupant]
```

c) **Providing content based on mood:**

- Mood-Content Mapping: The system will maintain a database mapping emotional states to appropriate content types.
- Dynamic Playlist Generation: Based on detected mood and preferences, the system will create real-time playlists.
- Seamless Source Switching: The system will switch between satellite and radio sources based on content availability and mood appropriateness.

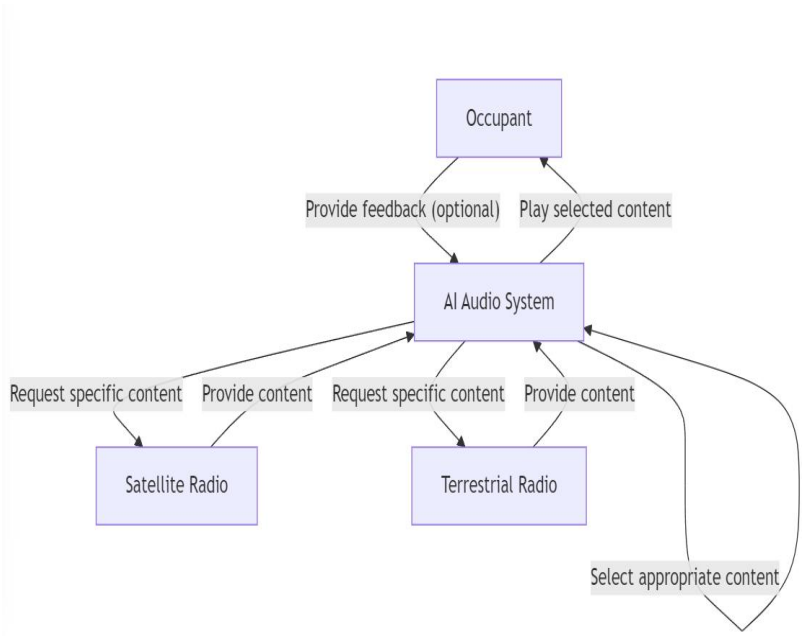
```
sequenceDiagram
    participant O as Occupant
    participant AS as AI Audio System
    participant SR as Satellite Radio
    participant TR as Terrestrial Radio

    AS->>AS: Detect mood
    AS->>AS: Select appropriate content
    AS->>SR: Request specific content
    SR-->>AS: Provide content
    AS->>TR: Request specific content
    TR-->>AS: Provide content
    AS->>O: Play selected content
    O->>AS: Provide feedback (optional)
```




```
flowchart TD
    O[Occupant]
    AS[AI Audio System]
    SR[Satellite Radio]
    TR[Terrestrial Radio]

    AS -->|Detect mood| AS
    AS -->|Select appropriate content| AS
    AS -->|Request specific content| SR
    SR -->|Provide content| AS
    AS -->|Request specific content| TR
```

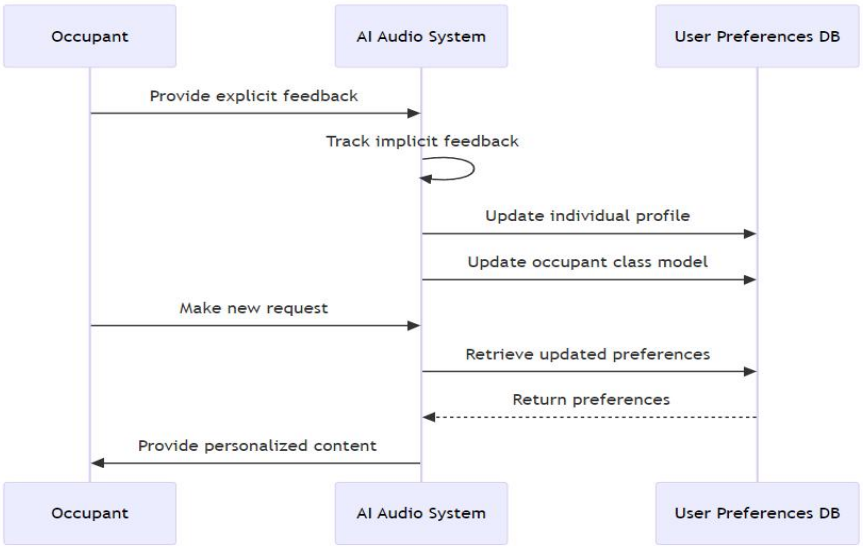


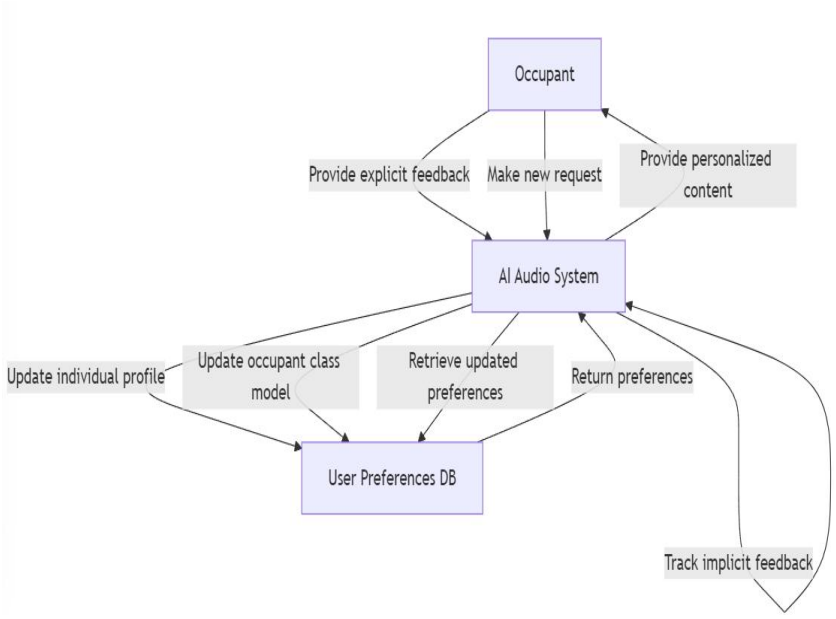
d) **Learning user preferences:**

- Explicit Feedback: Allow users to rate content and create favorites.
- Implicit Feedback: Track listening duration, skips, and repeats to infer preferences.
- User Profiles: Create and update individual profiles for frequent occupants.
- Occupant Classification: Develop preference models for different occupant classes (e.g., age groups, frequent vs. occasional users).

```
sequenceDiagram
    participant O as Occupant
    participant AS as AI Audio System
    participant UP as User Preferences DB

    O->>AS: Provide explicit feedback
    AS->>AS: Track implicit feedback
    AS->>UP: Update individual profile
    AS->>UP: Update occupant class model
    O->>AS: Make new request
    AS->>UP: Retrieve updated preferences
    UP-->>AS: Return preferences
    AS->>O: Provide personalized content
```






```
flowchart TD
    O[Occupant]
    AS[AI Audio System]
    UP[User Preferences DB]

    O -->|Provide explicit feedback| AS
    AS -->|Track implicit feedback| AS
    AS -->|Update individual profile| UP
    AS -->|Update occupant class model| UP
    O -->|Make new request| AS
    AS -->|Retrieve updated
```

3. Automated Detection and Handling of Partner Failure

Let's use pseudocode to show the automated detection and handling of a failure in the Satellite Radio partner:

app.py > check_satellite_radio_status

```python
# Suhas Palani - A20548277 - spalani3@hawk.iit.edu

import time
import random  # Just a little something to simulate radio status

def send_request_to_satellite_radio():
    # Let's pretend we're asking the satellite radio for a status update.
    if random.choice([True, False]):
        return {"status": "OK"}  # Success! We're good to go.
    else:
        raise ConnectionError("Failed to connect")  # Oops, something went wrong.

def check_satellite_radio_status():
    try:
        response = send_request_to_satellite_radio()
        return response["status"] == "OK"  # Check if the status is all clear.
    except ConnectionError:
        log_error("Whoops! There was a connection issue while checking the satellite radio.")
        return False
    except TimeoutError:
        log_error("Oh no! The connection timed out.")
        return False

def handle_satellite_radio_failure():
    log_error("Yikes! The satellite radio isn't connecting.")
    notify_user("Looks like the satellite radio is down. Switching to terrestrial radio.")
    switch_to_terrestrial_radio()  # Time to switch gears!
    start_background_reconnection_attempts()  # Let's keep trying to reconnect in the background.

def main_audio_loop():
    while True:
        if not check_satellite_radio_status():
            handle_satellite_radio_failure()  # If the radio's down, handle it.

        # Here comes the fun part: the main audio system logic!
        process_user_requests()  # See what our users want.
        analyze_emotions()  # Let's tune into the mood.
        generate_playlist()  # Time to create a great playlist.
        play_audio()  # Let's get the music playing!

        if user_requests_shutdown():  # Check if it's time to wrap things up.
            break
```

```
def background_reconnection_attempt():
    max_retries = 12  # We'll try reconnecting a few times (about an hour if we wait 5 minutes each).
    retries = 0

    while retries < max_retries:
        if check_satellite_radio_status():
            log_info("Hooray! We've reconnected to the satellite radio.")
            notify_user("Great news! The satellite radio is back up.")
            return  # Exit the reconnection loop; we're good now!
        time.sleep(300)  # Let's wait 5 minutes before trying again.
        retries += 1

    log_error("Uh oh! We reached the maximum number of reconnection attempts.")

def switch_to_terrestrial_radio():
    # Here's where we switch to terrestrial radio.
    log_info("Switching gears to terrestrial radio now.")
    # Add the actual implementation for switching here.

def notify_user(message):
    # This is how we keep our users informed.
    print(f"User Notification: {message}")

def log_error(message):
    # Let's log any errors we encounter.
    print(f"ERROR: {message}")

def log_info(message):
    # Time to log some informative messages.
    print(f"INFO: {message}")

def process_user_requests():
    # This is where we handle user requests.
    pass

def analyze_emotions():
    # Here's where we get in touch with the emotional vibe.
    pass

def generate_playlist():
    # Let's whip up a playlist for our users.
    pass

def play_audio():
    # Time to start the music!
    pass

```

```
def user_requests_shutdown():
    # Check if the user wants to shut everything down.
    return False  # Adjust this based on how you want to handle shutdowns.

def start_background_reconnection_attempts():
    # In a real scenario, we'd run this in a separate thread.
    background_reconnection_attempt()

# And we're off! Start the main audio loop.
main_audio_loop()

```

This pseudocode demonstrates how the system would:

1. Regularly check the status of the satellite radio connection.
2. Detect a failure if the connection is lost or returns an error.
3. Handle the failure by logging the error, notifying the user, switching to terrestrial radio, and starting background reconnection attempts.
4. Continuously attempt to reconnect to the satellite radio in the background.
5. Notify the user and resume normal operation once the connection is re-established.

This approach ensures a seamless experience for the user, automatically handling partner failures and recovering when possible.