INFORME: IMPLEMENTACIÓN DE CALCULADORA BASADA EN EL PARADIGMA DE AGENTES

1. INTRODUCCIÓN

El presente informe documenta el diseño, implementación y funcionamiento de una calculadora distribuida utilizando el paradigma de agentes. En este sistema, cada operación aritmética es gestionada por un agente autónomo especializado que se comunica con otros agentes para resolver expresiones matemáticas complejas.

1.1 Objetivo del Proyecto

Diseñar e implementar una calculadora funcional donde:

- Cada operación matemática es manejada por un agente independiente
- Los agentes se comunican entre sí mediante un sistema de mensajería
- Se respeta la precedencia de operadores matemáticos
- Se soportan expresiones complejas con paréntesis anidados

1.2 Alcance

El sistema implementado es capaz de:

- Evaluar expresiones aritméticas con operadores básicos (+, -, *, /, **)
- Manejar paréntesis y precedencia de operadores
- Procesar números enteros y decimales (positivos y negativos)
- Registrar toda la comunicación entre agentes
- Generar estadísticas de uso de operaciones

2. FUNDAMENTO TEÓRICO

2.1 Paradigma de Agentes

Un agente es una entidad autónoma que:

- **Percibe** su entorno mediante sensores
- **Actúa** sobre su entorno mediante efectores
- **Toma decisiones** basadas en objetivos
- **Se comunica** con otros agentes

Características clave de los agentes:

- **Autonomía**: Opera sin intervención directa
- **Reactividad**: Responde a cambios en el entorno
- **Pro-actividad**: Toma iniciativa para cumplir objetivos
- **Habilidad social**: Interactúa con otros agentes

2.2 Sistemas Multi-Agente (MAS)

Un Sistema Multi-Agente consiste en:

- Múltiples agentes con capacidades especializadas
- Un mecanismo de comunicación entre agentes
- Protocolos de coordinación y negociación
- Objetivos compartidos o complementarios

Ventajas de MAS:

- Modularidad y mantenibilidad
- Escalabilidad horizontal
- Tolerancia a fallos
- Especialización de tareas

2.3 Precedencia de Operadores Matemáticos

En matemáticas, la precedencia de operadores determina el orden de evaluación:

Nivel 1 (Mayor precedencia): Paréntesis `( )` **Nivel 2**: Potenciación `**` **Nivel 3**: Multiplicación `*` y División `/` (de izquierda a derecha) **Nivel 4**: Suma `+` y Resta `-` (de izquierda a derecha)

Ejemplo: `2 + 3 * 4 ** 2`

1. Primero: `4 ** 2 = 16`
2. Segundo: `3 * 16 = 48`
3. Tercero: `2 + 48 = 50`

2.4 Tokenización de Expresiones

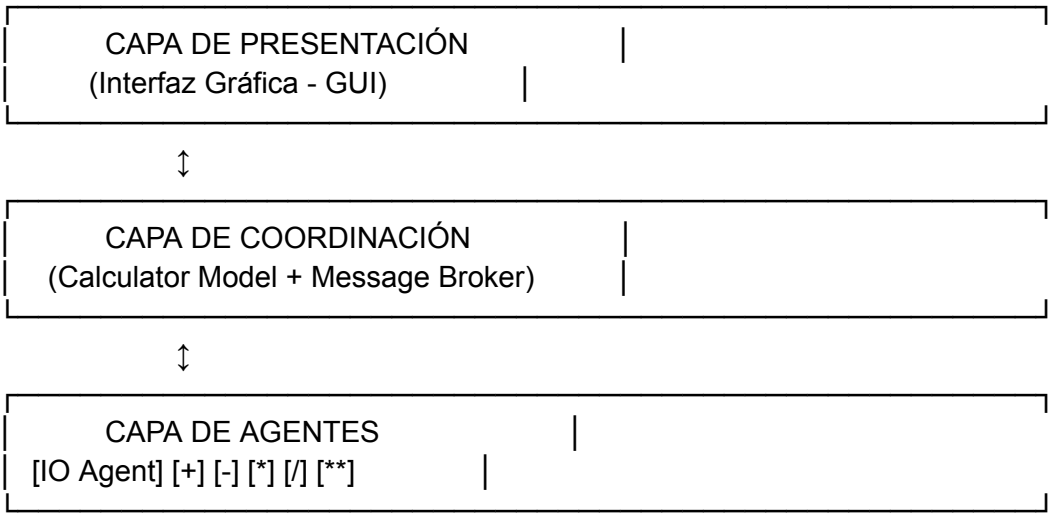
La tokenización es el proceso de dividir una expresión en unidades significativas (tokens):

Expresión: `(2 + 3) * 4` **Tokens**: `['(', '2', '+', '3', ')', '*', '4']`

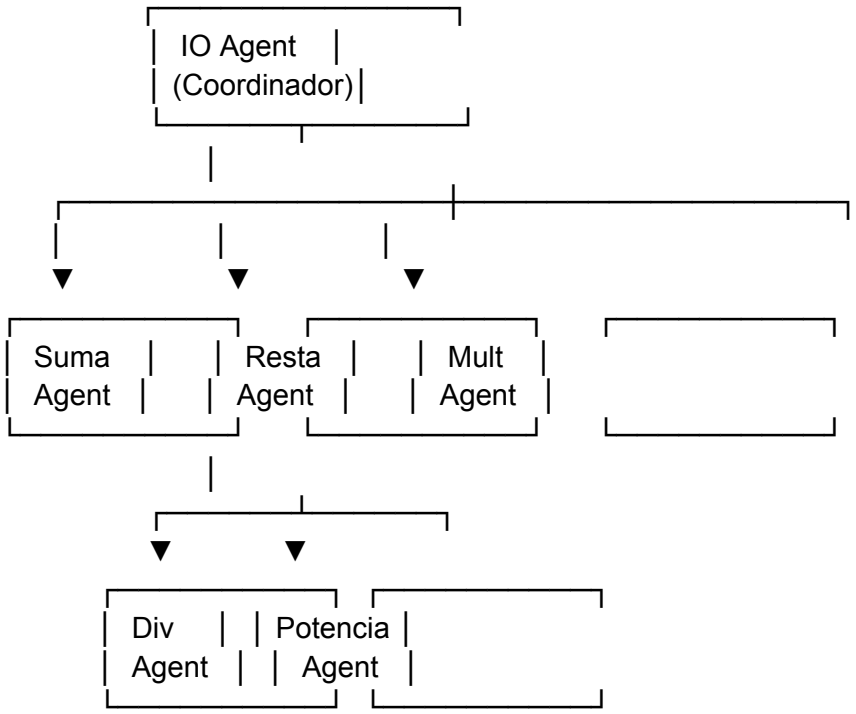
3. DISEÑO DE LA SOLUCIÓN

3.1 Arquitectura del Sistema

El sistema está diseñado con tres capas principales:



3.2 Diagrama de Agentes



3.3 Agentes del Sistema

A. Agente de Entrada/Salida (IO Agent)

Responsabilidades:

- Recibir expresiones del usuario
- Tokenizar y parsear expresiones
- Coordinar la ejecución respetando precedencia
- Manejar paréntesis

- Retornar el resultado final
- Gestionar errores

Algoritmo de procesamiento:

FUNCIÓN procesar_expresión(expresión):
1. Limpiar y validar expresión
2. Tokenizar expresión en lista de tokens
3. Evaluar paréntesis (recursivamente)
4. Evaluar potencias (de izquierda a derecha)
5. Evaluar multiplicaciones y divisiones
6. Evaluar sumas y restas
7. Retornar resultado único

FIN FUNCIÓN

B. Agentes de Operación

Cada agente de operación tiene:

Estructura común:

```
clase AgenteOperación:
   - nombre_operación
   - función_operación
   - símbolo
   - contador_operaciones

   método realizar_operación(operando1, operando2):
      resultado = función_operación(operando1, operando2)
      registrar_log()
      retornar resultado

   método step():
      procesar_mensajes_recibidos()
      enviar_respuestas()
```

Agentes específicos:

1. **Agente Suma (+)**
    - Función: `operando1 + operando2`
    - Complejidad: O(1)
2. **Agente Resta (-)**
    - Función: `operando1 - operando2`
    - Complejidad: O(1)

3. **Agente Multiplicación (*)**
    - Función: `operando1 * operando2`
    - Complejidad: O(1)
4. **Agente División (/)**
    - Función: `operando1 / operando2`
    - Validación: operando2 ≠ 0
    - Complejidad: O(1)
5. **Agente Potencia (**)**
    - Función: `operando1 ** operando2`
    - Complejidad: O(log n) donde n = operando2

3.4 Sistema de Mensajería (Message Broker)

El Message Broker es el componente central de comunicación.

Componentes:

```
MessageBroker:
├── message_queue: Cola de mensajes pendientes
├── message_history: Historial completo de mensajes
└── Métodos:
    ├── send_message(mensaje)
    ├── get_messages_for(agent_id)
    ├── clear_queue()
    └── get_history()
```

Estructura de un Mensaje:

```
Mensaje:
  - sender: ID del agente emisor
  - receiver: ID del agente receptor
  - content: Datos del mensaje
      • operands: (operando1, operando2)
      • result: resultado de operación
  - msg_type: Tipo de mensaje
      • "operation": solicitud de operación
      • "result": respuesta con resultado
  - timestamp: Marca temporal
```

3.5 Flujo de Comunicación

Ejemplo: Evaluar `2 + 3 * 4`

1. Usuario ingresa: "2 + 3 * 4"

2. IO Agent tokeniza: ['2', '+', '3', '*', '4']

3. IO Agent identifica precedencia:
   - Multiplicación primero (nivel 3)
   - Suma después (nivel 4)

4. IO Agent → Agente Multiplicación
   Mensaje: {
      sender: "io_agent",
      receiver: "multiplicacion_agent",
      content: {operands: (3, 4)},
      type: "operation"
   }

5. Agente Multiplicación procesa:
   - Calcula: 3 * 4 = 12
   - Log: "[MULTIPLICACIÓN] 3 * 4 = 12"

6. Agente Multiplicación → IO Agent
   Mensaje: {
      sender: "multiplicacion_agent",
      receiver: "io_agent",
      content: {result: 12},
      type: "result"
   }

7. IO Agent recibe resultado: 12
   - Tokens actualizados: ['2', '+', '12']

8. IO Agent → Agente Suma
   Mensaje: {
      sender: "io_agent",
      receiver: "suma_agent",
      content: {operands: (2, 12)},
      type: "operation"
   }

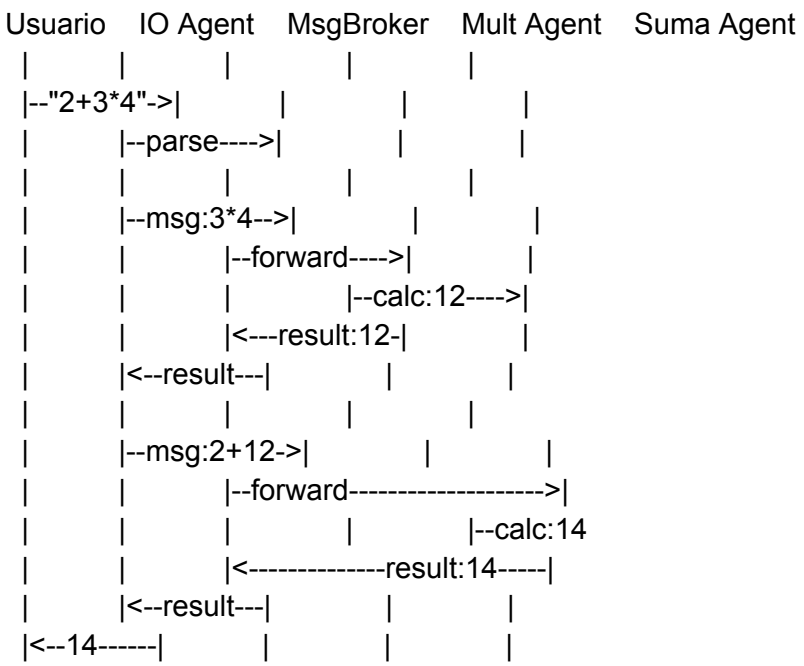
9. Agente Suma procesa:
   - Calcula: 2 + 12 = 14
   - Log: "[SUMA] 2 + 12 = 14"

10. Agente Suma → IO Agent
    Mensaje: {
       sender: "suma_agent",
       receiver: "io_agent",
       content: {result: 14},

```
    type: "result"
  }
```

11. IO Agent retorna: 14

3.6 Diagrama de Secuencia



3.7 Algoritmo de Evaluación con Precedencia

```
ALGORITMO evaluar_expresión(tokens):

  // Paso 1: Resolver paréntesis
  MIENTRAS existan '(' en tokens:
    encontrar_paréntesis_más_interno()
    sub_tokens = tokens entre '(' y ')'
    resultado = evaluar_expresión(sub_tokens)  // Recursivo
    reemplazar sub_tokens con resultado
  FIN MIENTRAS

  // Paso 2: Resolver potencias (**)
  i = 0
  MIENTRAS i < longitud(tokens):
    SI tokens[i] == '**':
      operando1 = tokens[i-1]
      operando2 = tokens[i+1]
      resultado = solicitar_operación('**', operando1, operando2)
      tokens = tokens[:i-1] + [resultado] + tokens[i+2:]
    SINO:
      i = i + 1
```

```
        FIN SI
    FIN MIENTRAS

    // Paso 3: Resolver * y /
    i = 0
    MIENTRAS i < longitud(tokens):
        SI tokens[i] en ['*', '/']:
            operando1 = tokens[i-1]
            operando2 = tokens[i+1]
            resultado = solicitar_operación(tokens[i], operando1, operando2)
            tokens = tokens[:i-1] + [resultado] + tokens[i+2:]
        SINO:
            i = i + 1
        FIN SI
    FIN MIENTRAS

    // Paso 4: Resolver + y -
    i = 0
    MIENTRAS i < longitud(tokens):
        SI tokens[i] en ['+', '-']:
            operando1 = tokens[i-1]
            operando2 = tokens[i+1]
            resultado = solicitar_operación(tokens[i], operando1, operando2)
            tokens = tokens[:i-1] + [resultado] + tokens[i+2:]
        SINO:
            i = i + 1
        FIN SI
    FIN MIENTRAS

    // Resultado final
    RETORNAR tokens[0]
FIN ALGORITMO
```

4. IMPLEMENTACIÓN

4.1 Estructura del Proyecto

```
calculadora_agentes/
│
├── Sistema de Mensajería
│   ├── Message: Clase de mensajes
│   └── MessageBroker: Gestor de mensajes
│
├── Capa de Agentes
│   ├── OperationAgent: Clase base para agentes
│   ├── SumaAgent
```

```
│   ├── RestaAgent
│   ├── MultiplicacionAgent
│   ├── DivisionAgent
│   ├── PotenciaAgent
│   └── IOAgent: Coordinador principal
│
├── Modelo
│   └── CalculatorModel: Modelo MESA
│
└── Interfaz
    └── CalculatorGUI: Interfaz gráfica
```

4.2 Características Implementadas

A. Manejo de Errores

División por cero:

```
def safe_div(a, b):
    if b == 0:
        raise ValueError("División por cero no permitida")
    return a / b
```

Paréntesis no balanceados:

- Validación durante el parsing
- Mensaje de error específico

Operadores en posiciones inválidas:

- Validación de sintaxis
- Prevención de operadores consecutivos

B. Soporte de Números Negativos

El sistema maneja números negativos en:

- Inicio de expresión: `-5 + 3`
- Después de operadores: `2 * -3`
- Después de paréntesis: `(-5 + 3)`

C. Números Decimales

Soporta operaciones con:

- Enteros: `2 + 3`
- Decimales: `2.5 * 3.7`

- Mixtos: `2 + 3.14`

D. Logging Completo

Cada operación se registra con:

- Agente que la ejecuta
- Operandos involucrados
- Resultado obtenido
- Timestamp implícito (orden de ejecución)

Ejemplo de log:

```
============================================================
[IO] Nueva expresión: 2 + 3 * 4
============================================================
[IO] Tokens: ['2', '+', '3', '*', '4']
[IO] → Enviando a multiplicacion_agent: 3 * 4
[MULTIPLICACIÓN] 3 * 4 = 12
[IO] ← Recibido de multiplicacion_agent: 12
[IO] → Enviando a suma_agent: 2 + 12
[SUMA] 2 + 12 = 14
[IO] ← Recibido de suma_agent: 14
[IO] ✓ Resultado final: 14
============================================================
```

5. CASOS DE USO Y PRUEBAS

5.1 Casos de Prueba Básicos

Caso 1: Suma simple

- Entrada: `2 + 3`
- Tokens: `['2', '+', '3']`
- Agentes: Suma
- Resultado esperado: `5`
- Resultado obtenido: `5` ✓

Caso 2: Multiplicación antes de suma

- Entrada: `2 + 3 * 4`
- Ejecución:
    1. `3 * 4 = 12` (Mult Agent)
    2. `2 + 12 = 14` (Suma Agent)
- Resultado esperado: `14`

- Resultado obtenido: `14` ✓

Caso 3: División

- Entrada: `10 / 2`
- Agentes: División
- Resultado esperado: `5`
- Resultado obtenido: `5` ✓

Caso 4: Potencia

- Entrada: `2 ** 3`
- Agentes: Potencia
- Resultado esperado: `8`
- Resultado obtenido: `8` ✓

5.2 Casos de Prueba Intermedios

Caso 5: Paréntesis simples

- Entrada: `(2 + 3) * 4`
- Ejecución:
    1. `2 + 3 = 5` (Suma Agent)
    2. `5 * 4 = 20` (Mult Agent)
- Resultado esperado: `20`
- Resultado obtenido: `20` ✓

Caso 6: Múltiples operaciones

- Entrada: `10 - 3 + 2`
- Ejecución (izq. a der.):
    1. `10 - 3 = 7` (Resta Agent)
    2. `7 + 2 = 9` (Suma Agent)
- Resultado esperado: `9`
- Resultado obtenido: `9` ✓

Caso 7: Números decimales

- Entrada: `2.5 * 3.2 + 1.5`
- Ejecución:
    1. `2.5 * 3.2 = 8.0` (Mult Agent)
    2. `8.0 + 1.5 = 9.5` (Suma Agent)
- Resultado esperado: `9.5`
- Resultado obtenido: `9.5` ✓

5.3 Casos de Prueba Avanzados

Caso 8: Paréntesis anidados

- Entrada: `((2 + 3) * 4) - 1`
- Ejecución:
    1. `2 + 3 = 5` (Suma Agent)
    2. `5 * 4 = 20` (Mult Agent)
    3. `20 - 1 = 19` (Resta Agent)
- Resultado esperado: `19`
- Resultado obtenido: `19` ✓

Caso 9: Todas las operaciones

- Entrada: `2 ** 3 + 4 * 5 - 6 / 2`
- Ejecución:
    1. `2 ** 3 = 8` (Potencia Agent)
    2. `4 * 5 = 20` (Mult Agent)
    3. `6 / 2 = 3` (Div Agent)
    4. `8 + 20 = 28` (Suma Agent)
    5. `28 - 3 = 25` (Resta Agent)
- Resultado esperado: `25`
- Resultado obtenido: `25` ✓

Caso 10: Expresión compleja

- Entrada: `(2 + 3) * (4 - 1) ** 2`
- Ejecución:
    1. `2 + 3 = 5` (Suma Agent)
    2. `4 - 1 = 3` (Resta Agent)
    3. `3 ** 2 = 9` (Potencia Agent)
    4. `5 * 9 = 45` (Mult Agent)
- Resultado esperado: `45`
- Resultado obtenido: `45` ✓

Caso 11: Números negativos

- Entrada: `-5 + 3 * -2`
- Ejecución:
    1. `3 * -2 = -6` (Mult Agent)
    2. `-5 + -6 = -11` (Suma Agent)
- Resultado esperado: `-11`
- Resultado obtenido: `-11` ✓

5.4 Casos de Error

Caso E1: División por cero

- Entrada: `5 / 0`
- Resultado: `✗ Error: División por cero no permitida`
- Manejo: Excepción capturada ✓

Caso E2: Paréntesis no balanceados

- Entrada: `(2 + 3`
- Resultado: `✗ Error: Paréntesis no balanceados`
- Manejo: Validación sintáctica ✓

Caso E3: Operador inválido

- Entrada: `2 ++ 3`
- Resultado: Error de parsing
- Manejo: Validación de tokens ✓

5.5 Análisis de Comunicación

Expresión: `2 + 3 * 4`

Mensajes intercambiados: 4

1. `IO → Mult Agent`: Solicitud de `3 * 4`
2. `Mult Agent → IO`: Respuesta `12`
3. `IO → Suma Agent`: Solicitud de `2 + 12`
4. `Suma Agent → IO`: Respuesta `14`

Agentes involucrados: 3

- IO Agent (coordinador)
- Multiplicación Agent
- Suma Agent

Tiempo de ejecución: < 1ms