

Práctica de Sistemas Basados en el Conocimiento Recomendación de Viviendas en Alquiler

Kirian Roca, Kai Knox e Iván Alcubierre

14 de diciembre de 2025

1. Introducción y planteamiento del problema

1.1. Estructura del documento

Este documento se organiza en tres partes principales:

- **Parte 1 (Introducción):** Planteamiento del problema, objetivos y metodología.
- **Parte 2 (Desarrollo):** Conceptualización, formalización (ontología) e implementación en CLIPS.
- **Parte 3 (Resultados):** Casos de prueba, análisis de resultados, conclusiones y trabajo en equipo.

1.2. Contexto y motivación

La práctica plantea el desarrollo de un sistema de recomendación de viviendas en alquiler que pueda ser utilizado por una entidad pública para ayudar a los ciudadanos a encontrar ofertas adecuadas a sus necesidades.

El problema implica combinar dos tipos de información:

- **Características de las viviendas:** precio, superficie, tipo de propiedad, amenities, ubicación, etc.
- **Perfil y preferencias del usuario:** presupuesto, composición familiar, necesidades de proximidad a servicios, restricciones, etc.

La complejidad del dominio y la necesidad de aplicar conocimiento experto para ponderar múltiples factores convierten este problema en un caso idóneo para un Sistema Basado en el Conocimiento (SBC).

1.3. Objetivos de la práctica

El objetivo principal de esta práctica es diseñar e implementar un **Sistema Basado en el Conocimiento (SBC)** que emule el razonamiento de un experto inmobiliario para recomendar viviendas en alquiler. Los objetivos específicos son:

- Analizar el dominio del problema e identificar las fuentes de conocimiento necesarias.

- Construir una ontología formal que represente los conceptos clave: **viviendas, solicitantes, servicios urbanos y zonas de la ciudad.**
- Implementar en CLIPS un sistema de reglas que realice:
 - Evaluación de compatibilidad entre perfil del solicitante y características de la vivienda.
 - Cálculo de proximidad a servicios relevantes (transporte, educación, ocio, salud, etc.).
 - Clasificación de las recomendaciones en tres categorías: *parcialmente adecuada, adecuada y muy recomendable.*
- Validar el sistema mediante casos de prueba representativos y documentar el proceso de desarrollo siguiendo la metodología de ingeniería del conocimiento.

1.4. Metodología de desarrollo

Para desarrollar el trabajo hemos seguido estos 5 pasos:

1. **Identificación:** Análisis del enunciado, definición de objetivos y alcance.
2. **Conceptualización:** Extracción y organización del conocimiento experto, identificación de conceptos y relaciones.
3. **Formalización:** Construcción de la ontología en Protégé y diseño de la estrategia de razonamiento.
4. **Implementación:** Codificación en CLIPS del sistema de reglas y hechos.
5. **Prueba:** Validación mediante casos de prueba y análisis de resultados.

2. Desarrollo de la solución

2.1. Conceptualización

2.1.1. Extracción de conocimiento

Entrevista con experto IA (Gemini 3.0)

Interviewer: "We are going to do a role playing exercise. You are going to help a knowledge engineer perform knowledge elicitation for building a rule-based expert system for the recommendation of home rentals. You are a real estate agent with wide experience in this problem who knows how to match different kinds of people (families with or without children, students, single people, elderly) according to the characteristics of the home and its neighborhood. The knowledge engineer is going to ask about the characteristics, criteria, and knowledge that you use for making the decisions. Your answer should be direct and precise. The answers should be helpful for building rules for making decisions from the answers of people who are looking to rent a flat or a house."**Expert:** To build a robust recommendation system, we must classify clients into four specific archetypes, each with distinct priorities:

- **Elderly:** Their non-negotiable priority is Accessibility. If a property is not on the ground floor, it must have an elevator. They require essential services (Pharmacy, Supermarket) within short walking distance and strictly reject "Nightlife" zones due to noise.
- **Families:** They prioritize Space and Safety. They need a specific number of bedrooms to match their family size. They prioritize proximity to Schools and Green Areas and avoid "High Noise." or "Tourist" districts.
- **Students:** They are budget-driven but have specific technical needs. Internet and Furnishing are mandatory hard constraints. They will tolerate noise or older building conditions if the price is right. They prioritize proximity to the University or Public Transport.
- **Young Couples/Professionals:** They prioritize Lifestyle. They value proximity to Gyms, Restaurants, and Leisure. If they own a car, a Garage becomes a critical requirement due to the Lifestyle urban location.

Interviewer: "Now I will send you a basis of the attributes for clients and properties that I am sure they have to be. Can you use your expert knowledge to suggest some attributes to make it more complete?"

1. Monthly price in euros.
 2. Location (city district, address, GPS coordinates).
 3. Usable floor area in m².
 4. Number of bedrooms and whether they are double or single.
 5. Whether the dwelling has a terrace or balcony.
 6. Whether the dwelling is rented furnished and/or with appliances.
 7. Type of dwelling: flat, duplex, single-family house. For flats or duplexes, the floor level (mezzanine, first, second...).
 8. Other features such as: no-pets rule, heating/air conditioning, whether the flat gets sunlight (morning, afternoon, all day), access to a communal pool, parking space included, property with views.
-
1. Maximum price they are willing to pay and whether the limit is strict or they could pay more if the offer is worth it.
 2. Restrictions regarding the number/size of the bedrooms.
 3. Minimum price, below which they believe the offer is too cheap to be adequate.
 4. Specific restriction or preference regarding distance to certain services (schools nearby, public transport nearby...).
 5. Whether they prefer to use public transport for commuting.
 6. Age of the applicants.

7. Type of applicants: couple without children, couple planning to have children soon, family (single-parent, two-parent) with children/elderly dependents (number), group of students (number), individual...
8. Whether they work/study in the city and where the workplace/place of study is located.
9. Whether they own a car."

Expert: To strengthen the system and reduce ambiguity in decision rules, we need additional measurable attributes for both Property and Clients. These enable clearer constraints, preference scoring, and rule branching.

Property – Suggested Additional Attributes

- Noise Level Classification: Quiet / Moderate / High. Necessary for Families and Elderly.
- Accessibility Indicators: Presence of elevator, ramp access, door width for wheelchair. Critical for Elderly.
- Energy Efficiency Rating: Impacts long-term cost; students and families react strongly to energy bills.
- Internet Availability and Speed: Mandatory for Students; highly relevant for Couples/Professionals working remotely.
- Heating Type and Insulation Quality: Relevant in older buildings; impacts comfort and utility expenses.
- Proximity Metrics: Walking time (not just distance) to public transport, university/schools, supermarkets, pharmacy.
- Neighborhood Category: Residential / Tourist / Nightlife / Business. Used to match lifestyle vs. noise tolerance.
- Safety Indicators: Crime level or “perceived safe area” flag. Families care; students tolerate lower.
- Building Age and Renovation Condition: Students tolerate older; families avoid degraded.
- Contract Constraints: Minimum rental duration, deposit size. People with strict budgets (students) react strongly.
- Pets Allowed / Forbidden: Already noted, but must be categorical because it blocks many options.

Clients – Suggested Additional Attributes

- Noise Tolerance Level: Critical decision boundary between Elderly/Families vs. Students/Professionals.
- Mobility or Accessibility Needs: Presence of disability or reduced mobility; elevator becomes hard constraint.
- Internet Requirement: Binary but decisive.
- Expected Lease Duration: Students short-term; families long-term. Impacts landlord suitability.
- Work Modality: Remote/hybrid/on-site. Remote workers require strong internet and quiet environment.
- Lifestyle Priority: Quiet, central/leisure-oriented, green areas, nightlife. Makes rule pruning more accurate.
- Commute Distance Tolerance: Define acceptable walking/public transport travel time.
- Car Usage Frequency: Not only ownership; usage defines need for guaranteed parking.

These additions allow the rules to be actionable and conflict-resolving instead of vague.

Interviewer: “Can you suggest some filters that you apply to decide if a Property is good to a certain client?” **Expert:** First, we must apply a hard financial filter before looking at preferences.

Financial Filters

- The 40 % Rule: The monthly rent must not exceed 40 % of the household’s total monthly income. This is a safety check for solvency.
- Budget Limit: We respect the client’s "Maximum Price," but we should allow a small margin (e.g., +10 %) if the property is Ideal, unless the client marks their budget as "Strict."

Next, we apply structural and capacity filters to guarantee functional suitability:

Structural Filters

- Bedroom/Occupancy Consistency: The number and type of bedrooms must align with the household size. A family of four cannot be matched with a single-bedroom flat. Couples planning children should not be placed in a studio.
- Accessibility Compliance: If any applicant has reduced mobility or is elderly, the property must have either ground-floor access or a functioning elevator. Absence of these eliminates the property immediately.

- **Mandatory Utility Requirements:** If the client requires high-speed internet (students, remote workers), or heating/air conditioning (elderly, families in cold/hot climates), properties without these features are discarded.
- **Transportation or Parking Constraint:** If the client relies on public transport, the property must be within a reasonable walking radius of a reliable connection. If the client owns a car and parking is required, the property must provide either a garage or a guaranteed parking spot.
- **Neighborhood Compatibility:** Noise and security conditions must be aligned with the sensitivity of the client type. Nightlife areas are automatically excluded for elderly or families. Tourist zones are low priority for families but acceptable for students.

These filters eliminate structurally incompatible options before any ranking or preference scoring begins.

Relación entre el conocimiento del experto y el código CLIPS:

Aplicado

1. **Presupuesto** Se solicita precio máximo y si es estricto o flexible. Se descartan propiedades que superan el máximo; si es flexible se permite hasta 10 %.
2. **Dormitorios y espacio** Filtrado por número de dormitorios dobles y simples.
3. **Mascotas** Se descartan viviendas donde no se permiten mascotas.
4. **Coche** Atr. owns-car, propiedades con **Garage**. Eliminación o penalización según cliente.
5. **Trabajo/estudio** Se registran coordenadas de trabajo/estudio.
6. **Regla del 40 %** Se registra ingreso mensual total y se descartan viviendas fuera del presupuesto.
7. **Accesibilidad** Atributos de ascensor y planta; reglas para clientes sensibles.
8. **Ruido** Atributos de ruido recibidos; filtros por tipo de cliente.
9. **Seguridad** Atributo de nivel de seguridad; filtrado según cliente.
10. **Proximidad a servicios** Coordenadas de servicios y propiedades.
11. **Condiciones climáticas (sol)** Atributos de AC, heating y orientación solar.

No aplicado

1. **Internet** Se considera que cada cliente contrata su propio servicio.
2. **Clima adicional no relacionado con sol** Se omiten factores climáticos globales para reducir complejidad del modelo.

2.1.2. Conceptos y relaciones principales

A partir del análisis del dominio y de la experiencia en portales inmobiliarios, identificamos los siguientes conceptos clave, que posteriormente se formalizarán en la ontología:

- **Cliente:** persona o grupo que busca vivienda. Se distingue entre:
 - **Individual:** Adulto, Joven, Estudiante, Anciano.
 - **Grupo:** Pareja (con o sin hijos previstos, joven o anciana), Familia (con o sin ancianos), Grupo de estudiantes.
- **Vivienda:** oferta de alquiler. Incluye subtipos como:
 - **Piso, Dúplex, Ático, Estudio.**
 - **Casa individual, Casa pareada, Casa adosada.**
 - **Habitación** para alquiler individual.
- **Servicio:** elemento urbano que influye en la calidad de vida:
 - **Transporte:** Parada de autobús, metro, tren, aparcamiento.
 - **Educación:** Colegio, instituto, universidad.
 - **Salud:** Clínica, hospital, farmacia.
 - **Comercio:** Supermercado, hipermercado, centro comercial, tienda.
 - **Ocio:** Parque, piscina, gimnasio, cine, museo, zona de vida nocturna, restaurante, playa, polideportivo.
- **Zona:** área de la ciudad con características homogéneas:
 - **Residencial, Centro, Núcleo urbano, Negocios, Industrial.**

Relaciones principales identificadas:

- Una **Vivienda** *está ubicada en* una **Zona** (relación `is_located_in_zone`).
- Una **Vivienda** *está cerca de* uno o más **Servicios** (relación `is_near_service`).
- Un **Cliente** *tiene preferencias y restricciones* sobre:
 - Precio máximo, flexibilidad presupuestaria, precio mínimo sospechoso (*too bargain*).
 - Número mínimo de dormitorios y baños.
 - Necesidad de transporte público, posesión de coche.
 - Ubicación de trabajo/estudio (coordenadas).
 - Mascotas, ruido tolerado, características deseadas (terraza, ascensor, amueblado, etc.).
- Una **Vivienda** *puede satisfacer o no* dichas preferencias, lo que determina su **grado de recomendación** (parcialmente adecuada, adecuada, muy recomendable).

2.2. Formalización

2.2.1. Ontología del dominio

La ontología se ha construido en Protégé siguiendo la metodología de desarrollo de ontologías 101, y está compuesta por 4 conceptos raíz principales, sus jerarquías de subclases, propiedades de objeto y datos, y restricciones de dominio/rango.

Jerarquía de clases

- **Client_Group**: representa a los solicitantes. Se divide en:
 - **Individual**: Adult, Young, Student, Elderly.
 - **Group**: Couple (Elderly_Couple, Planning_Kids, Young_No_Kids), Family (No_Elderly, With_Elderly), Student_Group.

Se ha modelado así para capturar perfiles con necesidades diferenciadas (ej: jóvenes buscan ocio, familias buscan escuelas).

- **Property**: representa las viviendas en alquiler. Incluye:
 - Tipos urbanos: Apartment, Duplex, Penthouse, Studio, Triplex, Room.
 - Tipos unifamiliares: Detached_House, Semidetached_House, Row_House, Single-family_House.

Se han definido como *disjoint* para evitar inconsistencias.

- **Service**: modela los servicios urbanos. Subclases:
 - Transport (Bus_stop, Metro_station, Train_station, Parking, Cable_car_station).
 - Education (School, High_School, University).
 - Healthcare (Clinic, Hospital, Pharmacy).
 - Shopping (Supermarket, Hypermarket, Mall, Store).
 - Recreation (Park, Pool, Gym, Beach, Cinema, Museum, Nightlife, etc.).

Esta categorización permite evaluar la proximidad a servicios de forma estructurada.

- **Zone**: describe zonas de la ciudad:
 - Residential, Business, Downtown, Industrial, Urban_core.

Cada zona tiene atributos como nivel de ruido y seguridad.

Propiedades de objeto

- **is_located_in_zone**: relaciona una **Property** con una **Zone**. Es funcional (cada vivienda está en una sola zona).
- **is_near_service**: relaciona una **Property** con un **Service**. Permite modelar proximidad a múltiples servicios.

Propiedades de datos Se han agrupado en tres categorías para facilitar la modularidad:

- **property_attribute:** atributos de la vivienda (precio, metros², habitaciones, terraza, ascensor, amueblado, etc.).
- **client_attribute:** atributos del solicitante (edad, presupuesto, preferencias, ubicación trabajo/estudio, etc.).
- **zone_attribute:** atributos de la zona (nivel de ruido, seguridad).

Se han usado tipos de datos específicos (`boolean`, `int`, `float`) y enumerados (ej: `Sun_Time: Morning, Afternoon, All_Day, Never`).

Decisiones de diseño justificadas

- **Coordenadas enteras:** se usan `geo_lat` y `geo_long` como enteros para simplificar el cálculo de distancias en CLIPS.
- `floor_level` solo aplica a `Apartment`, `Room` y `Studio`, no a viviendas unifamiliares.
- `Too_Bargain:` se incluye como atributo del cliente para detectar precios sospechosamente bajos.
- *Disjoint* entre clases: evita que una misma instancia sea, por ejemplo, un `Apartment` y una `Detached_House`.

2.2.2. Metodología de resolución de problemas

Usamos clasificación heurística como metodología de resolución de problemas y la dividimos en estas partes:

1. **Reglas de abstracción:** Para transformar datos concretos (coordenadas, edades, precios) en categorías simbólicas que permiten una evaluación cualitativa. Esto incluye:
 - Abstracción implícita (datos ya categóricos).
 - Abstracción inducida (preferencias deducidas del perfil demográfico).
 - Abstracción explícita (cálculo de distancias y ratios).
2. **Reglas de Clasificación Heurística:** Cada propiedad se evalúa contra una lista de criterios derivados del perfil del cliente. En lugar de una puntuación numérica, se utiliza un sistema de *checklist* cualitativo que registra cumplimientos, fallos y extras.
3. **Clasificación por categorías predefinidas:** El resultado de la evaluación de clasificación heurística se traduce a una de cuatro categorías de recomendación, facilitando la interpretación por parte del usuario final.
4. **Estrategia de búsqueda exhaustiva:** El sistema evalúa **todas** las propiedades disponibles contra el perfil del cliente, sin poda temprana, garantizando que ninguna opción potencialmente adecuada sea descartada prematuramente.

Esta metodología permite un balance entre flexibilidad (acepta múltiples tipos de clientes y propiedades) y transparencia (el usuario puede ver qué criterios se cumplen o fallan).

2.2.3. Diseño del Sistema de Reglas

El sistema implementa tres tipos principales de reglas que permiten transformar los datos concretos del dominio en conocimiento abstracto para la toma de decisiones. Estas reglas siguen la metodología de resolución de problemas descrita anteriormente.

2.2.3.1 Reglas de Abstracción Estas reglas realizan la transición del **problema concreto** al **problema abstracto**, transformando datos brutos en atributos categóricos que permiten una evaluación sistemática. Se dividen en tres subtipos:

Reglas de Abstracción Implícita: La información ya se encuentra en formato abstracto cuando el usuario la proporciona. No se requiere procesamiento adicional.

- **Ejemplo:** Cuando el usuario responde “¿*Tiene coche propio?*” con **yes**, el sistema asigna directamente **yes** al atributo **owns_car**.
- **Otros ejemplos:** **has_pets**, **Budget_Is_Strict**, **Works_In_City**.

Reglas de Abstracción Inducida: Se infieren preferencias a partir de información demográfica, basándose en conocimiento experto sobre patrones comunes.

- **Ejemplo:** Si el cliente es clasificado como **Elderly**, **Elderly_Couple** o **With_Elderly**, el sistema induce preferencias como "elevator", "quiet", "single-floor", "healthcare".
- Estas preferencias se añaden automáticamente a **desired_features**.

Reglas de Abstracción Explícita: Transforman datos numéricos o geográficos en categorías simbólicas mediante fórmulas y umbrales predefinidos. Estas reglas realizan cálculos concretos que permiten evaluar características espaciales y cuantitativas de las propiedades.

- **Cálculo de densidad de habitaciones (room-density):** Para determinar si una propiedad tiene espacio suficiente para el grupo de clientes, el sistema calcula:

$$\text{capacidad} = (\text{Num_Double_Rooms} \times 2) + \text{Num_Single_Rooms}$$

$$\text{ratio} = \frac{\text{capacidad}}{\text{num_people}}$$

La clasificación resultante es:

- **saturated** (ratio <1.0): insuficiente espacio
 - **sufficient** (ratio = 1.0): espacio justo
 - **spacious** (ratio >1.0): espacio abundante
- **Cálculo de distancias euclidianas:** Todas las mediciones de proximidad se basan en la función **euclid-dist**, que calcula la distancia en línea recta entre dos puntos geográficos:

$$\text{distancia}(x_1, y_1, x_2, y_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Donde (x_1, y_1) y (x_2, y_2) son las coordenadas enteras de los puntos a comparar.

- **Asignación de servicios cercanos:** Aplicando la función `euclid-dist`, el sistema clasifica la proximidad de cada servicio a una propiedad mediante dos reglas:
 - `inicio-calcular-cercania: servicio ≤ 200 m → is_near_service`
 - `inicio-calcular-media-cercania: servicio entre 200-700 m → is_semi_near_service`

Estas reglas se ejecutan al inicio del sistema con prioridad alta (`salience 100 y 99`) para garantizar que todas las conexiones se establezcan antes del proceso de recomendación.

- **Cálculo de distancias a trabajo/estudio para grupos:** Cuando múltiples personas trabajan o estudian en diferentes ubicaciones, el sistema implementa una estrategia conservadora considerando la **peor situación**. Para cada propiedad:
 1. Se calcula la distancia entre la propiedad y cada ubicación de trabajo/estudio usando `euclid-dist`.
 2. Se determina la **distancia máxima** mediante la función `max-distance`, que itera sobre todas las coordenadas proporcionadas.
 3. Esta distancia máxima se clasifica según umbrales predefinidos:
 - `<1000 m → cerca`
 - `1000-3500 m → media_distancia`
 - `>3500 m → lejos`

Justificación del diseño: Esta aproximación garantiza que la recomendación sea válida para todos los miembros del grupo, incluso para quien tenga el trayecto más largo. Si la propiedad satisface la distancia máxima, automáticamente satisface las distancias menores de otros miembros.

- **Función de clasificación de distancias:** La función auxiliar `classify-distance` encapsula la lógica de umbrales, permitiendo una modificación centralizada de los criterios de proximidad si fuera necesario en futuras versiones del sistema.

2.2.3.2 Reglas de Evaluación de Criterios Una vez obtenidos los atributos abstractos, el sistema evalúa cada propiedad contra los criterios del cliente. Estas reglas comparan características específicas y generan evaluaciones cualitativas.

Ejemplos de criterios evaluados:

- **Compatibilidad presupuestaria:** Compara `monthly_price` de la propiedad con `max_budget` del cliente.
- **Compatibilidad con mascotas:** Verifica `Pets_Allowed` de la propiedad contra `has_pets` del cliente.
- **Densidad adecuada:** Evalúa `room-density` para determinar si hay suficiente espacio.
- **Distancia a servicios clave:** Comprueba proximidad a trabajo/estudio según categorías `cerca`, `media_distancia`, `lejos`.

- **Satisfacción de características deseadas:** Para cada elemento en `desired_features`, verifica si la propiedad lo cumple mediante la función `feature-satisfied`.

Salida de estas reglas: Cada propiedad recibe dos listas:

- `met-criteria`: Criterios que satisface.
- `failed-criteria`: Criterios que no satisface.
- `extra-criteria`: Características adicionales positivas no solicitadas explícitamente.

2.2.3.3 Reglas de Clasificación y Recomendación Finalmente, el sistema clasifica cada propiedad en una de cuatro categorías de recomendación basándose en los resultados de la evaluación.

Lógica de clasificación implementada:

1. **No recomendado:** Si la propiedad falla en más de 3 criterios esenciales.
2. **Parcialmente adecuada:** Si falla en 1-3 criterios, pero satisface los esenciales.
3. **Adecuada:** Si satisface todos los criterios sin características extras destacables.
4. **Muy recomendable:** Si satisface todos los criterios y además tiene características extras valoradas (ej: estado excelente, parque cercano, terraza, piscina).

Estrategia de implementación: La regla `classify-property` analiza las listas `failed-criteria` y `extra-criteria` para asignar la categoría final. Esta clasificación permite presentar al usuario un ranking cualitativo de las propiedades disponibles.

Ejemplo de salida:

```
Propiedad apt-101 -> muy-adeecuado
Pros: [entra-en-presupuesto, admite-mascotas, ...]
Contras: ninguno
Extras: [estado-excelente, parque-cerca]
```

2.3. Implementación en CLIPS

2.3.1. Estructura del proyecto

El sistema está organizado en varias partes claramente separadas mediante comentarios en el código CLIPS, que se ejecutan en un orden lógico controlado por `salience` (prioridad):

1. **Lógica de startup y cálculo de proximidad (salience 100-99):** Inicializa las conexiones entre propiedades y servicios cercanos mediante reglas `inicio-calcular-cercania` e `inicio-calcular-media-cercania`.
2. **Herramientas de entrevista (funciones):** Contiene funciones utilitarias para la interacción con el usuario: `ask-number`, `ask-int-range`, `ask-yes-no`, `ask-coordinate`, etc.
3. **Estructura de datos temporal:** Define el `deftemplate user-responses` que almacena las respuestas del cuestionario paso a paso.

4. **Reglas del flujo de preguntas (salience variable):** Implementa el cuestionario guiado mediante reglas encadenadas que modifican el paso actual en `user-responses`. Incluye detección automática de perfiles (parejas, estudiantes).
5. **Lógica de clasificación y creación de instancia:** Contiene la regla `determine-and-create-cl` que, mediante un árbol de decisión, clasifica al cliente en una subclase específica y crea la instancia correspondiente.
6. **Cálculo de distancias trabajo/estudio y densidad (salience -20):** Calcula atributos abstractos para cada propiedad respecto al cliente creado: `work-place`, `study-place`, `room-density`.
7. **Recomendación cualitativa por criterios (salience -40 a -45):** Evalúa cada propiedad contra los criterios del cliente, genera listas de cumplimientos/fallos/extras, y clasifica la propiedad en una categoría final de recomendación.

Hechos iniciales: El sistema se inicia con el hecho (`start-execution`) y las instancias predefinidas en `definstances` (zonas, servicios y propiedades con atributos realistas y coordenadas coherentes).

2.3.2. Ejemplo de reglas clave

El sistema implementa diversos tipos de reglas que cubren desde la inicialización del sistema hasta la recomendación final. A continuación se muestran fragmentos comentados de algunas reglas representativas.

```
;; Conecta propiedades con servicios cercanos (<=200m)
(defrule inicio-calcular-cercania
  (declare (salience 100)) ; Prioridad máxima: se ejecuta primero
  ?p <- (object (is-a Property)
           (geo_lat ?plat) (geo_long ?plong)
           (is_near_service $?lista))
  ?s <- (object (is-a Service)
           (geo_lat ?slat) (geo_long ?slong))

  ;; Evita duplicados: solo si el servicio NO est ya en la lista
  (test (eq (member$ ?s ?lista) FALSE))
  ;; Condición de distancia: 200 metros o menos
  (test (<= (euclid-dist ?plat ?plong ?slat ?slong) 200))
  =>
  ;; Acción: añade el servicio a la lista de cercanías
  (send ?p put-is_near_service (insert$ ?lista 1 ?s))
  (printout t "Sistema: Conectado " (instance-name ?p)
            " con servicio cercano " (instance-name ?s) crlf)
)
```

Listing 1: Regla de inicialización: cálculo de proximidad a servicios

Esta regla se ejecuta con prioridad máxima (`salience 100`) al inicio del sistema. Su función es precalcular las relaciones espaciales entre propiedades y servicios, almacenando en el multislot `is_near_service` todos los servicios ubicados a menos de 200 metros. Este cálculo inicial optimiza el sistema, evitando recalcular distancias durante la evaluación de cada cliente.

```
;; Paso 3 del cuestionario: informaci3n econ3mica b3sica
(defrule ask-economics
  ?f <- (user-responses (step ask-economics))
  =>
  ;; Preguntas interactivas al usuario
  (bind ?inc (ask-number " Cual es vuestro ingreso mensual total
    (neto)?"))
  (bind ?bud (ask-number " Cual es vuestro presupuesto maximo
    objetivo?"))
  (bind ?strict (ask-yes-no " Es este presupuesto estricto (no
    podeis pagar mas)?"))
  (bind ?car (ask-yes-no " Teneis coche propio?"))
  (bind ?pet (ask-yes-no " Teneis mascotas?"))

  ;; Actualiza el template temporal con las respuestas
  (modify ?f (income ?inc)
    (max_budget ?bud)
    (budget_is_strict ?strict)
    (owns-car ?car)
    (has-pets ?pet)
    (step ask-work-logistics)) ; Pasa al siguiente paso
)
```

Listing 2: Regla del flujo de entrevista: captura de datos econ3micos

Esta regla forma parte del motor de entrevista interactiva. Utiliza funciones auxiliares (ask-number, ask-yes-no) para capturar datos del usuario y actualiza el deftemplate temporal user-responses. El sistema avanza modificando el slot step, lo que activa la siguiente regla del flujo.

```
;; Fragmento de la regla determine-and-create-client-final
(if (= ?np 1) then ; CASO 1: INDIVIDUAL
  (printout t "DEBUG: Rama INDIVIDUAL" crlf)
  (if ?has-elderly then
    (bind ?class-name Elderly)
    (printout t "DEBUG: Asignado ELDERLY" crlf)
  else
    (if (eq ?s-any yes) then
      (bind ?class-name Student)
      (printout t "DEBUG: Asignado STUDENT" crlf)
    else
      (if (<= (nth$ 1 ?ages) 30) then
        (bind ?class-name Young)
        (printout t "DEBUG: Asignado YOUNG" crlf)
      else
        (bind ?class-name Adult)
        (printout t "DEBUG: Asignado ADULT" crlf)
      )
    )
  )
)
```

Listing 3: Regla de clasificaci3n: 3rbol de decisi3n para individuos

Este fragmento muestra la lógica de clasificación para clientes individuales. El sistema evalúa secuencialmente: 1) si es anciano (edad ≥ 65), 2) si es estudiante (estudia en la ciudad), 3) si es joven (edad ≤ 30), 4) en caso contrario, adulto. Esta estructura de árbol de decisión codifica el conocimiento experto sobre las características distintivas de cada perfil.

```
;; Calcula si la propiedad tiene espacio suficiente para el grupo
(defrule calculate-room-density
  (declare (salience -20)) ; Prioridad baja: se ejecuta después
  (current-client ?client-id)
  ?client <- (object (is-a Client_Group)
                  (num_people ?num-people))
  ?prop <- (object (is-a Property)
                  (Num_Double_Rooms ?double-rooms)
                  (Num_Single_Rooms ?single-rooms))
  (not (room-density-done ?prop)) ; Evita recálculos
=>
  ;; Capacidad total: habitaciones dobles cuentan como 2 plazas
  (bind ?capacity (+ (* ?double-rooms 2) ?single-rooms))

  ;; Ratio: capacidad / número de personas
  (bind ?ratio (/ ?capacity ?num-people))

  ;; Clasificación según umbrales predefinidos
  (bind ?density
    (if (< ?ratio 1.0) then saturated
        else (if (= ?ratio 1.0) then sufficient
                  else spacious)))

  (send ?prop put-room-density ?density)
  (assert (room-density-done ?prop)) ; Marca como procesado
)
```

Listing 4: Regla de evaluación: cálculo de densidad de habitaciones

Esta regla de abstracción transforma datos concretos (número de habitaciones) en categorías simbólicas (*saturated*, *sufficient*, *spacious*). El uso de *salience -20* asegura que se ejecute después de la creación del cliente, y el hecho *room-density-done* evita procesamiento redundantes.

```
;; Evalúa múltiples criterios contra una propiedad
(defrule evaluate-basic-criteria
  (declare (salience -40))
  (current-client ?cid)
  ?c <- (object (is-a Client_Group)
                (has_pets ?pets)
                (max_budget ?maxb)
                (Works_In_City ?works)
                (Study_In_City ?studies)
                (desired_features $?feat))
  ?p <- (object (is-a Property)
                (monthly_price ?price)
                (Pets_Allowed ?pets-ok))
```

```

        (room-density ?dens)
        (work-place ?workplace)
        (study-place ?studyplace))
=>
(bind ?fails (create$))
(bind ?mets (create$))

;; Criterio 1: Presupuesto (filtro duro)
(if (> ?price ?maxb)
    then (bind ?fails (insert$ ?fails 1 fuera-de-presupuesto))
    else (bind ?mets (insert$ ?mets 1 entra-en-presupuesto)))

;; Criterio 2: Mascotas
(if (eq ?pets yes) then
    (if (eq ?pets-ok no)
        then (bind ?fails (insert$ ?fails 1 no-admite-mascotas)
            )
        else (bind ?mets (insert$ ?mets 1 admite-mascotas))))

;; Criterios 3-5: Densidad, trabajo, estudio
(if (eq ?dens saturated)
    then (bind ?fails (insert$ ?fails 1 poco-espacio)))

(if (eq ?works yes) then
    (if (eq ?workplace lejos)
        then (bind ?fails (insert$ ?fails 1 lejos-trabajo))
        else (bind ?mets (insert$ ?mets 1 cerca-trabajo))))

;; Evaluación de características deseadas dinámicamente
(foreach ?f ?feat
    (if (feature-satisfied ?p ?f)
        then (bind ?mets (insert$ ?mets 1 (sym-cat cumple- ?f)))
        else (bind ?fails (insert$ ?fails 1 (sym-cat falta- ?f))))
    )

(assert (prop-assessment (prop (instance-name ?p))
    (failed-criteria ?fails)
    (met-criteria ?mets)))
)

```

Listing 5: Regla de evaluación principal: criterios básicos

Esta regla es el núcleo del sistema de evaluación. Implementa una estrategia de *checklist* múltiple donde cada propiedad se evalúa contra todos los criterios relevantes del cliente. Los resultados se almacenan en el template `prop-assessment` para su posterior clasificación. La función `feature-satisfied` permite evaluar dinámicamente características específicas solicitadas por el cliente.

Cuadro 1: Clasificación de reglas en el sistema CLIPS

| Tipo de regla | Función principal | Ejemplo |
|------------------------------------|-----------------------------------|---|
| Reglas de inicialización | Precalcular relaciones espaciales | <code>inicio-calcular-cercania</code> |
| Reglas de flujo | Guiar la entrevista interactiva | <code>ask-economics</code> , <code>ask-work-logistics</code> |
| Reglas de clasificación | Determinar perfil del cliente | <code>determine-and-create-client-final</code> |
| Reglas de abstracción | Transformar datos en categorías | <code>calculate-room-density</code> |
| Reglas de clasificación heurística | Comparar propiedad con criterios | <code>evaluate-basic-criteria</code> |
| Reglas de recomendación | Asignar categoría final | <code>classify-property</code> |

Resumen de tipos de reglas implementadas: El sistema utiliza *salience* (prioridad) para controlar el orden de ejecución: las reglas de inicialización tienen prioridad alta (100-99), las de evaluación media (-20 a -40), y las de recomendación prioridad baja (-45). Este diseño asegura que todos los datos necesarios estén calculados antes de tomar decisiones de recomendación.

2.3.3. Desarrollo incremental

El desarrollo del sistema siguió una estrategia incremental, construyendo prototipos funcionales que evolucionaron hasta la versión final:

- Prototipo 1: Ontología y datos básicos.** Se creó la ontología en Protégé con las cuatro jerarquías principales (Client_Group, Property, Service, Zone) y se exportó a CLIPS mediante la herramienta `owl2clips`. Se definieron instancias de prueba mínimas (3 propiedades, 5 servicios) para validar la estructura básica.
- Prototipo 2: Entrevista y clasificación del cliente.** Se implementó el flujo de preguntas interactivo con reglas encadenadas (`ask-economics`, `ask-work-logistics`, etc.) y la lógica de clasificación demográfica en la regla `determine-and-create-client-final`. En esta versión, las recomendaciones se basaban únicamente en edad y número de personas, sin considerar servicios ni distancias.
- Prototipo 3: Cálculo de proximidades y distancias.** Se añadieron las funciones de distancia euclidiana (`euclid-dist`) y las reglas de startup `inicio-calcular-cercania` e `inicio-calcular-media-cercania` para conectar propiedades con servicios. También se implementó el cálculo de distancias a trabajo/estudio mediante las reglas `calculate-work-distances` y `calculate-study-distances`.
- Prototipo 4: Implementación de reglas de abstracción y asociación heurística.** Se desarrollaron dos categorías principales de reglas:
 - **Reglas de abstracción:** Transforman datos concretos en categorías simbólicas. Se implementaron reglas como `calculate-room-density` (clasifica densi-

dad de habitaciones como `saturated`, `sufficient` o `spacious`) y las funciones de clasificación de distancias (`classify-distance`).

- **Reglas de asociación heurística:** Aplican conocimiento experto para evaluar compatibilidad entre propiedades y clientes. La regla principal `evaluate-basic-criteria` implementa un sistema de *checklist* que compara múltiples atributos del cliente con los de la propiedad, generando listas de criterios cumplidos y fallados.

Se introdujo la clasificación final mediante la regla `classify-property` que, basándose en los resultados de las reglas anteriores, asigna categorías de recomendación (`partial`, `adecuado`, `muy-adecuado`, `ideal`).

5. **Prototipo final: Depuración y datos realistas.** Se expandió el conjunto de instancias en `definstances` a 20 propiedades y 50+ servicios con coordenadas coherentes. Se afinaron los umbrales de distancia, se añadieron criterios extras (orientación solar, calefacción, mobiliario) y se implementaron mecanismos para evitar bucles infinitos (`test (member$...)`) y reprocesamiento (`work-distance-done`, `study-distance-done`). Se ajustaron las prioridades (`salience`) para optimizar el flujo de ejecución.

Cada prototipo fue probado con casos específicos, permitiendo detectar y corregir inconsistencias (ej: servicios duplicados en listas, clasificación errónea de parejas sin hijos, manejo de coordenadas múltiples). Este enfoque incremental permitió validar cada componente del sistema de forma aislada antes de integrarlo en el conjunto, reduciendo la complejidad del proceso de depuración y facilitando la identificación temprana de problemas de diseño.

3. Resultados y validación

3.1. Juegos de prueba

Explicación de cómo se seleccionaron los casos: perfiles variados (estudiante, familia, pareja), casos límite, cobertura de funcionalidades.

3.2. Ejecución de pruebas

Tabla resumen con al menos 6 casos diferentes, mostrando:

- Perfil de entrada.
- Viviendas recomendadas (adecuadas, parcialmente adecuadas, muy recomendables).
- Criterios incumplidos o características destacadas.

3.3. Análisis de resultados

Explicación de por qué el sistema toma ciertas decisiones, ejemplos concretos basados en el conocimiento codificado.

Cuadro 2: Ejemplo de casos de prueba

| Perfil | Vivienda | Grado | Observaciones |
|-------------------------|-------------------|------------------|---------------------------|
| Estudiante, precio bajo | Atico céntrico | Parcial | Sin terraza |
| Familia con hijos | Dúplex con jardín | Muy recomendable | Cerca de colegio y parque |

3.4. Conclusiones

3.4.1. Logros y dificultades

Qué se ha conseguido, qué partes fueron más complejas, limitaciones del sistema.

3.4.2. Trabajo en equipo

Breve descripción de cómo se organizó el grupo, reuniones, reparto de tareas, uso de la planificación propuesta.

3.4.3. Posibles mejoras

Extensiones futuras: integración con datos reales de mapas, más factores de ponderación, interfaz gráfica.

A. Anexo A: Ontología completa

Gráfico exportado desde Protégé con la jerarquía de clases y propiedades.

B. Anexo B: Diálogo con modelo de lenguaje

Si aplica: contexto usado, preguntas más relevantes y respuestas resumidas.

C. Anexo C: Código fuente completo

Listado completo del código CLIPS (o enlace al repositorio).