

3. ANTECEDENTES

En este capítulo se explican los fundamentos básicos sobre los que se ha desarrollado el proyecto. Para ello se hace un análisis detallado de todas estas tecnologías en el contexto que engloba este proyecto. Primero se hace un análisis detallado sobre el ApR, extendiéndonos en el algoritmo QL. Posteriormente se muestran avances actuales sobre el uso del algoritmo QL con redes neuronales. A continuación, se explica la tecnología para la definición y lectura de los mundos. Por último, se hará un análisis del uso de la API REST para desplegar el proyecto, así como una pequeña introducción de las bases de datos relacionales.

3.1 ApR

El Aprendizaje por Refuerzo es un área del aprendizaje automático inspirada en la psicología conductista. Trata de simula el aprendizaje de sistemas biológicos reales en un entorno determinado a partir de acciones que devuelven una recompensa o refuerzo que puede ser bueno o malo.

Las recompensas o refuerzos sirven para definir políticas óptimas en procesos de decisión de Markov (MDPs). Una política óptima es una política que maximiza la recompensa total esperada. La tarea del ApR es utilizar las recompensas observadas para aprender una política óptima para el entorno donde el agente se encuentra.

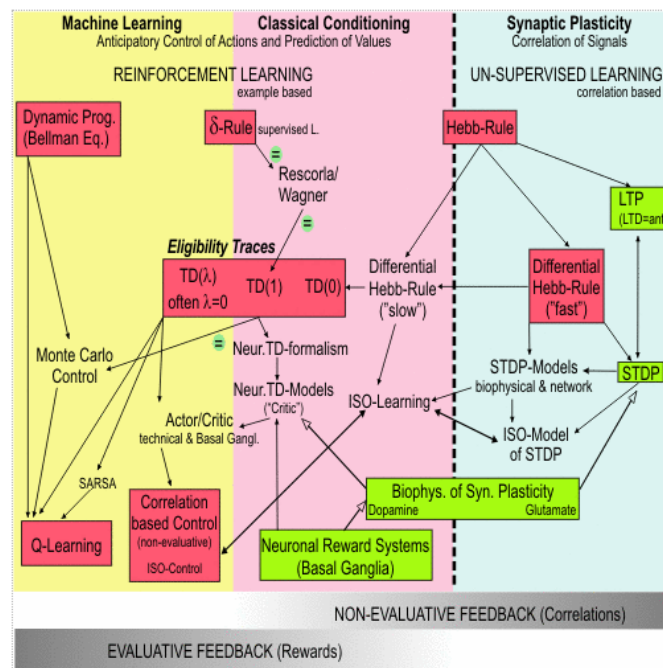


Figura 3.1: El contexto del ApR.

3.1.1 Historia del ApR

La Fig. 3.1 muestra las distintas disciplinas académicas que han contribuido al ApR. Las más importantes son dos:

1. Control óptimo (parte izquierda).
2. Aprendizaje animal por prueba y error (parte derecha).

Los problemas de control óptimo han sido llevados a cabo por la programación dinámica (Bellman, 1957). El aprendizaje por prueba y error tiene sus raíces en la psicología, especialmente en el condicionamiento clásico e instrumental. Como consecuencia, el control óptimo estuvo desde el principio gobernado por enfoques altamente algorítmicos o matemáticos, mientras que para el aprendizaje animal tardaron mucho más tiempo en desarrollarse los primeros modelos matemáticos.

El control óptimo y el condicionamiento instrumental trataron con problemas de control basados en retroalimentación. Sin embargo, el condicionamiento clásico trató con problemas de predicción única porque la respuesta del animal no influenciaba el experimento, es decir, no influenciaba el entorno. Un buen resumen que relaciona los enfoques algorítmicos con los experimentos reales de condicionamiento clásico fue dado por Balkenius y Moren en 1998.

A partir del estudio interdisciplinario de estos dos campos, apareció un método computacional muy influyente, llamado método de aprendizaje de Diferencia Temporal (TD) (Witten 1977, Sutton y Barto 1981). El aprendizaje TD fue originalmente asociado al aprendizaje animal, donde un refuerzo de ocurrencia temprana, también denominado estímulo condicionado, necesita ser asociado con un estímulo incondicionado que ocurre más tarde creando una situación donde las diferencias temporales de una función de valor necesitan ser evaluadas. El objetivo de esta computación es asegurar que después del aprendizaje, el estímulo condicionado se convierta en un predictor del incondicionado.

Mientras que el aprendizaje TD fue originalmente diseñado para tratar con los problemas de predicción, también fue usado para resolver problemas de control óptimo. De particular interés es el trabajo de Watkins (1989) sobre Q-Learning, un algoritmo de control basado en la diferencia temporal.

Fue esencialmente el trabajo de Klopff (1972, 1975, 1982, 1988), que comenzó a acercar los métodos TD a las teorías de aprendizaje animal. También introdujo la

diferencia entre retroalimentación evaluativa y no evaluativa, donde asoció la retroalimentación evaluativa a un entorno que no produce ninguna evaluación. La retroalimentación que llega del entorno a los sensores de la criatura solo puede ser no evaluativos. Toda evaluación, en este caso, debe ser realizada solo internamente por el mismo animal. Ya que los animales no reciben retroalimentación evaluativa, ApR parecería ser un ejemplo de aprendizaje sin supervisar. Sin embargo, esta formulación oculta el problema de cómo definir realmente el entorno.

Los métodos de aprendizaje TD necesitan predecir recompensas futuras. Con el fin de alcanzar esto, el aprendizaje TD usa funciones de valor $V(t)$ que asignan valores a estados y entonces calcula el cambio de estos valores por medio de una derivada temporal. Como consecuencia, estos métodos están relacionados con los métodos de correlación basados en el aprendizaje diferencial de Hebb (en la parte derecha de la Fig. 3.1), donde un peso sináptico cambia por la correlación entre sus señales de entrada con la derivada de su salida. Tales reglas fueron principalmente discutidas por Kosco (1986) y Kopf (1986, 1988).

3.1.2 Proceso de Decisión de Markov

El agente en la toma de decisiones sigue un Proceso de Decisión de Markov (MDP). Un MDP proporciona un marco matemático para modelar la toma de decisiones en situaciones donde los resultados son parcialmente aleatorios y parcialmente están bajo el control de un responsable de la toma de decisiones.

Un MDP es un proceso de control estocástico en tiempo discreto. En cada paso de tiempo el proceso está en algún estado s , y el agente que toma las decisiones puede elegir cualquier acción a que esté disponible en s . El proceso responde en el paso siguiente moviéndose aleatoriamente a un nuevo estado s' y dando al agente una recompensa correspondiente $R_a(s, s')$.

La probabilidad de que el proceso se mueva a su estado s' está influenciado por la acción elegida. Específicamente, está dado por la función de transición de estado $P_a(s, s')$. Por lo tanto, el próximo estado s' depende del estado actual s y la acción a del agente que toma las decisiones. Por lo tanto s y a son condicionalmente independientes a todos los estados y acciones anteriores; es decir, las transiciones de estado de un MDP satisfacen la propiedad de Markov. Un proceso estocástico tiene la propiedad de Markov

si la distribución de probabilidad de los estados futuros del proceso depende sólo del estado actual, y no también de la secuencia de eventos que lo precedieron.

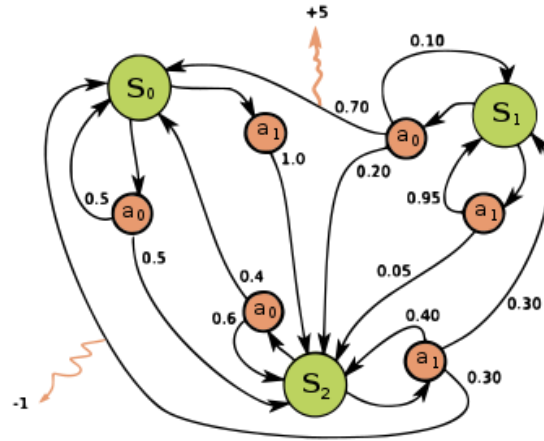


Figura 3.2: Ejemplo de un MDP simple.

En Fig 3.2 podemos observar un ejemplo de un MDP con tres estados (círculos verdes) y dos acciones (círculos rojos), con dos recompensas (flechas).

Un proceso de decisión de Markov es una 5-tupla $(S, A, P.(\cdot, \cdot), R.(\cdot, \cdot), \gamma)$ donde:

- S es un conjunto finito de estados.
- A es un conjunto finito de acciones.
- $P_a(s, s')$ es la probabilidad de que la acción a en el estado s en el momento t conduzca al estado s' en el momento $t + 1$. Donde $\sum_{s' \in S} P(s', s) = 1$.
- $R_a(s, s')$ es la recompensa inmediata recibida después de la transición del estado s al estado s' debido a la acción a .
- γ es el factor descuento, que representa la diferencia en importancia entre las recompensas futuras y las recompensas actuales.

El problema central de los MDPs es encontrar una política para el agente que toma las decisiones, es decir, una función π que especifica la acción del agente que toma las decisiones en el estado s , denotándolo como $\pi(s)$.

El objetivo es elegir una política π que maximice alguna función acumulativa de recompensas denominada recompensa acumulativa. Para hacer esto el agente combina la recompensa inmediata con otras recompensas futuras. Esta recompensa acumulativa para

que esté formada por recompensas mixtas, es decir futuras e inmediatas, tenemos que aplicarle un factor descuento γ ya que cuanto más futura sea una recompensa, menor valor tendrá. Entonces, a esta función V se le llama recompensa descontada:

$$V = \sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) \quad (3.1)$$

Donde se elige $a_t = \pi(s_t)$.

3.1.3 ApR Activo

Tal y como se explica en el capítulo 1, el ApR tiene 3 tipos de agentes. En el presente TFG se toma como agente ApR el agente QL debido a que no necesita un modelo del entorno.

El agente QL tiene que aprender un modelo completo con salidas probabilísticas para todas las acciones, en lugar de tener que aprender sólo el modelo para la política fija.

Hay que tener en cuenta que el agente tiene que realizar la elección entre las acciones. La utilidad de un estado es el valor que tiene ese estado para que el agente lo tenga en cuenta en la toma de decisiones. En Fig. 3.3 podemos ver un entorno con las utilidades de cada estado. Las utilidades que necesita aprender son las definidas por la política óptima, que siguen la ecuación de Bellman basada en la función de utilidad:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s') \quad (3.2)$$

Donde $R(s)$ es la función de recompensa, que especifica la recompensa de cada estado, $T(s, a, s')$ es el modelo de transiciones, que especifica la probabilidad de alcanzar el estado s' a partir de estado s después de realizar la acción a y γ el factor de descuento.

| | | | | |
|---|-------|-------|-------|---------------|
| 3 | 0,812 | 0,868 | 0,918 | <div>+1</div> |
| 2 | 0,762 | | 0,660 | <div>-1</div> |
| 1 | 0,705 | 0,655 | 0,611 | 0,388 |
| | 1 | 2 | 3 | 4 |

Figura 3.3: Entorno con la utilidad de cada estado.

Esta ecuación se resuelva para aprender la función de utilidad U a partir de los algoritmos de iteración de valor. Cuando se obtiene tal función de utilidad U , el agente puede tomar una acción óptima a partir de un paso de mirar hacia adelante para maximizar la utilidad esperada.

Exploración

Un agente que solo sigue las indicaciones de una política óptima para el modelo aprendido en cada paso no aprende las utilidades verdaderas o la política óptima verdadera. A este agente se le denomina agente voraz, y pocas veces converge en la política óptima para este entorno mientras que otras veces converge en políticas francamente horribles.

El problema viene de que el modelo aprendido no es el mismo que el entorno real, porque el agente no sabe cuál es el entorno real, por lo que no puede calcular la acción óptima apropiada al mismo.

Este problema se soluciona cuando el agente adopta una estrategia basada en un equilibrio entre la explotación, que se basa en maximizar la recompensa, y la exploración, que tiene como objetivo maximizar su comportamiento a largo plazo.

Por lo tanto, un agente tendría que ser voraz en el límite de infinitas exploraciones, o GLIE (Greedy in the Limit with Infinite Exploration). El esquema GLIE es un método de exploración que se basa en que el agente intente cada acción en cada estado dado un número ilimitado de veces, para evitar tener una probabilidad finita de que una acción óptima se pierda por una mala serie de resultados atípicos. Sin embargo, también es necesario que este esquema se convierta con el tiempo en voraz para que las acciones se conviertan en óptimas respecto al modelo aprendido.

Esquemas GLIE:

- El agente toma una acción aleatoria durante una fracción $1/t$ de tiempo y que sea voraz en el resto de los casos. Tiene el inconveniente de que el aprendizaje puede ser demasiado lento.
- Proporcionar un peso mayor a las acciones que el agente no ha intentado muy a menudo, evitando a la vez las acciones que tienen baja utilidad.

Se denota $U^*(s)$ como la estimación optimista de la utilidad del estado s siguiendo un esquema GLIE. Sea $N(a, s)$ el número de veces que la acción a se ha llevado a cabo en s , se actualiza la ecuación (3.2) para añadir la estimación optimista:

$$U^x(s) \leftarrow R(s) + \gamma \max_{a'} f(\sum_{s'} T(s, a, s') U^*(s'), N(a, s)) \quad (3.3)$$

Donde $f(u, n)$ es denominado función de exploración, donde u es la utilidad y n son las acciones que no se intentan a menudo. Esta función determina la relación entre la explotación y la exploración. Un ejemplo de esta función es la siguiente:

$$f(u, n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{en otro caso} \end{cases} \quad (3.4)$$

Donde R^+ es un pronóstico optimista de la mejor recompensa posible obtenible en cualquier estado y N_e es un parámetro fijo. La aplicación de esta función produce que el agente intente cada par estado-acción como mínimo N_e veces.

El uso de U^* hace que los beneficios de la exploración se propaguen recíprocamente desde los bordes de las regiones sin explorar, por lo tanto, los movimientos hacia regiones inexploradas tienen mayor valor. En Fig. 3.4 se puede observar que siguiendo esta estrategia se consigue una convergencia rápida a un rendimiento óptimo.

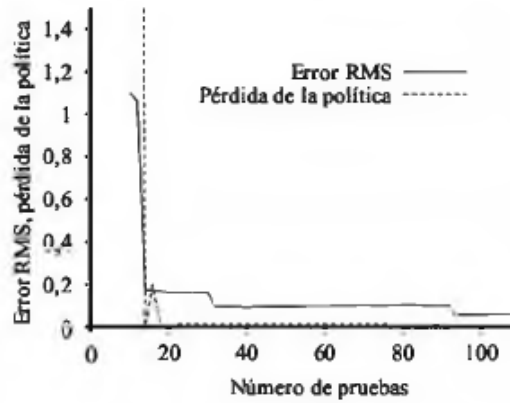


Figura 3.4: error valor cuadrático medio (RMS) en los valores de la utilidad y la pérdida de la correspondiente política.

Función Acción-Valor

El cambio más característico de un agente de aprendizaje pasivo a uno activo es que el agente no está equipado con una política fija, por lo que si aprende una función de utilidad U , tendrá que aprender un modelo que pueda elegir una acción basada en U a través de un paso de mirada hacia adelante.

Existe un método alternativo llamado aprendizaje-Q o Q-Learning que aprende una representación acción-valor en vez de aprender utilidades. Esta función-Q se denota como $Q(a, s)$ y presenta el valor de realizar una acción a en el estado s . Los valores de utilidad están relacionados de forma directa con los valores-Q:

$$U(s) = \max_a Q(a, s) \quad (3.5)$$

Por lo tanto, la función-Q es otra forma de almacenar información de utilidad y, además, el agente no necesita un modelo ni para el aprendizaje ni para la selección de acciones. Por esto se dice que el aprendizaje QL es un método libre de modelo. Habiendo introducido la función-Q, se puede actualizar la ecuación de Bellman de la siguiente forma:

$$Q(a, s) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(a', s') \quad (3.6)$$

Se puede usar esta ecuación de forma directa como ecuación de actualización para un proceso iterativo que calcule los valores-Q en un modelo estimado. Sin embargo, se necesita del aprendizaje del modelo, ya que en la ecuación (3.6) se usa $T(s, a, s')$. En un enfoque de diferencia temporal no hace falta ningún modelo, por lo tanto, será necesario sustituir $T(s, a, s')$ por una velocidad de aprendizaje α actualizando la ecuación (3.6) a la introducida en el capítulo 1 (1.1). Como se menciona brevemente en el capítulo 1, α es la velocidad de aprendizaje y γ es el factor de descuento:

Velocidad de aprendizaje

Es un valor entre 0 y 1 que indica cuanto puede el agente aprender de cada experiencia. 0 significa que no aprende nada y 1 significa que olvida todo lo que sabía ahora y solo se acuerda de la nueva experiencia. Para mejorar el aprendizaje se recomienda un valor dinámico donde $\alpha = 1/n$, siendo n el número de veces que se ha hecho una determinada acción en un estado dado, también definido por la función $N(s, a)$.

Factor de descuento

Es también un valor entre 0 y 1 que indica como de importante es una acción a largo plazo. 0 significa que solo importan los refuerzos inmediatos, y 1 los refuerzos a largo plazo. Por lo tanto, este factor ayuda a mezclar recompensas directas con recompensas a largo plazo dando lugar a una recompensa mixta.

3.1.4 Algoritmo QL

Uno de los primeros avances en el ApR fue el desarrollo del algoritmo QL (Watkins, 1989), definido por la ecuación (1.1).

```
1 Inicializar  $Q(s,a)$ , inicializar  $N(s,a)$ ,  $\forall s \in S$ ,  $a \in A(s)$ , y  $Q(\text{estadoterminal}, *) = 0$ 
2 Repetir (para cada episodio):
3   Inicializar  $S$ 
4   Repetir (para cada paso del episodio):
5     Elegir  $A$  de  $S$  usando una política derivada de  $Q$ 
6     tomar una acción  $A$  y observar  $R, S'$ 
7     incrementar  $N(S,A)$ 
8      $Q(S,A) \leftarrow Q(S,A) + \alpha(N(S,A)) [R + \gamma \max_{A'} Q(S',A') - Q(S,A)]$ 
9      $S \leftarrow S'$ 
10  hasta que  $S$  sea terminal
```

Figura 3.5: Pseudocódigo del algoritmo QL.

En este caso, la aprendida función- Q , aproxima directamente Q_* , la función- Q óptima, la cual es independiente de la política que es seguida. Esto simplifica dramáticamente el análisis del algoritmo y permite pruebas de convergencia temprana. La política todavía tiene un efecto que determina cuales pares estado-acción son visitados y actualizados. Sin embargo, todo lo que se requiere para una correcta convergencia es que todos los pares continúen siendo actualizados. El pseudocódigo del algoritmo QL se puede observar en Fig 3.5.

3.2 Deep Q-Learning

En la actualidad, el ApR se ha llevado más allá que a la simple búsqueda de una política óptima en un entorno simple. El primer modelo de ApR con redes neuronales, llamado Deep Q-Learning en inglés, es capaz de aprender políticas de control directamente a partir de entradas sensoriales de altas dimensiones. Estas entradas sensoriales de altas dimensiones son nada más y nada menos que los píxeles en bruto del entorno donde se encuentra el agente.

El modelo de redes neuronales puede ser una red neuronal convolucional, entrenado con una variante de QL cuya entrada son píxeles y la salida es una función valor estimando recompensas futuras.

Este método se ha aplicado a algunos juegos Atari 2600, como los de Fig. 3.6, superando a expertos humanos en varios de estos.



Figura 3.6: Capturas de pantalla de 5 juegos Atari 2600.

El aprendizaje de agentes de control a partir de entradas sensoriales de altas dimensiones es uno de los grandes desafíos del ApR. Las aplicaciones ApR más exitosas que operan en estos dominios se han basado en características hechas a mano con funciones de valores lineales. Por lo tanto, el rendimiento de tales sistemas depende de la calidad de la representación de las características.

Avances recientes en aprendizaje profundo han hecho posible extraer características de alto nivel a partir de datos sensoriales, conduciendo a avances en visión computacional y reconocimiento de voz. Estos métodos utilizan un rango de arquitecturas de redes neuronales, incluyendo redes convolucionales, perceptrones multicapa, entre otros.

Sin embargo, ApR presenta distintos desafíos a partir de la perspectiva del aprendizaje profundo. Primeramente, la mayor parte de las aplicaciones exitosas en aprendizaje profundo hasta la fecha han requerido una gran cantidad de datos clasificados como conjuntos de entrenamiento. Por otra parte, los algoritmos RL deben ser capaces de aprender a partir de una recompensa escalar que es frecuentemente escasa, ruidosa y atrasada en el tiempo. Otro problema es que la mayoría de los algoritmos de aprendizaje profundo asumen que las muestras de datos son independientes, mientras que en ApR uno encuentra normalmente secuencias de estados altamente correlacionados.

El algoritmo Deep Q-Learning demuestra que una red neuronal convolucional puede llegar a sobrepasar estos desafíos aprendiendo políticas de control exitosas a partir de datos de video en entornos de ApR complejos. La red es entrenada con una variante

del algoritmo Q-learning, con el Gradiente Descendente Estocástico (SGD) para actualizar sus pesos.

3.3 Videojuego basado en Celdas

Es un tipo de videojuego cuya área de juego está formado por imágenes cuadradas pequeñas, aunque éstas también pueden tener forma rectangular o hexagonal.

Una característica visual esencial es que las personas que juegan no pueden distinguir la división del mapa, es decir, las celdas es básicamente una distinción técnica.

En el área de juego se usa un conjunto de imágenes denominado conjunto de patrones. Un ejemplo de conjunto de patrones se puede ver en Fig. 3.7.



Figura 3.7: Ejemplo de conjunto de patrones.

Los juegos formados por conjuntos de celdas suelen simular una vista de arriba hacia abajo, vista lateral o 2.5D del área del juego, y son generalmente bidimensionales. Como se puede observar en el ejemplo de la figura 3.8.

Los videojuegos en el rango de la época de los años 70 a los años 90 dependían de un hardware que tenía soporte nativo para mostrar pantallas en celdas con poca participación de la CPU.

Los videojuegos basados en celdas no son un género de videojuegos específico, si no que el término se refiere a la tecnología que usa un motor de juego para su representación visual.

Los motores basados en celdas permiten a los desarrolladores crear mundos complejos y grandes eficientemente y con recursos mínimos. Estos videojuegos suelen

usar atlas de texturas, la cual es una imagen que contiene una colección de imágenes más pequeñas, por razones de eficiencia. También guardan metadatos sobre las celdas, tales como las colisiones, daño, propiedades, etc.

3.3.1 Historia

Este modelo de videojuegos fue introducido por la marca Namco en su juego Galaxian en el 1979. El tamaño del patrón más común usado en estos videojuegos era de 8 x 8 píxeles. La consola de videojuegos Intellivision, lanzada en 1979, fue diseñada para usar gráficos basados en celdas. Ya que sus juegos tenían que poder almacenarse en cartuchos de videojuegos de hasta 4K de memoria.



Figura 3.8: Ejemplo de videojuego basado en celdas.

Los ordenadores personales tenían soporte hardware en forma de caracteres ASCII, generalmente con el propósito de mostrar texto, pero los juegos también podrían escribirse usando letras y signos de puntuación como elementos del juego. Los personajes de los juegos podrían ser cualquier gráfico del juego que encajase en una celda de 8x8 píxeles.

Este modelo de videojuegos se utilizó principalmente en géneros como plataformas y rol, llegando a su auge durante la era de las consolas de 8 y 16 bits.

La mayoría de los primeros juegos basados en celdas usaban una perspectiva descendente. Posteriormente la perspectiva de arriba hacia abajo evolucionó a un ángulo simulado de 45 grados permitiendo al jugador ver tanto la parte superior como la lateral de los objetos, para dar mayor sentido a la profundidad. A medida que los ordenadores avanzaban, las perspectivas isométricas y dimétricas comenzaron a predominar en los juegos basados en celdas.

3.4 Aplicación Web REST

Las aplicaciones web basadas en la Transferencia de Estado Representacional (REST) son una forma de proporcionar interoperabilidad entre sistemas informáticos en Internet. Es una técnica de arquitectura de software en red para sistemas distribuidos que aprovecha de las tecnologías y protocolos de la World Wide Web (WWW).

REST permite crear servicios y aplicaciones que pueden ser usadas cualquier dispositivo o cliente que procese HTTP, por lo que es mucho más simple y convencional que alternativas anteriores como SOAP y XML-RCP.

Hay que tener en cuenta que REST no es ni una especificación ni un estándar, sino una técnica de arquitectura basado en estándares como HTTP, URL y XML.

REST se definió en el 2000 por Roy Fielding, coautor principal también de la especificación HTTP. Considerando a REST como un Framework para construir aplicaciones web respetando HTTP. Fielding desarrolló este tipo de arquitectura de desarrollo web, para identificar los problemas que existían en la web y así poder comparar soluciones alternativas a esos inconvenientes y por lo tanto certificar que extensiones de protocolo no serían dañinas para las restricciones principales que aseguran el éxito de la web.

Las APIs REST contienen las siguientes características:

- Identificación de recursos: Los recursos deben ser totalmente accesibles, es decir deben poseer un URI de identificación única
- Manipulación de recursos: Uso de recursos HTTP a partir del acceso gracias a sus métodos Get, Post, Put, Delete, Patch.
- Metadatos para describir nuevos recursos: Usa métodos o tecnologías estándar como HTTP, URI, XML, JSON, etc.
- Comunicación sin estado: no guardas las transacciones entre el cliente y el servidor. Tiene como finalidad disminuir el tiempo de respuesta de invocación al servicio web.



Figura 3.9: Esquema Cliente-Servidor en API REST.

Mediante el uso de un protocolo sin estado y operaciones estándar, los sistemas REST apuntan a un rendimiento rápido, confiabilidad y la capacidad de crecer, al reutilizar componentes que se pueden administrar y actualizar sin afectar al sistema en su conjunto, incluso mientras se está ejecutando. En Fig. 3.9 se puede ver el esquema Ciente-Servidor en API REST.

En la implementación de una API REST, las URLs permiten acceder a cada página, sección o documento del sitio web. Cada página, sección o documento se denomina como recurso. Por lo tanto, el recurso es la información a la que queremos acceder independientemente de su formato.

Las URL son un tipo de Uniform Resource Identifier (URI) en inglés, que permiten identificar únicamente cada recurso.

Para manipular los recursos, HTTP concede los siguientes métodos para operar:

- GET: Consulta y lectura de recursos.
- POST: Creación de recursos.
- PUT: Editar recursos.
- DELETE: Eliminar recursos.
- PATCH: Editar partes determinadas de un recurso.

Sin embargo, durante los últimos años los desarrolladores web han usado únicamente los métodos GET y POST para realizar todas estas acciones. Esto ha podido ser debido al soporte de ciertos navegadores o al desconocimiento de los desarrolladores. Esto puede generar problemas en la API REST a la hora de crear las URLs.

Como último aspecto básico de la API REST, está Hypermedia. La definición y objetivo de Hypermedia es conectar mediante vínculos las aplicaciones clientes con las APIs, permitiendo a dichos clientes despreocuparse por conocer de antemano como acceder a los recursos. Hypermedia es útil por ejemplo para que el cliente no necesite conocer las URLs de los recursos, evitando realizar el mantenimiento de cada uno si en el futuro se producen cambios en la URLs.

3.5 Bases de datos relacionales

Una base de datos relacional es una base de datos que es utilizada como un conjunto de tablas operándola conforme con el modelo de datos relacional. Está formada por un conjunto de objetos que se utilizan para el almacenamiento, gestión y acceso a los datos. Como ejemplos de objetos hay tablas, vistas, índices, funciones, etc.

Una base de datos relacional particionada es aquella donde los datos se manejan repartidos en múltiples particiones o nodos. Esta separación de los datos es transparente para los usuarios de gran parte de las sentencias de SQL.

En estas bases de datos cada tabla representa una relación, cada fila de esta tabla una tupla y cada columna un atributo. Estas bases de datos deben contener una clave primaria que no puede ser nula y pueden tener una clave foránea que sirve para establecer relaciones con otra tabla.

3.6 Object-Relational Mapping

Es un modelo de programación que consiste en transformar las tablas de una base de datos en una serie de objetos que simplifiquen las operaciones de las bases de datos por parte del programador. SQL ha sido sin duda el lenguaje más usado para acceder a las bases de datos relacionales.

Con el uso de ORN se mejora mucho más la productividad ganando importancia a los datos de la aplicación y dejando de lado los datos de persistencia. Como ventaja tenemos un aumento de seguridad ante el ataque contra las bases de datos. Sin embargo, es negativo en aplicaciones dependientes del rendimiento ya que su curva de aprendizaje es muy elevada. Como ejemplo de ORM a continuación se menciona a SQLAlchemy.

3.6.1 SQLAlchemy

SQLAlchemy es un conjunto de herramientas de código abierto en SQL y un ORM para el lenguaje de programación Python. SQLAlchemy proporciona una interfaz general para crear y ejecutar código independiente de la base de datos sin necesidad de escribir sentencias SQL.

SQLAlchemy se puede usar con o sin las características de ORM. En Fig 3.10 se pueden ver algunos ejemplos de configuraciones de bases de datos con Frameworks web.





| web framework | None | Flask | Flask | Bottle |
|----------------------|--|--|--|--|
| Database interaction | SQLAlchemy Core | SQLAlchemy Core | SQLAlchemy Core + ORM | SQLAlchemy Core + ORM |
| database connector | (built into Python stdlib) | psycopg | psycopg | psycopg |
| relational database |  SQLite |  PostgreSQL |  PostgreSQL |  PostgreSQL |

Figura 3.10: Ejemplo de configuraciones de Frameworks web con bases de datos.

Entre las ventajas de SQLAlchemy, la más característica es que permite a los programadores usar Python en su proyecto para mapear desde el esquema de la base de datos hasta los objetos de Python de la aplicación. Otra ventaja es que no son necesarios conocimientos en SQL para manejar la base de datos.

Una gran idea es usar SQLAlchemy en una aplicación web como backend de base de datos. Además, en la creación de aplicaciones web con Flask o Bottle existen librerías de Python que sirven como puente entre la librería SQLAlchemy y estos dos frameworks.