UNIVERSIDAD DE PUERTO RICO
RECINTO UNIVERSITARIO DE MAYAGÜEZ
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

Is the A/C on?
Architecture and Design Documentation

Andrea Rodríguez Astacio
Erika Pagán Vélez
Kevin Lorenzo Vega

Index

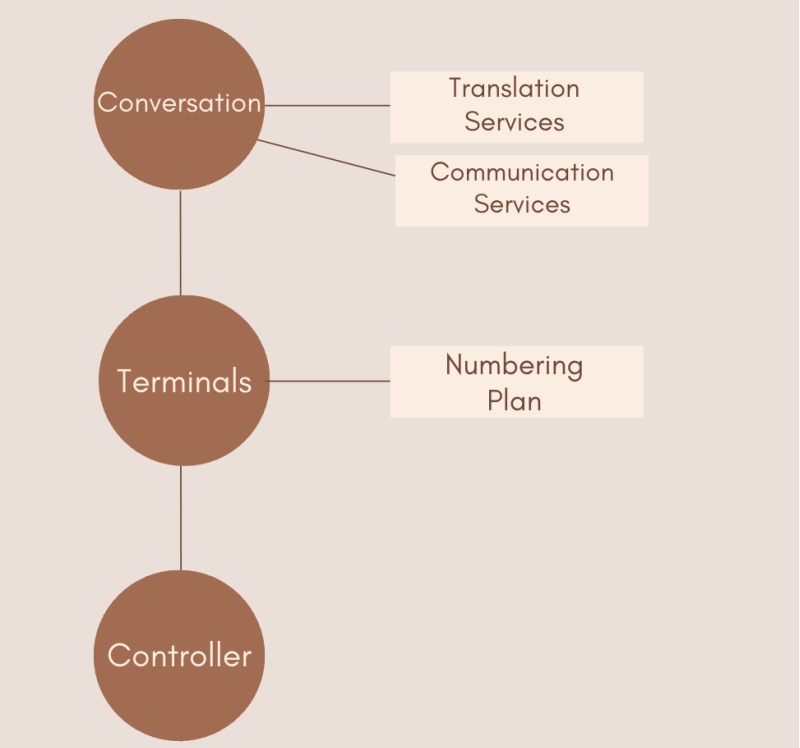
SECTION A. DESCRIPTION OF PROJECT

A.1 Purpose and general description of project

>> Air conditioners are normally controlled by an IR remote control that sends or transmits infrared signals in the form of hexCode or raw data when a certain button is pressed. Then the receiver in the A/C interprets the signal codes and converts them into instructions to perform certain tasks such as turning on/off, increasing/reducing temperature, changing modes etc. In our project there are ESP32's equipped with Temperature sensors and proximity sensors that are located in several rooms in the buildings across campus. These devices send constant measurements to a central computer in the cloud. End Users can go to a web site and review the current and historical temperature measurements. They can also query the system via a "Siri" or "OK Google" shortcuts. For this to work we need to make sure the ESP32 can support calibration for the temperature sensor, commissioning (information about the room, location, wifi,etc.), configuration (changing location name, time, units) and policy (specific measurement rate at different times of day) by subscribing to a device-specific MQTT Topic, it also need to support sending temperature measurements to the central server and support local visualization and interaction; be it via LED/LCD display, keypad, etc.

SECTION B. ARCHITECTURE MODEL

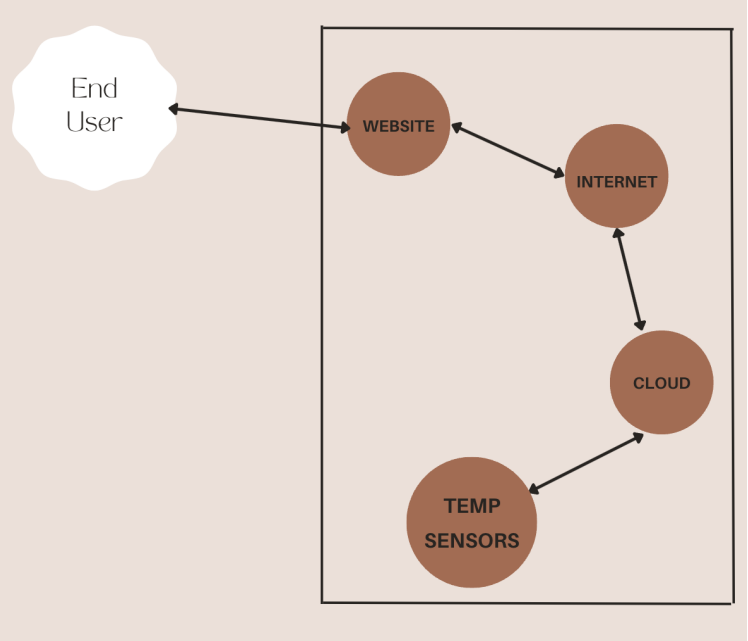
B.1 Logical Architecture



Very similar to the example 4+1 article, the ESP32 establishes communications betweens terminals. Said terminals may be a phone line, a data line, etc.; these different lines are supported by different interface cards. Their (line controller object) responsibility is to decode

and inject all the signals on the line interface card translating card-specific signals to and from a small, uniform set of events: in our case the temperature. The responsibility of the terminal object is to maintain the state of a terminal, and negotiate services on behalf of that line. For example, it uses the services of the numbering plan to interpret the dialing in the selection phase. The conversation represents a set of terminals engaged in a conversation. The conversation uses translation services (location name, time, units), and connection services to establish a path between terminals.

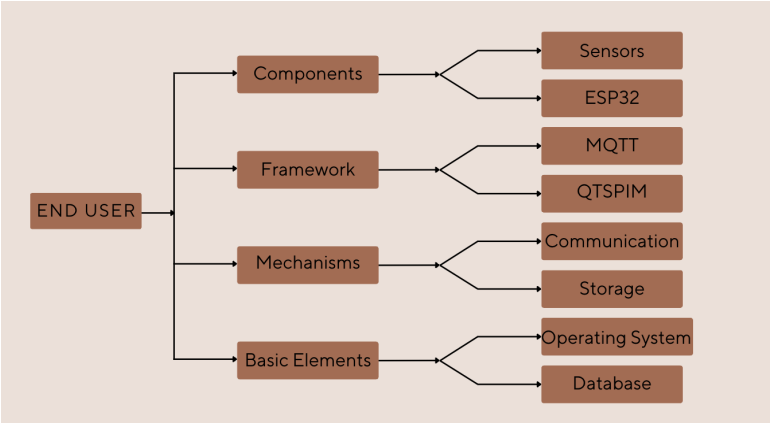
B.2 Process Architecture



This process architecture can be viewed as a set of independently executing logical networks of communicating programs (processes), distributed across a set of hardware resources connected by a LAN or a WAN. These processes represent the level at which the architecture can be tactically controlled. It is partitioned into a set of independent tasks, which are a separate thread of control. All terminals are handled by a single terminal process, which is driven by messages in its input queues. The controller objects are executed on one of four tasks that composes the controller process: the temperature sensor detects any change of state and passes that data to the cloud where it is stored. Connected to the internet, the end user can access a website that allows them to get the data from the cloud that interprets the temp sensors data and communicates

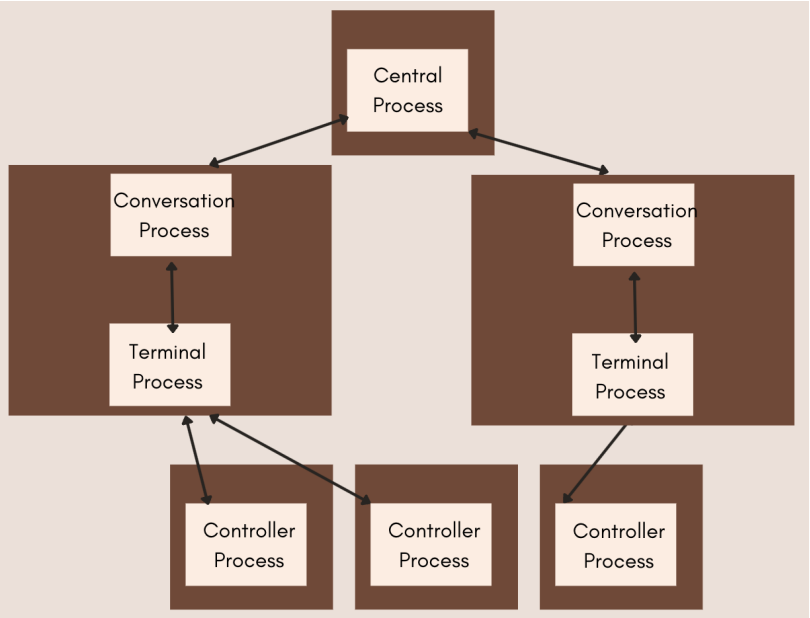
with them by message to the corresponding terminal. Here message passing within the controller process is done via shared memory (cloud).

B.3 Development Architecture



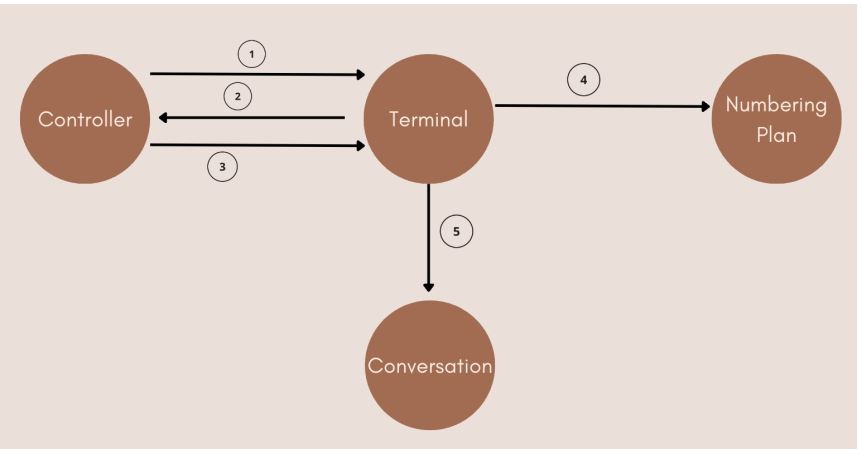
The development architecture focuses on the actual software module organization of the software development environment. The software is packaged in small subsystems that can be developed by one or more developers. The subsystems are organized in a hierarchy of layers, where each layer provides a narrow interface to the layers above it. Also, the development architecture of the system is explained or represented by module and subsystem diagrams between export and import relationships. The rules that govern the development architecture can be listed as: partitioning, grouping and visibility. The development architecture refers to internal requirements related to the ease of development, software management, commonality and constraints imposed by the toolset or programming language. While we develop, the view serves as the basis for requirements allocation, for team organization, monitoring progress, security, for each category so that there is better work and even cost evaluation. On the other hand, for the development view it is very important to use a layered style. Then, there is a rule with the design where a subsystem can only depend on subsystems that are in the same layer or in layers below not above. This layered style minimizes the development of complex networks with dependencies of other modules.

B.4 Physical Architecture



The physical architecture focuses first on the non-functional requirements of the system such as availability, reliability and performance. The software works on a network of computers or processing nodes; elements such as networks, processes, tasks and objects have to be mapped in various nodes. Additionally, we have different physical configurations as: development and testing, deployment of the system for different sites or customers. The mapping of the software to the nodes needs to be flexible and have minimal impact on the source code.

B.5 Scenarios



The elements in the four views work together seamlessly by the use of a small set of important scenarios for which we describe the corresponding sequences of interactions between objects, and between processes. The scenarios are in some sense an abstraction of the most important requirements. Their design is expressed using object

scenario diagrams and object interaction diagrams. This view is redundant with the other ones (hence the "+1"), but it serves two main purposes: as a driver to discover the architectural elements during the architecture design as we will describe later, as a validation and illustration role after this architecture design is complete, both on paper and as the starting point for the tests of an architectural prototype. The notation is very similar to the Logical view for the components but uses the connectors of the Process view for interactions between objects. In this scenario:

1. The controller of the end user's phone is responsible for decoding and injecting all the signals on the line interface, it detects and validates the transition from temperature changes and sends a message to wake up the corresponding terminal object.
2. The terminal allocates some resources, it maintains the state of a terminal and negotiates services on behalf of that line.
3. The controller receives digits (temperatures) and transmits them to the terminal.
4. The terminal uses the numbering plan to analyze the digit flowing.
5. When a valid sequence of digits has been entered, the terminal opens a conversation using translation services where we can get information on location names, time and unit of temperatures.

SECTION C. REFERENCES

1. "Architectural blueprints the 4+1 view model of software architecture." [Online]. Available: https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf. [Accessed: 24-Mar-2023].
2. D. Garlan & M. Shaw, "An Introduction to Software Architecture," Advances in Software Engineering and Knowledge Engineering, Vol. 1, World Scientific Publishing Co. (1993).
3. D. E. Perry & A. L. Wolf, "Foundations for the Study of Software Architecture," ACM Software Engineering Notes, 17, 4, October 1992, 40-52.
4. Ph. Kruchten & Ch. Thompson, "An Object-Oriented, Distributed Architecture for Large Scale Ada Systems," Proceedings of the TRI-Ada '94 Conference, Baltimore, November 6-11, 1994, ACM, p.262-271.

5. G. Booch: Object-Oriented Analysis and Design with Applications, 2nd. edition, Benjamin-Cummings Pub. Co., Redwood City, California, 1993, 589p.
6. K. P. Birman, and R. Van Renesse, Reliable Distributed Computing with the Isis Toolkit, IEEE Computer Society Press, Los Alamitos CA, 1994.
7. K. Rubin & A. Goldberg, “Object Behavior Analysis,” CACM, 35, 9 (Sept. 1992) 48-62
8. B. I. Witt, F. T. Baker and E. W. Merritt, Software Architecture and Design—Principles, Models, and Methods, Van Nostrand Reinhold, New-York (1994) 324p.
9. D. Garlan (ed.), Proceedings of the First Internal Workshop on Architectures for Software Systems, CMU-CS-TR-95-151, CMU, Pittsburgh, 1995