CACiC 2023
CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACIÓN

Caso de estudio: metodología para el diseño y desarrollo de sistemas embebidos distribuidos

Ing. Luis Orlando Ventre[1], Dr. Ing. Orlando Micolini[1], Ing. Mauricio Ludemann[1], Agustín Carranza[1], David D'Andrea[1], Enzo Candotti[1]

[1] Laboratorio de Arquitectura de Computadoras,
FCEFyN-Universidad Nacional de Córdoba
Av. Vélez Sarsfield 1601, CP-5000, Córdoba, Argentina
{luis.ventre, orlando.micolini, mauri.ludemann}@unc.edu.ar

Abstract. En este proyecto se introduce una metodología que se aplica al diseñoy desarrollo de un sistema de control de acceso embebido y distribuido. Esta metodología facilita desacoplar la lógica, la política de resolución de conflictos y las acciones, lo que da lugar a un sistema modular, simple, mantenible, formaly flexible. Además, se logra la verificación formal de la lógica en todas las fasesdel desarrollo. Para modelar la lógica del sistema, se emplean redes de Petri y seconvierten en código ejecutable mediante la ecuación de estado generalizada. Esta solución consigue conservar las propiedades verificadas mediante el uso del formalismo matemático. La implementación incluye un monitor de concurrenciaque integra los diversos componentes de software y hardware del sistema. Asimismo, se establecen interfaces definidas entre los dispositivos e incorpora librerías y protocolos estándares. Además, se destacan los beneficios de aplicar la metodología propuesta al diseño de sistemas críticos y reactivos. Se evidencia su facilidad para abordar problemas complejos, garantizando la escalabilidad y la fiabilidad del sistema desarrollado.

Keywords: Metodología, diseño, sistemas embebidos, Redes de Petri, sistemasdistribuidos

1. Introducción

Un sistema distribuido es una colección de sistemas con capacidad de computó independientes interconectados en red que aparentan ser un único sistema coherente para sus usuarios. El objetivo principal es facilitar el acceso a recursos, y compartirlos de una manera eficiente y controlada. Son transparentes, ocultan los procesos y recursos físicamente distribuidos. Además, son *escalables*, respecto de su tamaño, su distribución geográfica y la capacidad de mantener su administración controlada [1].

En el desarrollo de sistemas embebidos y distribuidos se utilizan técnicas de co-diseño de hardware software y lenguajes de programación como C, C++, Java, VHDL y Verilog. Sin embargo, la codificación manual tiene desventajas dada la propensión a errores, la limitada mantenibilidad y escalabilidad, la alta volatilidad en los requerimientos y los elevados recursos necesarios para la validación y verificación a través de simulaciones y testing. En los sistemas embebidos complejos, se requiere de un proceso de codificación iterativo, para la corrección de errores o cambio en requerimientos, así como la validación y testing de prototipos; dicha metodología solo

es capaz de mitigar la ausencia de errores. Estas tareas impactan significativamente en el tiempo involucrado para el desarrollo del sistema [2].

Para el desarrollo de este trabajo se utiliza diseño basado en modelos [3]. Es un método visual y/o matemático para comprender y resolver diseños asociados con sistemas embebidos complicados. Esta metodología engloba otros conceptos importantes como "arquitectura dirigida por modelos" [4]. El diseño basado en modelos utiliza éstos para dar soporte a otras etapas del desarrollo como por ejemplo la simulación, validación, verificación e implementación.

Las redes de Petri (RdP) [5], son un formalismo de modelado que soporta el desarrollo de sistemas basado en modelos. Este formalismo gráfico-matemático soporta explícitamente y facilita el modelado de: concurrencia, conflictos, recursos compartidos, exclusión mutua y sincronización. Es importante notar que las RdP tienen claramente definida su semántica de ejecución, y su representación matemática, soportando rigurosa documentación, simulación, verificación y traducción a código de ejecución [6]. Además, las RdP son un formalismo matemático abstracto por lo que son intrínsecamente independientes de la plataforma logrando así flexibilidad para alcanzar diversos objetivos de performance, costos, consumo de energía entre otros.

En este informe se presenta un caso de estudio donde se usa una metodología, que posibilita la automatización de la codificación del control de una planta modelada con RdP no autónomas. Esta metodología evita los inconvenientes asociados a la codificación manual de la lógica, y reduce significativamente el tiempo requerido para las etapas de simulación y prototipos. Además, garantiza la validación y verificación matemática del modelo lógico, lo que implica que la especificación documenta fielmente la implementación real.

2. Desarrollo

Para el desarrollo de este sistema distribuido resulta fundamental definir interfaces e implementar protocolos estándares para los módulos de software. Esto, sumado al formalismo matemático de las RdP, colaboran en la obtención de un diseño compacto, robusto y seguro.

2.1 Arquitectura de la Solución

El principal objetivo de la metodología propuesta consiste en descomponer y desacoplar el sistema en: eventos o estímulos, estados, lógica, política y acciones; como se muestra en la Fig. 1.

El punto de partida del diseño es el bloque monitor de concurrencia etiquetado en la Fig. 1 como monitor, el cual será responsable de la gestión de los eventos en sección crítica, sincronización y resolución de conflictos con el fin de determinar cuál acción ejecutar y cuando. A continuación, se propone establecer la lógica del sistema a partir de la realización de un modelo del sistema con una RdP, el monitor a través de la RdP determinará las acciones posibles de ejecución, como se muestra en la Fig. 1.

El sistema requiere de una política para la resolución de los conflictos de la RdP, este componente es utilizado para decidir entre las posibles acciones ejecutables. Además, es necesario un módulo para el manejo de los eventos y colas para almacenarlos (ver Fig. 1). Al descomponer el sistema reactivo (RS) o guiado por eventos (EDA) en estos componentes, se busca mejorar la gestión y el rendimiento delsistema embebido logrando simplificar su diseño, desacoplar los componentes y gestionar su control y ejecución en forma centralizada.

Los componentes agrupados en el área de puntos rojo (Fig. 1) corresponden a un requerimiento de cómputo y memoria importante (base de dato, lógica del sistema, comunicaciones, etc.) por lo que se ha adoptado una placa Raspberry PI para su implementación. En cambio, los agrupados en líneas de punto azul (Fig. 1) corresponden a componentes que realizan acciones como: abrir puertas, tomar imágenes, etc. Estos han sido implementados en una Espressif ESP-32, dada sucapacidad para manejar puertos de entrada salida, comunicaciones y bajo consumo. La arquitectura de bajo nivel resultante de la aplicación puede observarse en la Fig. 1, también se encuentra información detallada al respecto de su implementación en [7].

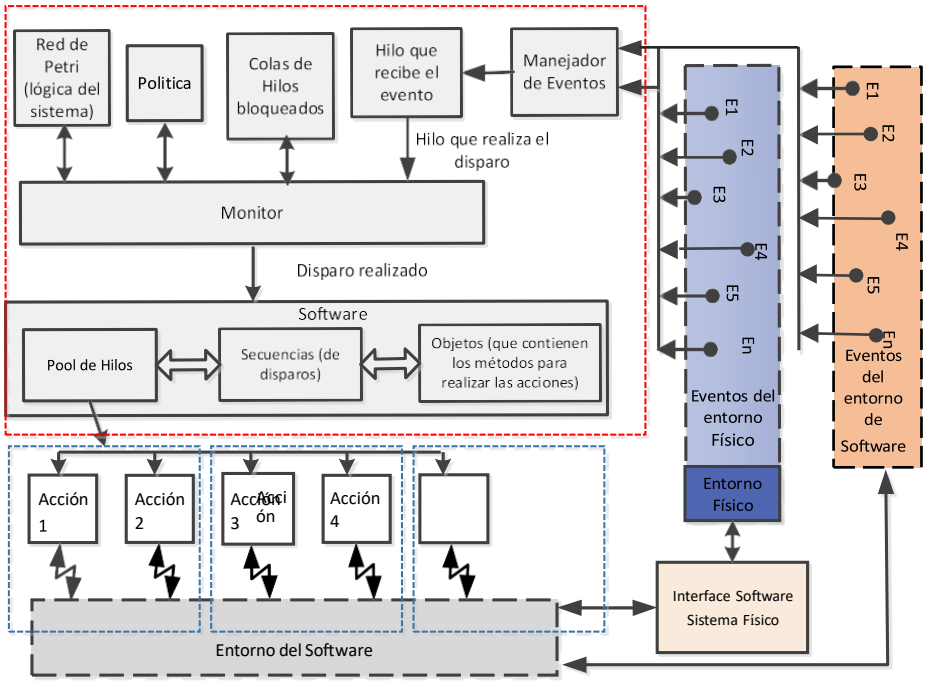


Fig. 1 Componentes de la arquitectura de bajo nivel

2.2 Diseño de componentes

En el diseño de la arquitectura se optó por la orientada a servicios por tratarse de una plataforma de recursos limitados como Raspberry Pi, principalmente en uso de memoria.

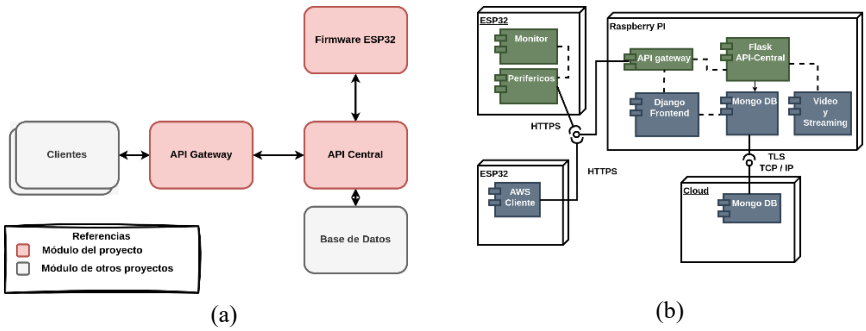
Para las comunicaciones, en una primera instancia, se usó el protocolo MQTT como mecanismo de comunicación entre los dispositivos ESP32-CAM y la Raspberry PI, por sus ventajas como sencillez y ligereza. Luego se descartó en favor de HTTP, ya que MQTT en Python sólo soporta comunicación asíncrona, lo que agrega complejidad y sobrecarga al diseño.

En general, Espressif [8] proporciona una placa de kit de desarrollo ESP32 que podemos usar directamente, este framework cuenta con una implementación de la librería pthreads. Esto se usó oportunamente para el desarrollo del monitor mediante diseños ya probados, los detalles de esta implementación se encuentran en [9]. La política para despertar los hilos de cada cola se delegó en el planificador de FreeRTOS.

También el framework provee implementaciones de clientes y servidores HTTP que fueron utilizados para construir la funcionalidad de cada endpoint.

En la Fig. 2 (a) se describe la arquitectura a nivel de módulos. El módulo API-Gateway sirve de punto de entrada para las consultas de los clientes, funciona como proxy reverso y centraliza la configuración de autenticación. Como parte de los mecanismos de seguridad, pese a que se implementan protocolos seguros en la REST API de los ESP32-CAM, los clientes nunca tienen acceso a ella, sino a través de la API central.

El diseño cuenta con una base de datos no relacional MongoDB [10] que se eligió debido al bajo acoplamiento de las entidades, la transparencia entre los diccionarios de Python y los documentos, sumado a la buena performance para almacenar archivos pequeños (imágenes).



(a) (b)

Fig. 2 Componentes de la arquitectura de alto nivel y diagrama de despliegue

Como parte del desacoplamiento y modularización de componentes, aquellos que se ejecutan en la plataforma Linux de Raspberry Pi fueron diseñados para ejecutarse en contenedores de Docker [11]. Con archivos de tipo 'Dockerfile' y 'docker-compose' se consigue que el sistema sea replicable, escalable y tolerante a fallos.

2.3 Topología de red

La Fig. 3 ilustra la topología de la red para el proyecto. Se divide en dos dominios: uno para el acceso de los usuarios a los clientes (web o REST) y otro para los dispositivos ESP32-CAM. La API central participa en ambos dominios gestionando las consultas. De esa manera se logró aislar el acceso a las interfaces expuestas por ellas y, a su vez, aumentar el alcance de la red inalámbrica.

La seguridad en los puntos de acceso, tanto en la API del nodo central como en cada dispositivo ESP32 consistió en la implementación del protocolo de Basic-Auth combinado con HTTPS [12] activando la verificación de certificados PKI.

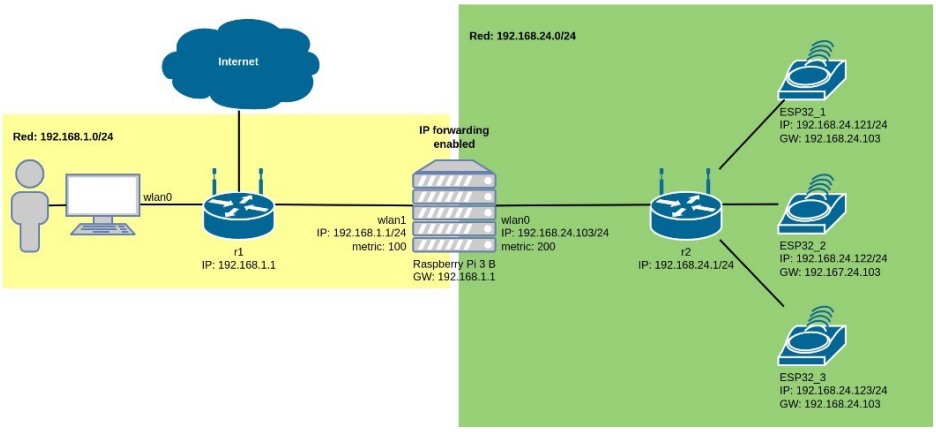


Fig. 3. Diseño topológico de las redes del sistema

2.4 Implementación y Selección de Componentes

A través de la utilización del *método iterativo,* se realizó el análisis, diseño, codificación y prueba de los módulos de software y la elección de componentes de hardware. Esto permitió obtener módulos funcionales al final de cada etapa, sirviendo los mismos como base para las siguientes iteraciones. La última de ellas ocurre cuando los requerimientos, repartidos estratégicamente, se cumplen.

Durante el desarrollo del sistema, luego de definir la arquitectura de bajo nivel, se procedió a realizar la selección de componentes de hardware. Para la implementación del hardware central se tuvieron en cuenta las siguientes opciones: Raspberry Pi, Orange Pi, Nvidia Jetson, Odroid, BeagleBone y Arduino. Luego de analizar las alternativas y con el criterio de menor costo, simplicidad de uso, soporte, y compatibilidad con los restantes módulos se seleccionó Raspberry Pi.

Para implementar el sistema de punto de acceso (puerta), se consideró la disponibilidad en el Laboratorio de Arq. de Computadoras de la F.C.E.F.y N. U.N.C. de la placa ESP32-CAM, que presenta diversas ventajas como su bajo costo, gran cantidad de puertos de entrada/salida, la disponibilidad de puertos genéricos y de vídeo. En cuanto a la elección del lenguaje de programación, se optó por Python debido a que es un lenguaje de alto nivel que posibilita un desarrollo rápido y eficiente. Aunquesi bien es cierto que su desempeño es más bajo comparado con otros lenguajes como

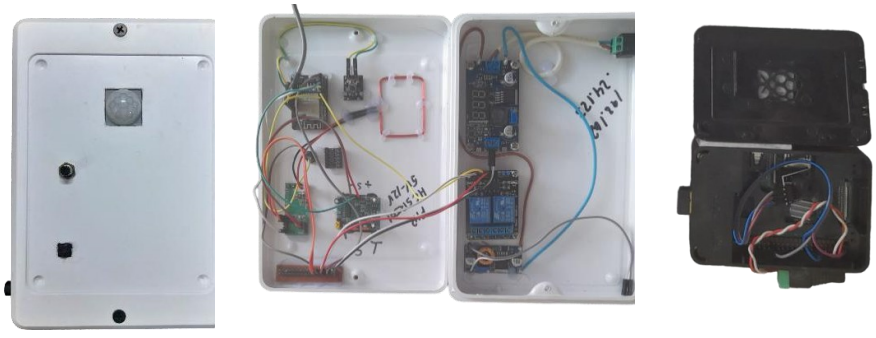
Java, C++ o Go, en este caso la velocidad no es la prioridad principal.

En lo referente a la selección del framework para el desarrollo de la API REST, se evaluaron las dos opciones más usadas para el desarrollo de aplicaciones web en Python, que son: Flask y Django. Para esta implementación se decidió utilizar Flask debido a que ofrece mayor flexibilidad y una curva de aprendizaje sencilla en comparación con Django. Para la selección del IDE/Framework luego de analizar las propiedades de Arduino, Espressif, Mongoose SO, Simba, Pumbaa y nanoFramework se optó por Espressif dado que implementa de forma nativa una versión de FreeRTOS compatible con multicore.

Para el periférico de acceso, se seleccionó RFID 125 KHz debido a su alta disponibilidad, lo que simplifica tanto su adquisición como reemplazo en caso de ser necesario.

En cuanto a fuente de alimentación DC-DC se optó por un modelo que implementa un voltímetro en la salida y la entrada con rango de tensión 1.23 – 35 V y 3A con un integrado LM2596. Los restantes periféricos fueron seleccionados de acuerdo con su disponibilidad y con criterio de menor costo con el objetivo de obtener un sistema final accesible y replicable.

La arquitectura del sistema de control de acceso distribuido se observa en la Fig. 4 (b), y fue orientada a servicios con protocolo http. La implementación del sistema para los puntos de acceso (puerta) se realizó en una carcasa genérica en donde se montaron los diferentes componentes.



(a) (b) (c)

Fig. 4. Vistas externa e interna del dispositivo de acceso e interna del dispositivo central.

El dispositivo resultante, en su vista exterior puede observarse en la Fig. 4 (a): en la parte superior el sensor de movimiento (HC-SR501), en el centro desplazado a

izquierda el pulsador y en la parte inferior el orificio por donde el módulo ESP32-CAM captura las imágenes.

En la Fig. 4 (b) se observa el interior de este dispositivo, conformado por: ficha de alimentación, fuente step-down DC-DC LM-2596 con voltímetro, conversor de niveles 5V - 3V3, placa ESP32-CAM con cámara OV2640 2Mp, módulo lector RFID RDM6300 y antena, módulo sensor de movimiento, interruptor, fichas y cables dupont. En la Fig. 4 (c) se puede observar la carcasa para el sistema central implementado con Placa SBC Raspberry PI 3b 1GB RAM 16GB SD y módulo buzzer pasivo.

El repositorio del proyecto se encuentra en https://github.com/orgs/Proyecto-Integrador-FCEFYN/dashboard.

3. Resultados

El presente trabajo expone como resultado un caso de estudio y aplicación, así como también la validación, de una metodología para diseñar y desarrollar sistemas embebidos, críticos, RS y EDA. Esta metodología logró gestionar eficazmente la complejidad del diseño, el testing y la codificación. Se minimizaron y mitigaron los riesgos desde el inicio.

Se obtuvo un sistema de control de acceso distribuido, capaz de permitir o restringir el acceso a zonas determinadas según parámetros de seguridad establecidos. Los usuarios son identificados mediante tarjetas o llaveros y se cuenta con datos personales almacenados al momento del registro. Además, el sistema cuenta con funcionalidades de seguridad adicionales tales como: restricciones horarias configurables, sensores de movimiento para la captura de imágenes ante situaciones inesperadas, grabación, almacenamiento y *streaming* de video, entre otras.

Entre las características del sistema se puede mencionar que es *distribuido* dado que esta implementado y soportado por múltiples dispositivos e interconectado mediante una red, *flexible* por ser apto para adaptarse a cambios en sus requerimientos minimizando su impacto por ser completamente modular, su lógica ha sido validada *formalmente* debido al fundamento matemático de las RdP. Es *mantenible* ya que es modificable efectiva y eficientemente debido a su modularidad, según necesidades evolutivas, correctivas o perfectivas y es *simple* por permitir visualizar las acciones, independientemente de la lógica y la política que conducen al sistema. Por lo tanto, se logra un código claro y sin responsabilidades solapadas.

Referencias

1. Tanenbaum, A.S. and M. Van Steen, Distributed systems: principles and paradigms. 2023: Prentice-Hall.
2. Hobbs, C., Embedded Software Development for Safety-Critical Systems. 2015: CRC Press.
3. Weilkiens, T., et al., Model-Based System Architecture. 2015: John Wiley & Sons.

4. Roberts, C.J., et al., Preliminary Results from a Model-Driven Architecture Methodology for Development of an Event-Driven Space Communications Service Concept. 2017.
5. Murata, T., Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, 1989: p. Vol. 77, No. 4, pp. 541-580.
6. Ventre, L.O. and O. Micolini. Extended Petri Net Processor and Threads Quantity Determination Algorithm for Embedded Systems. in Argentine Congress of Computer Science. 2020. Springer.
7. Ventre, L.O. and O. Micolini. Algoritmos para determinar cantidad y responsabilidad de hilos en sistemas embebidos modelados con Redes de Petri S3PR. in XXVII Congreso Argentino de Ciencias de la Computación (CACIC)(Modalidad virtual, 4 AL 8 DE OCTUBRE DE 2021.). 2022.
8. Sharp, A. and Y. Vagapov, Comparative analysis and practical implementation of the ESP32 microcontroller module for the Internet of Things. 2022.
9. Melgarejo, M.ı.d.M.G. Comunicacion y Sincronizacion con Monitores Resumen del Tema. 2002 [cited 2023 Mayo]; Available from: http://www.lcc.uma.es/~gallardo/temaCLASE.pdf.
10. Sharma, M., Full Stack Development with MongoDB: Covers Backend, Frontend, APIs, and Mobile App Development Using PHP, NodeJS, ExpressJS, Python and React Native. 2022: BPB Publications.
11. Schenker, G.N., Learn Docker–Fundamentals of Docker 19. x: Build, test, ship, and run containers with Docker and Kubernetes. 2020: Packt Publishing Ltd.
12. mozilla.org. Autenticación HTTP. 2023 [cited 2023 Mayo]; Available from: https://developer.mozilla.org/es/docs/Web/HTTP/Authentication.