Impaired Navigation App

Process Specification

Phase I - Phase II

Chandler Guthrie, Erik Winiski, Julian Hutchins, Logan Cribbs, Matthew Hill, Yaru Gao

11/04/25

1. Introduction

The impaired navigation app is an application that is used to help users navigate inside a building by providing the best path for them to take and guiding them through that path, with the assumption they are visually impaired. Many more features will be developed through the course of the project to provide a prototype for our client from the specification and elicitation. The process in which we go from elicitation, to requirements and designs, then analyzing those problems and solutions, as well as reviewing those by other members, and finally implementation will be covered in this document.

2. The Team

Our Team is made up of 6 people in total, and we all participated in the documentation and designs as well as the implementation of the application. The members of the team are listed as follows: Chandler Guthrie, Erik Winiski, Julian Hutchins, Logan Cribbs, Matthew Hill, and Yaru Gao. Since we have a bigger team, we split up the work as equally as possible, which helped everyone stay on the same page with different documents, designs, and models during the documentation steps of this project.

3. Process Iterations Models & Activities

3.1. High-Level View Model

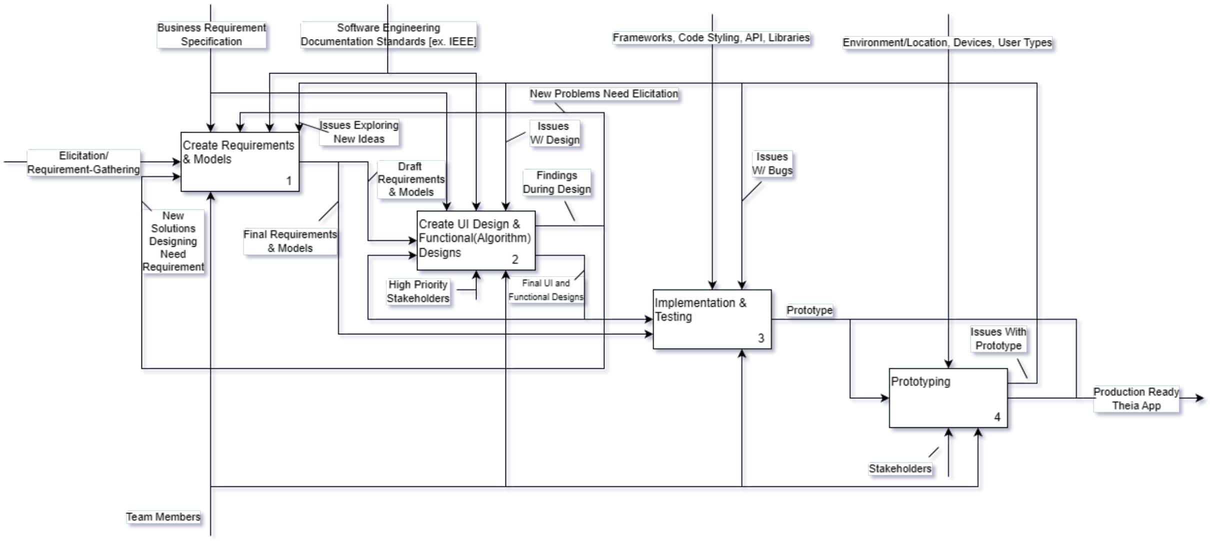

Business Requirement Specification
Software Engineering Documentation Standards [ex. IEEE]
Frameworks, Code Styling, API, Libraries
Environment/Location, Devices, User Types
New Problems Need Elicitation
Issues Exploring New Ideas
Issues W/ Design
Issues W/ Bugs
Elicitation/ Requirement-Gathering
Create Requirements & Models
1
Draft Requirements & Models
Findings During Design
New Solutions Designing Need Requirement
Final Requirements & Models
Create UI Design & Functional(Algorithm) Designs
2
High Priority Stakeholders
Final UI and Functional Designs
Implementation & Testing
3
Prototype
Issues With Prototype
Prototyping
4
Production Ready Theia App
Stakeholders
Team Members

Figure 3.1.1 High Level View IDEF0

The High Level view shows each function of our process iterations from the beginning of our phase 1 to the end of our phase 2. It also provides an in-depth showcase of how we dealt with changes and how that can affect different functions and the processes that go with them. Any possible problems with the prototype will be dealt with by confirming each function has been revisited to complete the project in an organized manner.

All of these also have controlling factors that drive how the different functions are solved, which can be a roadblock to some progress. Dealing with many of the functions, all team members played a part in them; however, diving deeper into each function in the following pages will show how each team member played a different and significant role in completing those functions and continuing them onto the next, until we finally have a working prototype or even a production-ready application.

3.2. Requirements & Models

Following the high-level view, the team constructed a series of models to interpret and refine the functionality of the Theia system. These include domain clarifications, scenario-based descriptions, and early structural models relating to navigation, obstacle detection, and user interaction. This stage also served as a bridge between the preliminary problem and the final WRS.

A key part of the section includes the KAOS models that were developed to address the requirements of the Theia project. These models were refined to match information from the team's WRS document.

Below is each KAOS model sectioned by type that includes a description of their purpose and what requirements they address.

KAOS Obstacle Models:

The KAOS obstacle models identify the conditions that could prevent each system goal from being achieved and outline the strategies that would resolve those risks. By linking goals, subgoals, obstacles, and resolutions, these models provide a clear understanding of where failures may occur and how the system should respond to maintain reliable behavior.

Figure 3.2.1 G1: Provide Reliable Indoor Navigation

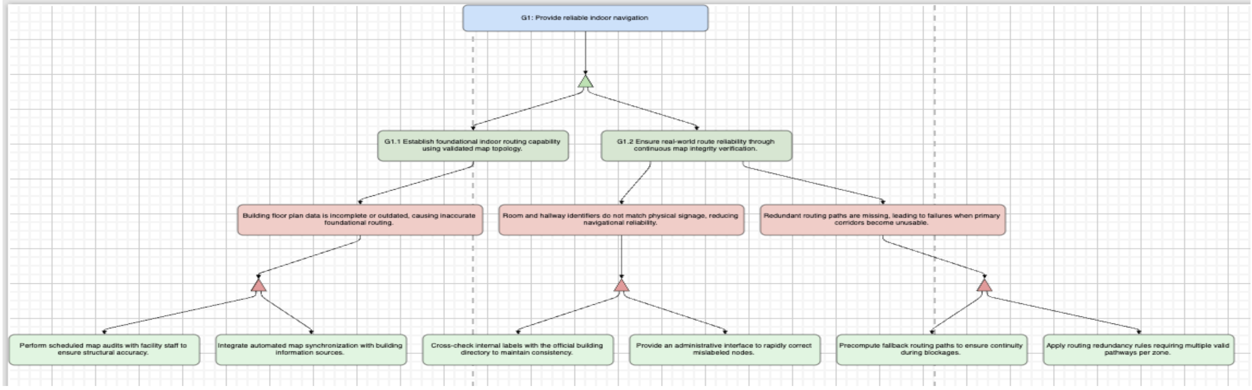

G1: Provide reliable indoor navigation
G1.1 Establish foundational indoor routing capability using validated map topology.
G1.2 Ensure real-world route reliability through continuous map integrity verification.
Building floor plan data is incomplete or outdated, causing inaccurate foundational routing.
Room and hallway identifiers do not match physical signage, reducing navigational reliability.
Redundant routing paths are missing, leading to failures when primary corridors become unusable.
Perform scheduled map audits with facility staff to ensure structural accuracy.
Integrate automated map synchronization with building information sources.
Cross-check internal labels with the official building directory to maintain consistency.
Provide an administrative interface to rapidly correct mislabeled nodes.
Precompute fallback routing paths to ensure continuity during blockages.
Apply routing redundancy rules requiring multiple valid pathways per zone.

Figure 3.2.2 G3: Generate Accurate Indoor Routes for User Navigation

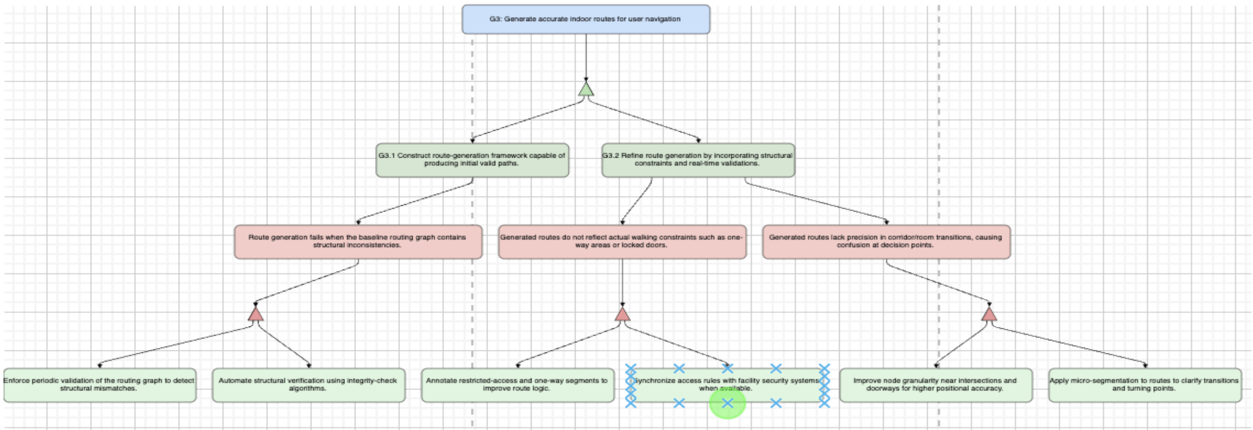

G3: Generate accurate indoor routes for user navigation
G3.1 Construct route-generation framework capable of producing initial valid paths.
G3.2 Refine route generation by incorporating structural constraints and real-time validations.
Route generation fails when the baseline routing graph contains structural inconsistencies.
Generated routes do not reflect actual walking constraints such as one-way areas or locked doors.
Generated routes lack precision in corridor/room transitions, causing confusion at decision points.
Enforce periodic validation of the routing graph to detect structural mismatches.
Automate structural verification using integrity-check algorithms.
Annotate restricted-access and one-way segments to improve route logic.
Synchronize access rules with facility security systems when available.
Improve node granularity near intersections and doorways for higher positional accuracy.
Apply micro-segmentation to routes to clarify transitions and turning points.

Figure 3.2.3 G4: Improve Indoor Sensor Reliability for Indoor Positioning

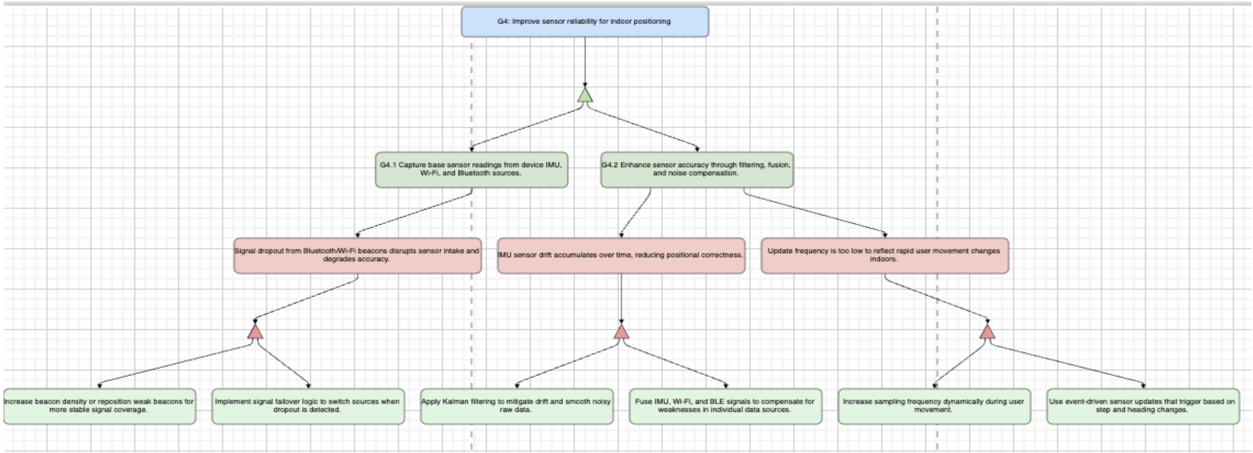

G4: Improve sensor reliability for indoor positioning
G4.1 Capture base sensor readings from device IMU, Wi-Fi, and Bluetooth sources.
G4.2 Enhance sensor accuracy through filtering, fusion, and noise compensation.
Signal dropout from Bluetooth/Wi-Fi beacons disrupts sensor intake and degrades accuracy.
IMU sensor drift accumulates over time, reducing positional correctness.
Update frequency is too low to reflect rapid user movement changes indoors.
Increase beacon density or reposition weak beacons for more stable signal coverage.
Implement signal failover logic to switch sources when dropout is detected.
Apply Kalman filtering to mitigate drift and smooth noisy raw data.
Fuse IMU, Wi-Fi, and BLE signals to compensate for weaknesses in individual data sources.
Increase sampling frequency dynamically during user movement.
Use event-driven sensor updates that trigger based on step and heading changes.

Figure 3.2.4 G6: Implement Real-Time Obstacle Detection for Safer Indoor Navigation

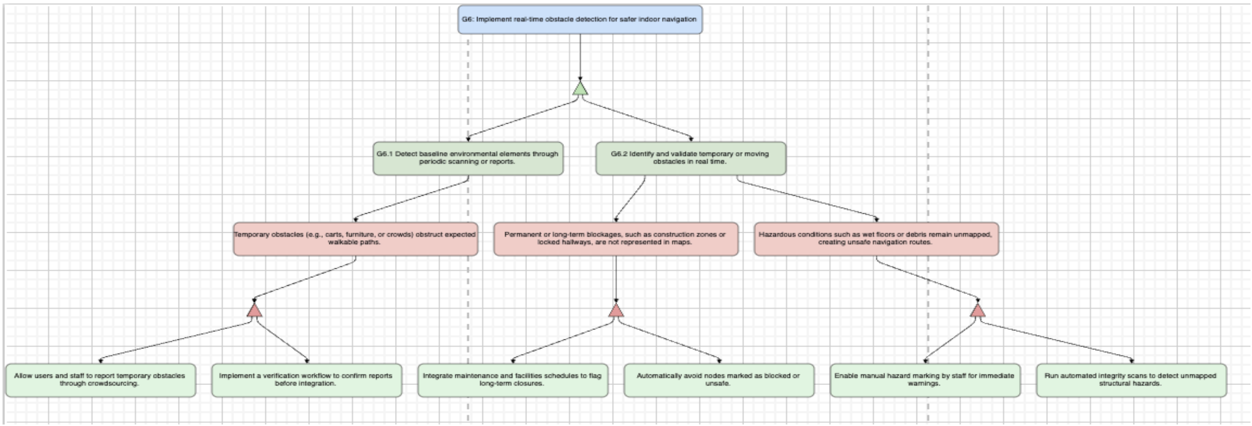
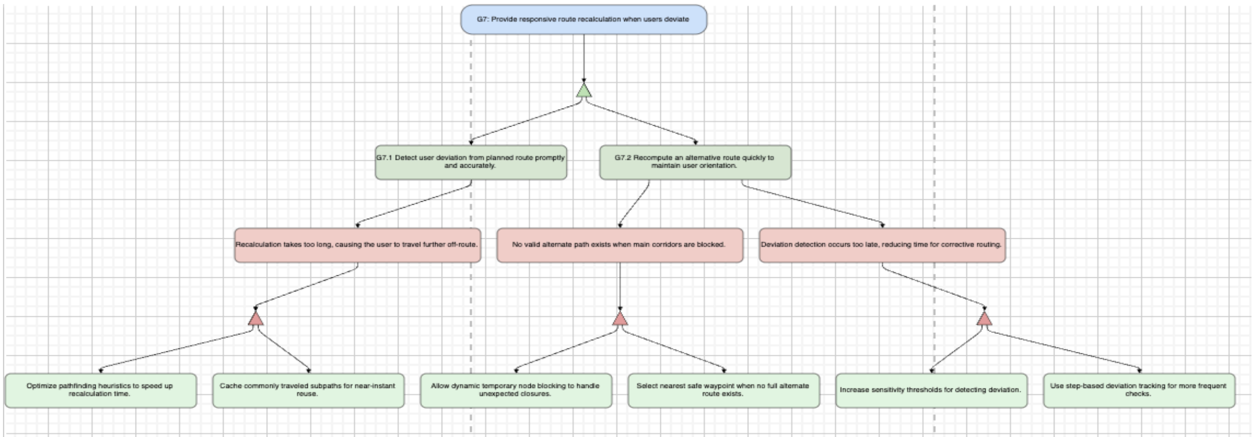

G6: Implement real-time obstacle detection for safer indoor navigation
G6.1 Detect baseline environmental elements through periodic scanning or reports.
G6.2 Identify and validate temporary or moving obstacles in real time.
Temporary obstacles (e.g., carts, furniture, or crowds) obstruct expected walkable paths.
Permanent or long-term blockages, such as construction zones or locked hallways, are not represented in maps.
Hazardous conditions such as wet floors or debris remain unmapped, creating unsafe navigation routes.
Allow users and staff to report temporary obstacles through crowdsourcing.
Implement a verification workflow to confirm reports before integration.
Integrate maintenance and facilities schedules to flag long-term closures.
Automatically avoid nodes marked as blocked or unsafe.
Enable manual hazard marking by staff for immediate warnings.
Run automated integrity scans to detect unmapped structural hazards.

Figure 3.2.5 G7: Provide Responsive Route Recalculation When Users Deviate


G7: Provide responsive route recalculation when users deviate
G7.1 Detect user deviation from planned route promptly and accurately.
G7.2 Recompute an alternative route quickly to maintain user orientation.
Recalculation takes too long, causing the user to travel further off-route.
No valid alternate path exists when main corridors are blocked.
Deviation detection occurs too late, reducing time for corrective routing.
Optimize pathfinding heuristics to speed up recalculation time.
Cache commonly traveled subpaths for near-instant reuse.
Allow dynamic temporary node blocking to handle unexpected closures.
Select nearest safe waypoint when no full alternate route exists.
Increase sensitivity thresholds for detecting deviation.
Use step-based deviation tracking for more frequent checks.

(I believe all KOAS models would go into subsections within 3.2; they can be added as we produce them)

3.3. UI Design & Functional Design

This phase focused on establishing how users interact with the application and how the system delivers its core functionality. UI flows, early mockups, and functional decomposition diagrams were developed to align user needs with the functional and non-functional requirements identified earlier. As the design evolved, feedback from the KAOS models and domain reasoning was incorporated to ensure clarity, accessibility, and alignment with stakeholder expectations.

3.4. Implementation

During implementation, the team transformed validated requirements and models into software artifacts. This iterative development cycle included code reviews, version control workflows, and the incorporation of prototype components such as navigation logic, basic sensors, data storage, and voice interaction. Each implementation step traced back to the requirements and obstacle resolutions identified earlier, ensuring alignment with the WRS.

3.5. Prototyping

The prototyping phase produced functional demonstrations of key system features, including navigation routes, hazard alerts, and voice-guided assistance. These prototypes provided a low-fidelity but testable version of the system, enabling the team to validate UI interactions, check compliance with accessibility needs, and confirm that the resolutions proposed in the KAOS obstacle models were practical within the system's architecture.

4. What’s Next

After completing Phase II and demonstrating the three selected prototype features, the next steps for the project would focus on expanding the system beyond the minimal functionality needed for the team's demo presentation. If the project were to continue, the team would begin refining the initial prototype, improving stability, and gradually incorporating additional requirements from the WRS. This would include strengthening navigation accuracy, expanding sensor integration, and enhancing obstacle-handling logic based on the risks identified in the KAOS models.

Future work would also involve integrating more complete UI interactions, improving accessibility features, and validating the system with more realistic movement scenarios. Additionally, with the use of goal diagrams, domain models, or an extended KAOS analysis, the team would be able to verify that the design remains consistent and well-documented with further development. While full implementation is beyond the scope of this class, these steps outline how the project could continue to evolve into a more complete indoor navigation solution if development were extended.