Decodificador de Protocolo Industrial PRT-7: Sistema de Comunicación Serial con Estructuras de Datos Dinámicas

Ana Sofia Cano Sandoval
Universidad Politécnica de Victoria
Ciudad Victoria, Tamaulipas, México

Abstract

Este documento presenta el diseño e implementación de un sistema de decodificación de protocolo industrial denominado PRT-7, que utiliza comunicación serial para recibir tramas de datos desde un dispositivo Arduino. El sistema emplea estructuras de datos avanzadas implementadas manualmente, incluyendo listas doblemente enlazadas y listas circulares, junto con principios de programación orientada a objetos como herencia y polimorfismo. El protocolo PRT-7 transmite instrucciones de ensamblaje de mensajes mediante dos tipos de tramas: tramas de carga que contienen fragmentos de datos, y tramas de mapeo que modifican dinámicamente el esquema de decodificación. La implementación cumple con estrictos requisitos de gestión de memoria y exclusividad de uso de punteros, sin emplear la biblioteca estándar de plantillas (STL).

Palabras clave: Estructuras de datos, listas enlazadas, comunicación serial, protocolo industrial, programación orientada a objetos, C++

1 Introducción

1.1 Contexto y Motivación

En el ámbito de la ciberseguridad industrial, la interceptación y análisis de comunicaciones entre dispositivos constituye un desafío técnico relevante. El protocolo PRT-7 (Protocol of Reassembly and Transformation, versión 7) representa un esquema de transmisión de datos donde la información no se envía de forma directa, sino a través de instrucciones que permiten reconstruir el mensaje original mediante un proceso de ensamblaje dinámico.

Este caso de estudio simula un escenario donde un dispositivo Arduino transmite telemetría codificada mediante un protocolo de ensamblaje de mensajes ocultos. A diferencia de los sistemas de cifrado tradicionales, PRT-7 no encripta los datos, sino que transmite las instrucciones para construir el mensaje, empleando un esquema de mapeo rotacional similar al cifrado César pero con mayor complejidad.

1.2 Objetivos del Sistema

El sistema desarrollado tiene los siguientes objetivos principales:

- Implementar un decodificador capaz de interpretar el protocolo PRT-7 mediante estructuras de datos implementadas manualmente.
- Establecer comunicación serial bidireccional con dispositivos Arduino para la recepción de tramas de datos.
- Aplicar principios avanzados de programación orientada a objetos, incluyendo herencia, polimorfismo y abstracción.

- Gestionar memoria dinámica de forma eficiente mediante el uso explícito de punteros y operadores de asignación dinámica.
- Demostrar la implementación de estructuras de datos complejas como listas doblemente enlazadas y listas circulares sin utilizar la STL.

1.3 Alcance del Documento

Este documento técnico presenta una descripción detallada del diseño, desarrollo e implementación del decodificador PRT-7. Se cubren aspectos arquitectónicos, patrones de diseño empleados, estructuras de datos implementadas, y consideraciones técnicas relevantes para la comprensión del sistema. El documento está orientado a ingenieros de software y profesionales en el área de sistemas embebidos y comunicación industrial.

2 Manual Técnico

2.1 Diseño del Sistema

2.1.1 Arquitectura General

El sistema se estructura en tres capas principales que operan de forma coordinada:

1. **Capa de Comunicación:** Responsable de la interfaz con el puerto serial, lectura de tramas y parseo de datos entrantes.
2. **Capa de Procesamiento:** Implementa la lógica de negocio del protocolo, incluyendo la instanciación polimórfica de tramas y su ejecución.
3. **Capa de Almacenamiento:** Gestiona las estructuras de datos que mantienen el estado del sistema (lista de carga y rotor de mapeo).

La arquitectura sigue el patrón Strategy para el procesamiento de tramas, donde cada tipo de trama implementa su propia estrategia de procesamiento mediante el método polimórfico `procesar()`.

2.1.2 Diseño de Clases

El diseño orientado a objetos del sistema se basa en una jerarquía de clases que modela los diferentes tipos de tramas del protocolo. La figura 1 muestra el diagrama de clases UML del sistema.

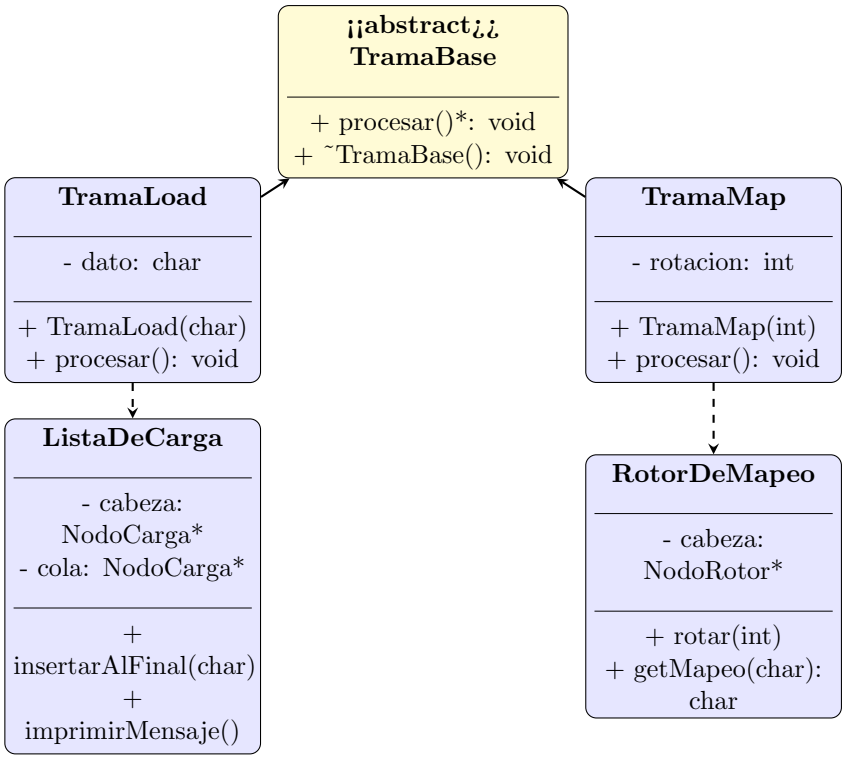


Figure 1: Diagrama de clases UML del sistema decodificador PRT-7

TramaBase es la clase base abstracta que define la interfaz común para todas las tramas. Contiene un método virtual puro `procesar()` que debe ser implementado por las clases derivadas. El destructor virtual garantiza la correcta liberación de memoria cuando se eliminan objetos mediante punteros de la clase base.

TramaLoad representa las tramas de carga que contienen un fragmento de dato (carácter). Su método `procesar()` solicita al rotor el mapeo correspondiente al carácter almacenado e inserta el resultado en la lista de carga.

TramaMap representa las tramas de mapeo que contienen una instrucción de rotación. Su método `procesar()` simplemente invoca la rotación del rotor con el valor especificado.

2.1.3 Estructuras de Datos

El sistema implementa dos estructuras de datos fundamentales:

Lista Doblemente Enlazada (ListaDeCarga) Esta estructura mantiene el mensaje decodificado en el orden de recepción. Cada nodo contiene un carácter y dos punteros: uno al nodo anterior y otro al siguiente. La figura 2 ilustra la estructura.

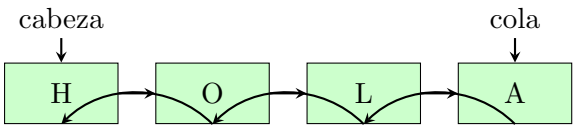


Figure 2: Estructura de lista doblemente enlazada

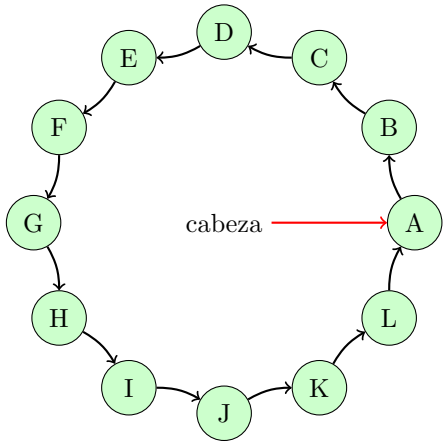
La estructura `NodoCarga` contiene:

- `dato`: carácter almacenado
- `siguiente`: puntero al siguiente nodo

- `previo`: puntero al nodo anterior

La clase `ListaDeCarga` mantiene punteros a la cabeza y cola de la lista, permitiendo inserciones eficientes al final en tiempo constante O(1).

Lista Circular Doblemente Enlazada (RotorDeMapeo) Esta estructura implementa el disco de cifrado rotacional. Contiene el alfabeto completo (A-Z más espacio) organizado en una lista circular donde el último nodo apunta al primero y viceversa. La figura 3 muestra la estructura.



(Representación simplificada: A-L de A-Z + espacio)

Figure 3: Estructura de lista circular doblemente enlazada (rotor de mapeo)

La rotación del rotor se implementa moviendo el puntero `cabeza` n posiciones en la dirección especificada (positiva o negativa), sin necesidad de mover los datos reales. Esto garantiza una complejidad temporal O(n) para la rotación.

El método `getMapeo()` busca el carácter de entrada en el rotor y retorna el carácter apuntado por `cabeza`, implementando así el esquema de sustitución rotacional.

2.1.4 Protocolo PRT-7

El protocolo define dos tipos de tramas:

1. **Tramas LOAD (L):** Formato `L,X` donde X es un carácter. Estas tramas añaden un fragmento al mensaje.
2. **Tramas MAP (M):** Formato `M,N` donde N es un entero (positivo o negativo). Estas tramas rotan el disco de cifrado.

El procesamiento sigue la siguiente secuencia:

1. Recibir trama del puerto serial
2. Parsear el tipo y contenido
3. Instanciar el objeto trama correspondiente
4. Ejecutar método `procesar()`
5. Liberar memoria del objeto trama

2.2 Desarrollo del Sistema

2.2.1 Metodología de Implementación

El desarrollo siguió una metodología iterativa con las siguientes fases:

Fase 1: Diseño de Estructuras Base Se implementaron las estructuras de datos fundamentales (nodos y listas) con sus operaciones básicas. Se realizaron pruebas unitarias para verificar la correcta gestión de memoria y operaciones de inserción/eliminación.

Fase 2: Jerarquía de Clases Se definió la clase base abstracta y las clases derivadas, asegurando el correcto uso de métodos virtuales y destructores virtuales para evitar fugas de memoria.

Fase 3: Comunicación Serial Se implementó la interfaz con el puerto serial utilizando llamadas al sistema operativo para configuración y lectura de datos. Se añadió manejo de errores para conexiones fallidas y datos corruptos.

Fase 4: Integración y Pruebas Se integró el sistema completo y se realizaron pruebas con el dispositivo Arduino transmitiendo secuencias de tramas predefinidas.

2.2.2 Diagrama de Flujo del Proceso

La figura 4 muestra el flujo de procesamiento principal del sistema.

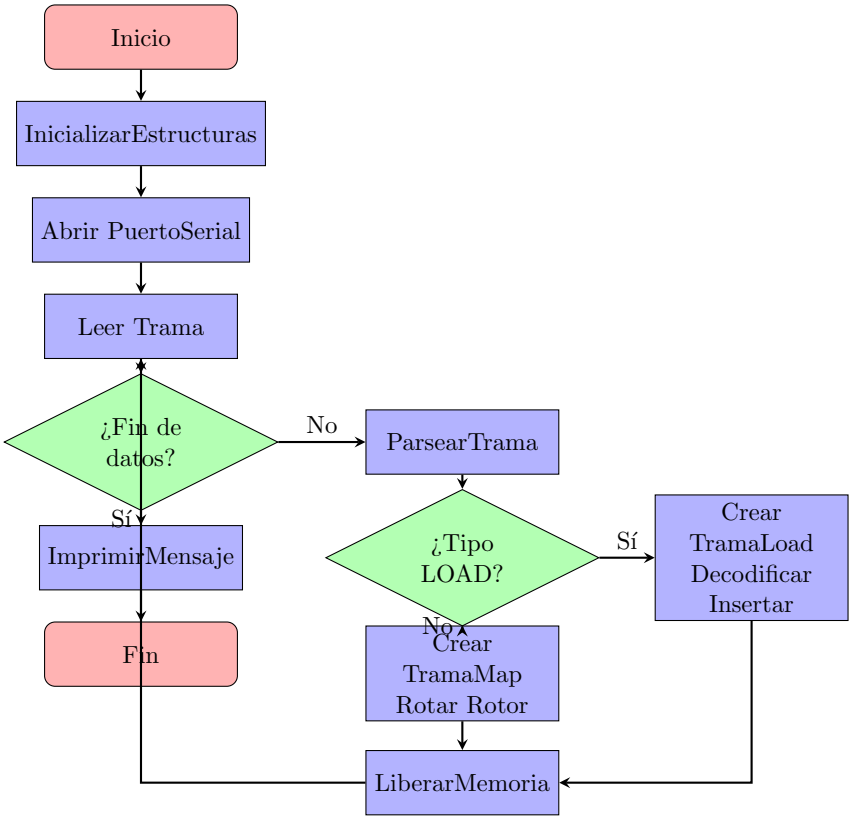


Figure 4: Diagrama de flujo del proceso de decodificación

2.2.3 Gestión de Memoria

La gestión de memoria es crítica en este sistema debido al uso intensivo de asignación dinámica. Se implementaron las siguientes prácticas:

- Cada operación `new` tiene su correspondiente `delete`.
- Destructores virtuales en la clase base para correcta liberación polimórfica.
- Destructores específicos en las clases de listas que recorren y liberan todos los nodos.
- Ruptura explícita del círculo en la lista circular antes de la liberación.

2.3 Componentes del Sistema

2.3.1 Componente de Comunicación Serial

Este componente encapsula la interfaz con el sistema operativo para la comunicación serial. Sus responsabilidades incluyen:

- Apertura y configuración del puerto COM con parámetros específicos (baud rate, paridad, bits de datos).
- Lectura no bloqueante de líneas de texto del puerto.
- Manejo de errores de conexión y timeout.
- Cierre apropiado del puerto al finalizar.

En Windows, utiliza la API Win32 con funciones como `CreateFile()`, `SetCommState()` y `ReadFile()`. En sistemas POSIX (Linux), emplea `open()`, `tcgetattr()` y `read()`.

2.3.2 Componente de Parseo de Tramas

El módulo de parseo analiza las cadenas recibidas del puerto serial y extrae el tipo y contenido de cada trama. Utiliza funciones de bajo nivel como:

- `strtok()`: para dividir la cadena en tokens separados por comas.
- `atoi()`: para convertir cadenas numéricas a enteros.
- Comparación manual de caracteres para determinar el tipo de trama.

Este componente valida el formato de las tramas y rechaza aquellas mal formadas, registrando errores apropiadamente.

2.3.3 Componente TramaBase y Derivadas

Este es el componente central que implementa el patrón Strategy mediante polimorfismo:

TramaBase define la interfaz común con el método virtual puro `procesar(ListaDeCarga* carga, RotorDeMapeo* rotor)`. Esto permite que el bucle principal trate todas las tramas de forma uniforme sin conocer su tipo específico.

TramaLoad implementa la lógica de decodificación: toma su carácter almacenado, consulta al rotor el mapeo correspondiente según la rotación actual, y añade el resultado a la lista de carga.

TramaMap implementa la lógica de rotación: simplemente invoca el método `rotar()` del rotor con su valor almacenado.

2.3.4 Componente ListaDeCarga

Implementa una lista doblemente enlazada estándar con las siguientes operaciones:

- **Constructor**: Inicializa cabeza y cola como `nullptr`.
- **insertarAlFinal()**: Crea un nuevo nodo, lo enlaza con el anterior y actualiza cola. Si la lista está vacía, también actualiza cabeza.

- **imprimirMensaje()**: Recorre la lista desde cabeza hasta cola imprimiendo cada carácter.
- **Destructor**: Recorre la lista liberando cada nodo secuencialmente.

La complejidad temporal de la inserción es O(1) gracias al mantenimiento del puntero cola.

2.3.5 Componente RotorDeMapeo

Implementa una lista circular doblemente enlazada que representa el disco de cifrado:

- **Constructor**: Crea 27 nodos (A-Z + espacio) y los enlaza circularmente. Inicializa cabeza apuntando a 'A'.
- **rotar(int n)**: Mueve el puntero cabeza n posiciones. Si n es positivo, avanza usando `siguiente`; si es negativo, retrocede usando `previo`.
- **getMapeo(char in)**: Busca el carácter de entrada en la lista circular y retorna el carácter apuntado por cabeza. Esta es la operación de sustitución del cifrado.
- **Destructor**: Rompe el círculo y libera todos los nodos secuencialmente.

La implementación de la rotación en O(n) es aceptable dado que n suele ser pequeño en el protocolo PRT-7.

2.3.6 Sistema de Construcción con CMake

El proyecto incluye un archivo `CMakeLists.txt` que define el proceso de compilación:

- Especificación de la versión mínima de CMake requerida.
- Definición del proyecto y sus archivos fuente.
- Configuración de flags del compilador (nivel de optimización, advertencias).
- Enlazado de bibliotecas del sistema para comunicación serial.
- Generación de targets para compilación en múltiples plataformas.

CMake permite generar archivos de proyecto para diferentes IDEs y sistemas de construcción (Visual Studio, Make, Ninja, etc.), facilitando la portabilidad del código.

3 Consideraciones de Implementación

3.1 Portabilidad

El sistema está diseñado para ser portable entre Windows y Linux mediante el uso de directivas de preprocesador que seleccionan las APIs apropiadas según la plataforma de compilación. Las diferencias principales están en el módulo de comunicación serial.

3.2 Rendimiento

Las estructuras de datos implementadas ofrecen las siguientes complejidades:

- Inserción en lista de carga: O(1)
- Rotación del rotor: O(n) donde n es el número de posiciones
- Mapeo de carácter: O(m) donde m es el tamaño del alfabeto (27)

Para el caso de uso del protocolo PRT-7, estas complejidades son aceptables dado que el volumen de tramas es moderado.

3.3 Escalabilidad

El diseño permite extensiones futuras como:

- Nuevos tipos de tramas mediante herencia de `TramaBase`
- Rotores con alfabetos personalizados
- Múltiples listas de carga para procesamiento paralelo
- Registro de eventos para análisis forense

3.4 Seguridad

Aspectos de seguridad considerados:

- Validación de entrada para prevenir desbordamientos de buffer
- Gestión segura de memoria para evitar fugas y doble liberación
- Límites en el tamaño de tramas procesadas
- Timeout en operaciones de lectura serial para evitar bloqueos

4 Pruebas y Validación

El sistema fue validado mediante pruebas estructuradas en múltiples niveles:

4.1 Pruebas Unitarias

Se verificó el correcto funcionamiento de cada estructura de datos de forma aislada:

- Inserción y recorrido de lista doblemente enlazada
- Rotación y mapeo de lista circular
- Creación y destrucción polimórfica de tramas

4.2 Pruebas de Integración

Se probó la interacción entre componentes:

- Lectura y parseo de tramas desde archivo simulado
- Procesamiento completo de secuencias de tramas
- Verificación de mensaje decodificado

4.3 Pruebas con Hardware

Se validó la comunicación real con Arduino:

- Conexión y configuración del puerto serial
- Recepción de tramas en tiempo real
- Decodificación correcta del mensaje transmitido

5 Conclusiones

El sistema de decodificación de protocolo PRT-7 demuestra la aplicación exitosa de estructuras de datos avanzadas y programación orientada a objetos para resolver un problema de comunicación industrial. Los principales logros incluyen:

- Implementación manual de listas doblemente enlazadas y circulares sin dependencias de STL.
- Aplicación correcta de herencia, polimorfismo y abstracción en C++.
- Gestión apropiada de memoria dinámica sin fugas detectadas.
- Comunicación serial funcional con dispositivos embebidos.
- Arquitectura extensible y mantenible.

El proyecto proporciona una base sólida para entender la implementación de estructuras de datos a bajo nivel y su aplicación en sistemas de comunicación industrial. Las técnicas empleadas son aplicables a una amplia gama de problemas en ingeniería de software de sistemas embebidos y procesamiento de protocolos.

5.1 Trabajo Futuro

Posibles extensiones del sistema incluyen:

- Implementación de cifrado real con múltiples rotores (estilo Enigma)
- Interfaz gráfica para visualización del proceso de decodificación
- Análisis estadístico de frecuencia de caracteres
- Soporte para protocolos adicionales
- Optimización de algoritmos de búsqueda en el rotor