CONTENTS

Framing the Picture

Wenchy Dutreuil

I. Abstract

Frames are an expressive and adaptable form of knowledge representation that can be used for automatic reasoning. This paper begins by describing what knowledge representation and reasoning is. There are numerous methods for knowledge representation and reasoning. One such method is a Frame, and these methods are not mutually exclusive. The paper then proceeds to explain that a Frame is a data structure used in knowledge representation and reasoning; so, in some cases logical inference can be combined with frames. Frames are commonly used in knowledge-based expert systems; so, the paper continues by exploring a case study on the use of frames in a simplified expert system. Currently expert systems are used for decision support, but their power and accuracy has led me to believe that they will begin to play a primary role in decision making.

II. Introduction

Artificial intelligence is a field of study that is concerned with developing intelligent agents. Agents in this context are computer systems that are able to perceive, through sensors, and act on their environments, through actuators 1. The environments the agents exist in can have different properties. Some common property descriptors are:

1) Static or dynamic
2) Fully or partially observable
3) Stochastic or deterministic
4) Discrete or continuous
5) Episodic or sequential
6) Known or unknown

Static means that the state of the environment does not change, while dynamic means that it does. Fully observable means that the agent can perceive all parts of the environment at once, while partially means just part of the system is observable at a time. Deterministic means that actions have a known and guaranteed outcome, while stochastic means that actions in the environment have a set of potential outcomes each with a corresponding probability of success. Episodic means that a previous action does not affect a future action, while sequential means previous actions do affect future actions. Finally known describes whether the designer or agent knows about the environment before it is placed in it, while unknown means it does not. Depending on the characteristics the design of the agent will change.

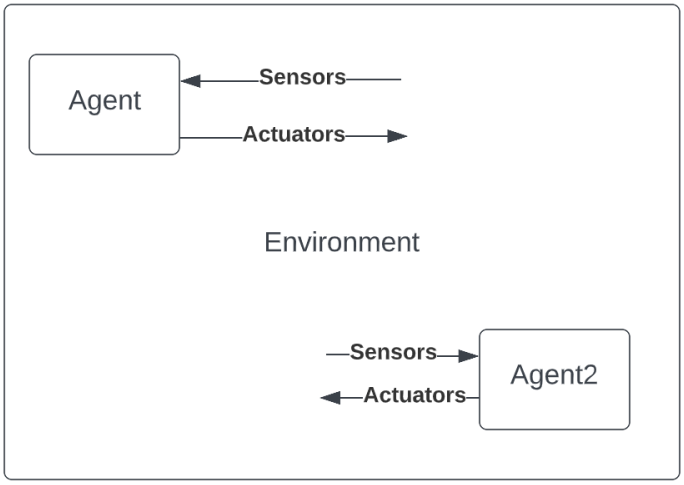

Fig. 1. Model of a typical formulation for intelligent agents in an environment. The environment can have one or more agents, and several properties.

Knowledge representation and reasoning (KRR) is a subfield of AI dedicated to representing information about the world in a form that a computer system can use to solve complex tasks. These agents are typically best suited for static fully observable deterministic discrete episodic known environments, but are used in other environments. This is because their optimal environment is not typical of the 'real' world. The real world is dynamic, partially observable, stochastic, continuous, sequential, and (partially) known/unknown. The world is complex; so, this usually results in the addition of constraints on the model of the world that the agent interacts with.

The agents built primarily around this paradigm are often referred to as knowledge-based agents. The central component of a knowledge-based agent is its knowledge base, or KB [9]. A knowledge base is a set of sentences; each sentence is expressed in a knowledge representation language and represents some assertion about the world [9]. Deriving information that is implied by the information that is already present is a form of reasoning [3]. Decision procedures constitute another component of reasoning [4]. A challenge to work in KRR is its complexity; a "minimal" common-sense system must "know" something about cause-and-effect, time, purpose, locality, process, and types of knowledge. It also needs ways to acquire, represent, and use such knowledge [6].

Knowledge can be categorized in many ways. Two characterizations that will be significant to us are declarative/descriptive and procedural knowledge [3]. These types of knowledge can be encoded in many different forms, such as: frame networks, declarative languages (logics), imperative languages (Java, Python, etc.), semantic networks, product rules,

neural networks, genetic algorithms, etc. [3]. Each knowledge representation format has its advantages and disadvantages.

One of the more common methods of knowledge representation is logic. Logic was the dominant paradigm in AI before the 1990s, but it had some drawbacks due to it being deterministic and rule based [10]. Despite these drawbacks it is very expressive and compact [10]. There are several different types of logics, such as: propositional, first-order, second-order, modal, fuzzy etc. [1]. We shall only concern ourselves with propositional and first-order logic. The goals of logical languages are to represent and reason about knowledge in the real world [10], there is a direct relation between the goals of KRR and logical languages.

In propositional logic there are propositional symbols and logical connectives (not, and, or, implication, bidirectional implication) [10]. A logical formula compactly represents a set of models where that formula is true [10]; for example, if we have propositional symbols P and Q, then P V Q represents all the worlds where P is true, or Q is true. In the logical paradigm, each sentence in a KB can be thought of as a logical formula that describes a set of models. For logical inferencing using propositional logic our agent can use either modus ponens or resolution [10]. Modus Ponens is sound and complete for propositional logic with horn clauses, and resolution is complete for propositional logic in general.

Unfortunately, resolution has exponential time complexity while Modus Ponens is linear [10]. But unfortunately, propositional logic is limited in its expressiveness; as a result, first-order logic adds variables, functions, and quantification. First-order logic has two types of quantifiers, universal and existential. The universal quantifier argues that every member of a group meets a condition, and the existential quantifier argues at least one member of a group meets a condition. If we also impose the restriction that there is a one-to-one mapping from object to constant symbol in first-order logic, then the consequence is this idea of propositionalization where first-order logic is just syntactic sugar for propositional logic and as a result we can use any inference algorithm for propositional logic on first-order logic [10].

Logic as a form of knowledge representation may seem very attractive. But, in relation to these systems, some people believe in simple cases one can get such systems to ”perform,” but as we approach reality the obstacles become overwhelming. The problem of finding suitable axioms–the problem of ”stating the facts” in terms of always-correct, logical, assumptions is much harder than is generally believed [6].

Another formalism for representing knowledge is referred to as a frame. A frame is a data-structure for representing a stereotyped situation, like being in a certain kind of living room, or going to a child’s birthday party [6]. We can think of a frame as a network of nodes and relations [7]. Collections of related frames are linked together into frame-systems. The effects of important actions are mirrored by transformations between the frames of a system [6]. The ”top levels” of a frame are fixed and represent things that are always true about the supposed situation. The lower levels have many terminals–”slots” that must be filled by specific instances or data. Each terminal can specify conditions its assignments must meet. (The assignments themselves are usually smaller ”sub-frames.”) Simple conditions are specified by markers that might require a terminal assignment to be a person, an object of sufficient value, or a pointer to a sub-frame of a certain type. More complex conditions can specify relations among the things assigned to several terminals [2, 7]. A frame’s terminals are normally already filled with ”default” assignments [6].

The frame system supports the so-called closed world inferring paradigm, where all facts that are presented in the system are true. If some fact is not presented, that means that it is untrue. It allows avoiding errors in inferring mechanisms related to the knowledge representation format [7]. Other formats as, for example, ontology, may support the open-world paradigm, where all facts that are not presented may also be true [7].

A frame-based knowledge base is one of the typical models or a part of such models for knowledge representation in expert and decision- making systems [7]. For example, in [5] the authors developed a question answering (QA) system using a textual KB constructed from a biology textbook, and in [4] the authors developed an emergency management system using a KB constructed from domain expertise. In general, the design of a knowledge model is based on a sequence of refinement steps, starting from a general valid reasoning method capable of meeting the goals of the target application [4]. Some applications of frame-based systems are emergency systems, machine translation, biomedicine, health care, probabilistic dialog systems, banking expert systems, natural language processing, question answering, information extraction/retrieval, classification, machine learning, robotics [7].

Despite the apparent separateness of the two discussed forms of KRR, logic and frames, they are not mutually exclusive. It is often the case that frame systems have their semantics defined as an extension to first-order logic. In addition, restricted and ad hoc forms of logical reasoning can be employed to derive new information. This coalescing of ideas helps to improve both systems.

III. Technical

IV. Frames

A method of knowledge representation in AI is known as a Frame or Frame Networks. Agents built using frames are referred to as knowledge-based agents or expert systems [9]. A typical design for knowledge based expert systems can be seen in figure 2. The central component of a knowledge-based agent is its knowledge base (KB) [9]. A knowledge base is a set of sentences; each sentence is expressed in a knowledge representation language and represents some assertion about the world [9]. As a result the main interface for knowledge-based agents are ASK, to add information to the KB, and TELL, to retrieve information from the KB [9]. In the case of frame based KBs the sentences are stored as frames. In a frame, all the information relevant to a particular concept is stored in a single complex entity, called a frame [2].

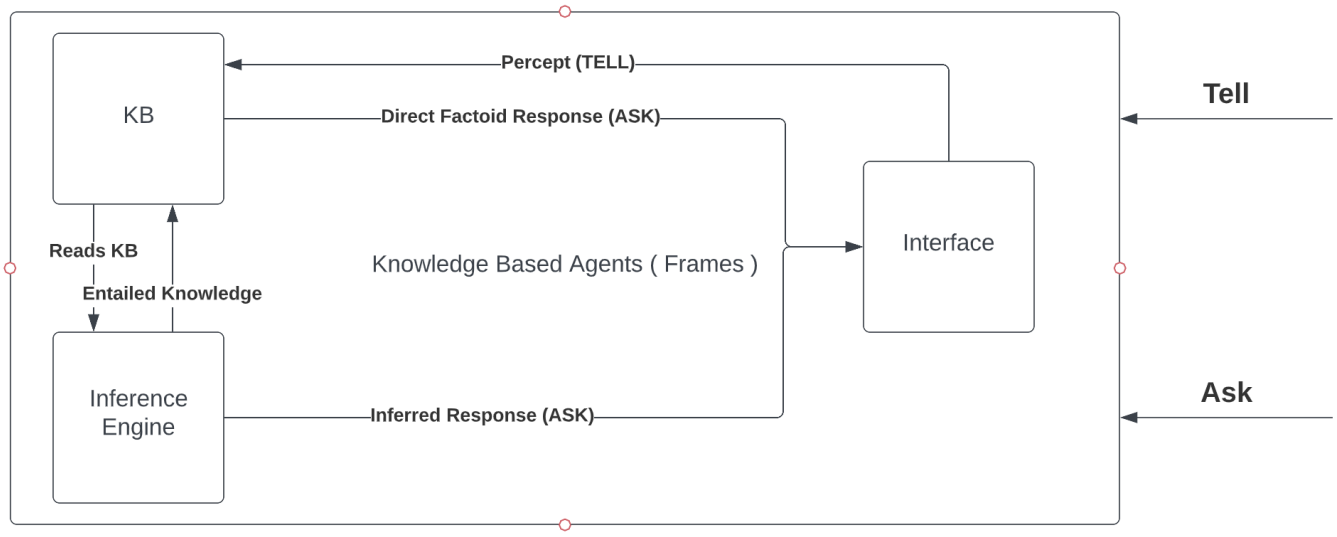

Fig. 2. A system design for a typical knowledge based expert system. The system has a ask and a tell api. When information is added to the KB or a query is submitted the system may call the inference engine to produce new information that is subsumed by the information currently in the KB.

Frames are structures that capture knowledge about a typical object, event, or relation. Typically frames are composed of an identifier, a set of superclasses, and a set of slots. For example a frame *Gray-Mammal* captures information about objects whose color slot is gray, like an elephant, and possess characteristics of mammals, such as warm-blooded . Frame based inference can take on different forms, but at the very least must support inheritance [2]. Frames can also support procedural based inference like product rules [6], [7] or general problem solving procedure as discussed in [8].

To elaborate on some of the aspects of frame systems we will discuss a specific implementation, the source code can be found at this repository. This frame based expert system is a *toy* example that helps track the freshness of food as time progresses. The example is intended to help in demonstrating the components of a frame based expert system. We will begin with a discussion on the interface used to interact with the system, we will then go on to the internal representation of the system, and finally we will discuss how the system reasons and derives new facts that are subsumed by the KB.

```
def main():
    kb = KnowledgeBase()
    line = input("> ").upper()

    while line != 'EXIT':
        interpret(kb, line)
        line = input("> ").upper()


if __name__ == "__main__":
    main()
```

Fig. 3. Main entry-point of the KB system. Reads input from the user and then interprets that input to change the state of the KB and/or output information for the user.

V. Interface

The interface for the system is a command line interpreter that reads one line at a time, interprets the line, then mutates the state of the system based on the input as shown in figure 3. There are two different means of knowledge acquisition manual and automatic [7]; this implementation is a manual form of knowledge acquisition. In fact, the most common way of entering knowledge into the frame system is manual, i. e. a knowledge engineer enters facts and assertions about the domain [7].

The language for the interface is specified using extended backus-naur(EBNF) as shown in figure 4. The table I is of valid sentences and their effects on the system. Further explanation of the language used to interface with the system is beyond the scope of this paper.

Sentence	Effects
tell add class day [number:{number}]	A percept sentence using the tell interface to add information to the knowledge base. The sentence instructs the system to add a class frame named day, with no super-classes, a slot number, and a facet of number. The name simply serves as an identifier. The set of super-classes are for the purpose of inheritance. The slot is for the purpose of information storage and reasoning. Finally, the facets, which are not always present, are for the purpose of enforcing constraints and enabling inference.
tell add instance day_1 day []	A sentence that adds an instance frame with the name day_1 which inherits from the day class frame. This causes the day_1 frame to implicitly add the number slot and facet as well, which lets us know that days are numbered.
ask day_1	Asks the KB for information about a frame.
ask day_1 slot number	Asks the KB for information about a slot on a frame.

TABLE I
TABLE OF VALID SENTENCES AND THEIR EFFECTS ON THE SYSTEM

```
<krl> ::= "TELL" <percept> |
          "ASK" <str> <query>?

<percept> ::= <add_frame> |
              <remove_frame> |
              <update_frame>

<add_frame> ::= "ADD" ( "CLASS" | "INSTANCE" )
                <str> "{" <str_list> "}" "[" <slot_list> "]"

<remove_frame> ::= "DELETE" <str>

<update_frame> ::= "UPDATE" <str> (
                       <update_type> |
                       <update_name> |
                       <update_super> |
                       <update_slots> )

<update_type> ::= "TYPE" ("CLASS" | "INSTANCE")

<update_name> ::= "NAME" <str>

<update_super> ::= "SUPER" ( "ADD" | "REMOVE" ) <str>

<update_slots> ::= "UPATE" "SLOT" ( ( ( "ADD" |
                                        "REMOVE" |
                                        "ADD_VALUE" |
                                        "DELETE_VALUE" ) <str> ) |
                                        <str> ":" <str> )

<query> ::= ( "SUPERS" |
```

VI. Representation

The internal frame structure, as shown in figure 5, that was built consisted of a reference to the frames name, type, set of superclasses, set of sub-classes, and set of slots, which each consist of a set of facets. The name field is used to uniquely identify the frames. The type field is used to partition the class and instance frames, so as to enforce the fact that only class frames can be instantiated. Instance classes cannot be re-instantiated. The superclasses and subclasses set are for managing inheritance and type based queries, such as *TYPEOF*. Finally slots are for storing information you know to be true about a frame and facets enforce constraints on these slots [6], [2], [5], [4].

```python
class Frame:
    INSTANCE = 'INSTANCE'
    CLASS = 'CLASS'

    def __init__(self,
                 frame_type: str,
                 frame_name: str,
                 superclasses: set = None,
                 slots: dict = None):

        if superclasses is None:
            superclasses = set()
        if slots is None:
            slots = {}

        self.name = frame_name
        self.type = frame_type
        self.superclasses = superclasses
        self.subclasses = set()
        self.slots = slots
```

Fig. 5. The code that defines the structure of a frame in the system. Frames have a name, type, set of superclasses, set of subclasses, and a set of slots. The slots themselves have a value/values and a set of facets.

VII. Inference

In terms of reasoning and inference, the most straightforward means of inference in frame systems is inheritance, the process of acquiring traits from a superclass [2], [6]. As in the previous example, an elephant is a subclass of *Gray-Mammal* and consequently it must have a color slot that takes on the value of gray. In addition to inheritance, this system's frames also support detached procedural inference [4], [7]. In our example we start with an empty KB and add information to it using the commands in the first input of figure 6.

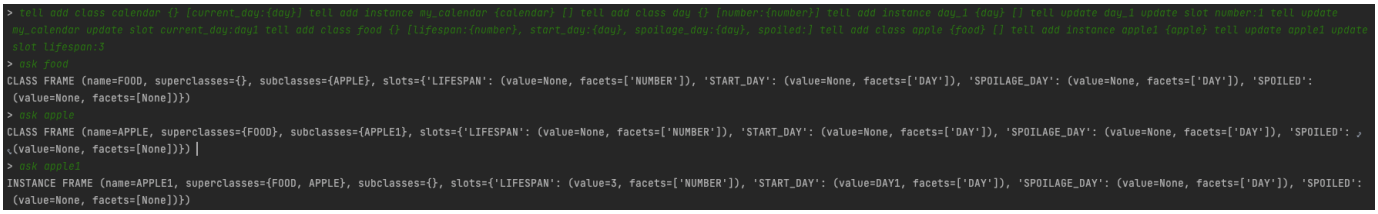

Fig. 6. A sequence of interactions with the KB using the aforementioned interface. Features both tell and ask operations. Note the fact that the frame Apple inherits properties from Food that weren't explicitly stated in the tell statement for the apple class. This is due to frame inheritance.

Then when the KB is queried it responds with the information it has stored, and has inferred due to inheritance. If you take a very close look at the input statement, you will notice a sequence of these four statements:

- tell add class food {} [lifespan:{number}, start_day:{day}, spoilage_day:{day}, spoiled:]
- tell add class apple {food} []
- tell add instance apple1 {apple}
- tell update apple1 slot lifespan:3

After we update the KB, the structure of the frames are as shown in figure 7. Despite not having explicitly added the slots lifespan, start_day, spoilage_day, or spoiled to apple1; when queried about it, the KB indicates that it has those slots. This is because apple1 was specified to be a subclass of apple which is a subclass of food. In addition, you may notice that the value of the slot start_day has been instantiated because without it ever being added to the KB explicitly. This is due to a stored procedure that updates the value of start_day when an instance of the food class is instantiated. In addition each time a day passes that lifespan is decreased, and then eventually when lifespan hits zero the food spoils and the spoiled slot is set to yes, as shown in figure 8.

VIII. Advantages and Limitations

Some advantages of frame-based systems are:

1) Programs can handle frames more easily
2) Makes the programming easier by grouping the related data
3) Flexible
4) Easy to include default data
5) Easy to modify
6) Easy to understand and visualize

Some limitations to frame-based systems are:

1) Inadequate representation of knowledge
2) Necessity to work with the completely know characteristics
3) Static knowledge domain
4) Representation of procedural knowledge as programming code inside frames
5) Complex structures can decrease the performance of the system inference and execution

[7].

IX. Conclusion

Frames are a common tool for knowledge representation in expert systems/ai. They are used to represent and reason about a stereotyped situation. Frame systems are expressive, but have some limitations as detailed in section VIII. Despite these limitations, frame-based representation of declarative and procedural knowledge has a wide application, and has recently been most prominent in healthcare and bio-medicine [7].

X. Reflection

During my time at CSBSJU, I have had the opportunity to take multiple challenging courses that helped me build my technical skills. I would say that all the courses helped helped prepare me for this, but the courses that I would say contributed the most to my success in this course are CSCI 340 (ORG OF PROGRAMMING LANGS), CSCI 200 (DATA STRUCTURES), and CSCI 331 (DATABASE SYSTEMS). CSCI 340 was a very interesting class that introduced me to the different ideas behind programming languages, in the class we discussed different programming paradigms and parsing methodologies. CSCI 200 was important because it introduced me to more structured and complex algorithmic problem solving. Finally, CSCI 331 was useful because it thought me ways to think about information and the structure of the information.

Specifically regarding my research topic on Frames and Knowledge Representation, CSCI 200 helped in just being able to understand frames as a complex data structure. CSCI 331 helped in understanding the field of knowledge representation in general. There exists different constraints that different attributes may have. CSCI 340 also helped because we discussed object oriented programming, which is one of the inspirations for frame based knowledge representation. Also in CSCI 340, I also learned a lot about parsing; so, I was able to build a front end for my knowledge based expert system.

This research was of benefit first and foremost because it thought me about knowledge representation and frames. So, I have this additional piece of knowledge to build my foundation as I continue to learn. In addition this research also helped me in better understanding some of the material from CSCI 317H (ARTIFICIAL INTELLIGENCE), which I was in while doing my research. This research also has helped improve my comfort and skill in terms of reading work in the field of computer science. This project was a challenging, but rewarding experience that helped me build skills that I believe will be valuable for the rest of my life and career.

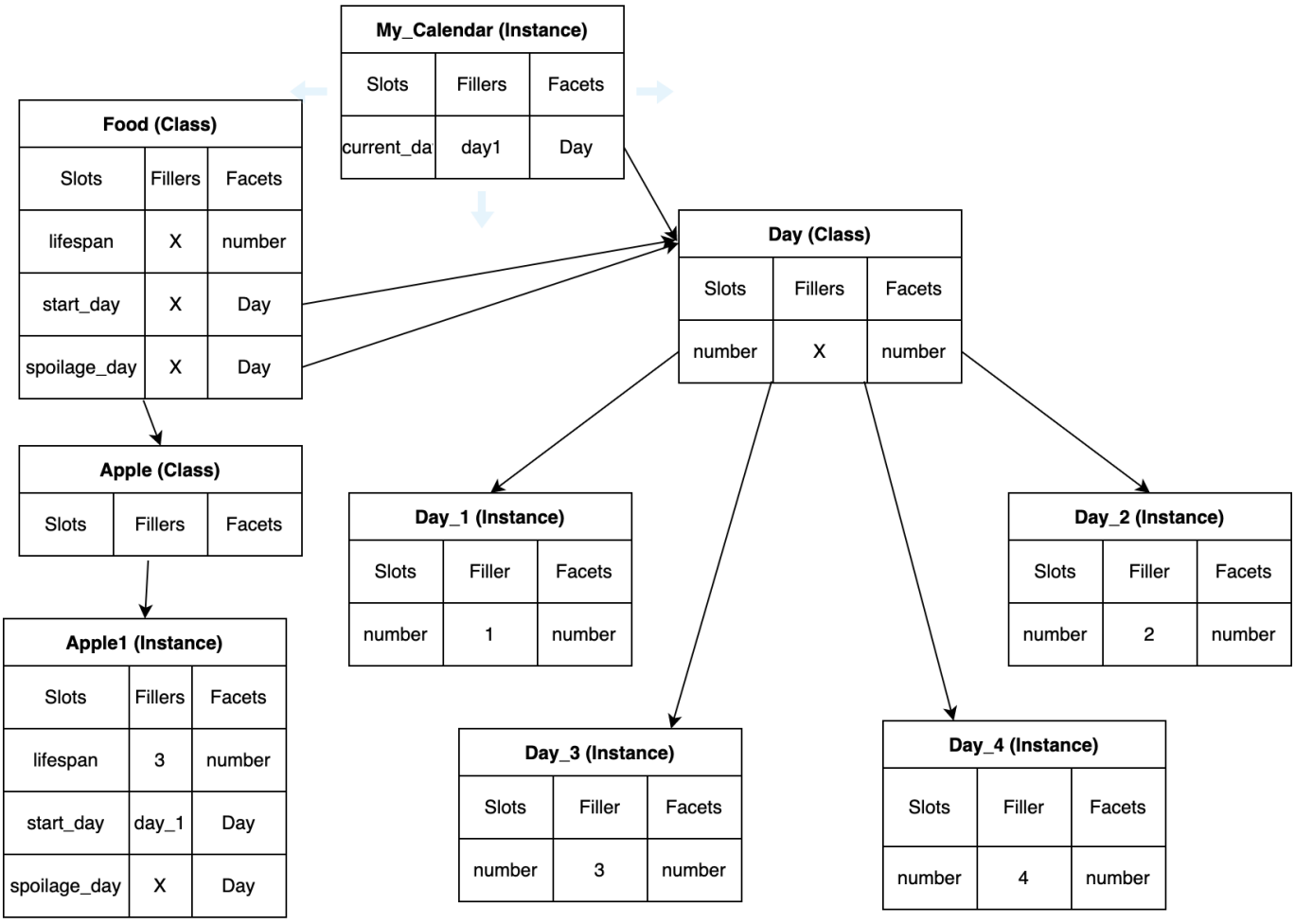

Fig. 7. A diagram of the frames in our system after the initial tell statements.

```
> tell add instance day_2 {day} [] tell update day_2 update slot number:2 tell update my_calendar update slot current_day:day2 tell add instance day_3 {day} [] tell update day_3 update slot number:3 tell update my_calendar
 update slot current_day:day3 tell add instance day_4 {day} [] tell update day_4 update slot number:4 tell update my_calendar update slot current_day:day4
> ask apple1 slot lifespan
(value=0, facets=['NUMBER'])
> ask apple1 slot spoiled
(value=YES, facets=[None])
> ask apple1 slot spoilage_day
(value=DAY4, facets=['DAY'])
```

Fig. 8. This shows a sequence of tell operations that update the KB calender's day, and then a sequence of ask operations that inspect the consequent state of the KB.

REFERENCES

[1] An introduction to formal logics, Jan 2016.

[2] Alison. Frames, Aug 1994.

[3] Vinay Chaturvedi. Understanding artificial intelligence and machine learning.

[4] Josefa Hernández and Juan Serrano. Knowledge-based models for emergency management systems. *Expert Systems with Applications*, 20, 02 2001.

[5] Peter Clark Vulcan Inc., Peter Clark, Vulcan Inc., Phil Harrison Vulcan Inc., Phil Harrison, Niranjan Balasubramanian University of Washington, Niranjan Balasubramanian, University of Washington, Oren Etzioni University of Washington, Oren Etzioni, and et al. Constructing a textual kb from a biology textbook: Proceedings of the joint workshop on automatic knowledge base construction and web-scale knowledge extraction, Jun 2012.

[6] Marvin Minsky. A framework for representing knowledge, Jun 1974.

[7] Vladislavs Nazaruks and Janis Osis. A survey on domain knowledge representation with frames. *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2017.

[8] James A. Reggia, Dana S. Nau, and Pearl Y. Wang. A new inference method for frame-based expert systems. In *Proceedings of the Third AAAI Conference on Artificial Intelligence*, AAAI'83, page 333–337. AAAI Press, 1983.

[9] & Norvig P. Russell, S. J. *Artificial Intelligence: A modern approach*. Prentice-Hall, 2010.

[10] stanfordonline. Logic 1 - propositional logic — stanford cs221: Ai (autumn 2019), Dec 2020.

APPENDIX

LIST OF FIGURES

LIST OF TABLES