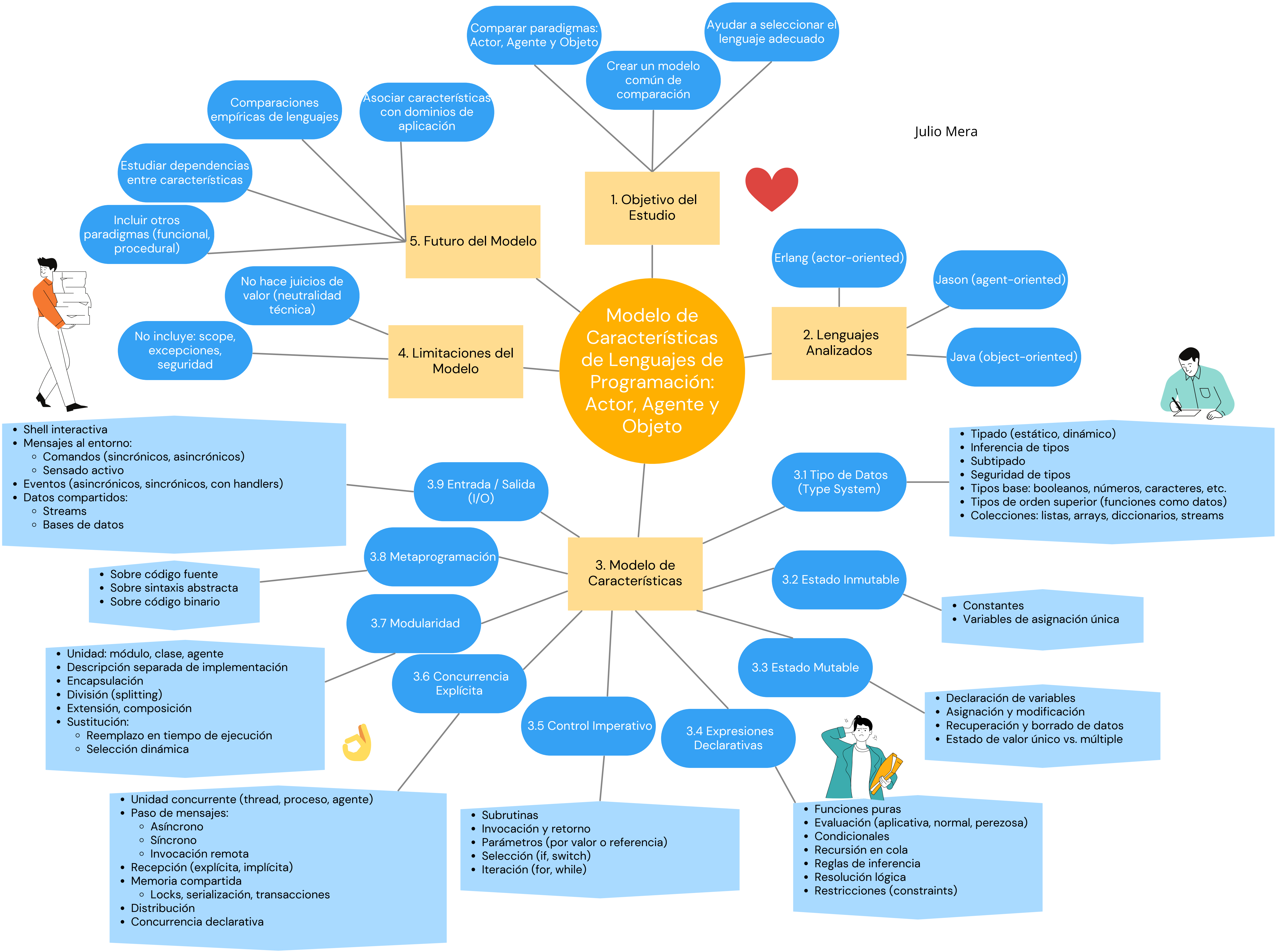

Julio Mera

Modelo de Características de Lenguajes de Programación: Actor, Agente y Objeto

1. Objetivo del Estudio

- Comparar paradigmas: Actor, Agente y Objeto
- Crear un modelo común de comparación
- Ayudar a seleccionar el lenguaje adecuado

2. Lenguajes Analizados

- Erlang (actor-oriented)
- Jason (agent-oriented)
- Java (object-oriented)

3. Modelo de Características

3.1 Tipo de Datos (Type System)

- Tipado (estático, dinámico)
- Inferencia de tipos
- Subtipado
- Seguridad de tipos
- Tipos base: booleanos, números, caracteres, etc.
- Tipos de orden superior (funciones como datos)
- Colecciones: listas, arrays, diccionarios, streams

3.2 Estado Inmutable

- Constantes
- Variables de asignación única

3.3 Estado Mutable

- Declaración de variables
- Asignación y modificación
- Recuperación y borrado de datos
- Estado de valor único vs. múltiple

3.4 Expresiones Declarativas

- Funciones puras
- Evaluación (aplicativa, normal, perezosa)
- Condicionales
- Recursión en cola
- Reglas de inferencia
- Resolución lógica
- Restricciones (constraints)

3.5 Control Imperativo

- Subrutinas
- Invocación y retorno
- Parámetros (por valor o referencia)
- Selección (if, switch)
- Iteración (for, while)

3.6 Concurrencia Explícita

- Unidad concurrente (thread, proceso, agente)
- Paso de mensajes:
  - Asíncrono
  - Síncrono
  - Invocación remota
- Recepción (explícita, implícita)
- Memoria compartida
  - Locks, serialización, transacciones
- Distribución
- Concurrencia declarativa

3.7 Modularidad

- Unidad: módulo, clase, agente
- Descripción separada de implementación
- Encapsulación
- División (splitting)
- Extensión, composición
- Sustitución:
  - Reemplazo en tiempo de ejecución
  - Selección dinámica

3.8 Metaprogramación

- Sobre código fuente
- Sobre sintaxis abstracta
- Sobre código binario

3.9 Entrada / Salida (I/O)

- Shell interactiva
- Mensajes al entorno:
  - Comandos (sincrónicos, asincrónicos)
  - Sensado activo
- Eventos (asincrónicos, sincrónicos, con handlers)
- Datos compartidos:
  - Streams
  - Bases de datos

4. Limitaciones del Modelo

- No hace juicios de valor (neutralidad técnica)
- No incluye: scope, excepciones, seguridad

5. Futuro del Modelo

- Comparaciones empíricas de lenguajes
- Asociar características con dominios de aplicación
- Estudiar dependencias entre características
- Incluir otros paradigmas (funcional, procedural)

Abstract del segundo paper

Título: A Comparative Study of Programming Languages in Rosetta Code
Autores: Sebastian Nanz, Carlo A. Furia
Fecha de publicación: 16 May 2015
Link de acceso: https://dl-acm-org.bibliotecavirtual.udla.edu.ec/doi/10.5555/2818754.2818848

“Sometimes debates on programming languages are more religious than scientific. Questions about which language is more succinct or efficient, or makes developers more productive are discussed with fervor, and their answers are too often based on anecdotes and unsubstantiated beliefs. In this study, we use the largely untapped research potential of Rosetta Code, a code repository of solutions to common programming tasks in various languages, which offers a large data set for analysis. Our study is based on 7'087 solution programs corresponding to 745 tasks in 8 widely used languages representing the major programming paradigms (procedural: C and Go; object-oriented: C# and Java; functional: F# and Haskell; scripting: Python and Ruby). Our statistical analysis reveals, most notably, that: functional and scripting languages are more concise than procedural and object-oriented languages; C is hard to beat when it comes to raw speed on large inputs, but performance differences over inputs of moderate size are less pronounced and allow even interpreted languages to be competitive; compiled strongly-typed languages, where more defects can be caught at compile time, are less prone to runtime failures than interpreted or weakly-typed languages. We discuss implications of these results for developers, language designers, and educators.”