**Comparing the Logic and the Object-Oriented Paradigms, and the Merging of the Two Paradigms**

**Abstract**

Object-oriented programming and logic programming are both programming paradigms that take different approaches to solving software-related problems. In this paper, these two paradigms, the benefits, and the applications of each paradigm are discussed. Finally, the integration of the two paradigms and the possible applications it can bring to the field is briefly discussed.

**Introduction**

In recent times, the demand for online services and information systems increases due the increasing usage of the internet throughout the world [1]. As the demand for online services increases, the tasks that these services need to execute will become more complex as a result. There are many different programming paradigms to give software developers the tools for approaching problems in many ways. The two paradigms that are will be looked at in this paper are the object-oriented paradigm and the logic paradigm. It is shown in this paper that each paradigm has its benefits and can prove to be useful when applied in many real-world scenarios.

The object-oriented paradigm is considered to be the industry standard for software development [1], while other paradigms like logic programming tend to be overshadowed by the object-oriented paradigm. This paper will hopefully shed some light on logic programming and show that the problems logic programming specializes in apply to many more real-world scenarios than people would think. Then the combination of the two paradigms in question is discussed and the possible benefits an object-oriented logic multi paradigm could bring to the field.

**A Brief History on the Object-Oriented Paradigm**

In 1962, Dr. Kristen Nygaard and Ole-Johan Dahl started to work on a project called Simula at Norsk Regnesentral in Oslo [1], resulting to the release of the project in 1964. Their approach to modelling reality was to look at reality as a number of processes, which contrasted with the approaches that earlier languages took on modelling reality. Then in the 1970s, Alan Kay develops the next object-oriented programming language called Smalltalk in the research laboratories of Xerox in Palo Alto (US).

Until 1984 the object-oriented paradigm was confined mostly to research laboratories, a few universities, some governmental agencies, and the artificial intelligence community [1]. This changed in the late 1980’s as the interest for object-oriented software development increased. Smalltalk played a major role in the increase of interest in Object-oriented programming, proving that the object-oriented paradigm is a complete programming paradigm [1].

**A Brief History on the Logic Paradigm**

In 1965, John Alan Robinson developed a deduction method called resolution, which was proposed as a uniform proof procedure for proving theorems. [2]**.** Resolution converted everything to clausal form and then used a method to attempt to obtain a proof by contradiction.Then in 1969, Carl Hewitt develops a language called Planner, a hybrid between the procedural and logical paradigms. [2]**.** Carl Hewitt himself states that Planner represented a rejection of the resolution uniform proof procedure paradigm [2]. In 1971, a subset of Planner, Micro-Planner, was implemented by Gerry Sussman, Eugene Charniak, and Terry Winograd as an extension to Lisp to save on memory space and processing time [2]. Shortly after in 1972, Roussel implemented a language called Prolog (designed by Alain Colmerauer and Robert Kowalski) into the field.Prolog implemented a number of non-logical computational primitives for input-output, etc. Like Planner [2].

**The Object-Oriented Paradigm**

The object-oriented paradigm is a programming style that is based on the modelling approach of the real world out of objects, classes, and inheritance [1]. Objects in OO software provide their respective services by providing object fields and object methods [3]. A field of an object is a binding of a name to an object, while a method of an object is a binding of a name to a function that performs a computation [3]. A class is a syntactic programming construct that mainstream OO languages offer for the specification of the common behavior shared by some objects that provide the same service [3]. Objects produced from a certain class are called class instances. [3] A class name is typically associated with a behavioral contract, i.e., with a specification of the service provided by instances of the class [3]. Object oriented programming is defined by two main features: encapsulation and inheritance [3]. Encapsulation is the bundling of object fields and methods, with the object methods having the ability to access and manipulate the object fields [3]. Because of this, encapsulation can also be used to hide the internals of an object. Inheritance is defined as a binary relation between classes in an OO program, where a class is said to inherit another class, where the inheriting class is called the subclass and the inherited class being called the superclass [3]. Inheritance enables software extensibility and software reuse, which can help avoid coding redundancies and reduce software development times [1].

Some common programming languages that are based on the object-oriented paradigm include Java, C++, Python, and Smalltalk. Unlike languages like Java, C++, and Python, Smalltalk is a pure object-oriented language. Languages like Java, C++, and Python, are not considered pure object-oriented languages because they include features that come from other paradigms, essentially making them multi-paradigm languages. What is the difference between a pure object-oriented language and languages like Java? First, Smalltalk’s grammar is very simple, often being compared to written sentences and “to the conventional human thinking” [4]. Languages like Java have more complex syntaxes, having code that is hard to understand from an everyday person’s perspective. Programmers often provide comments for their code so others can understand what the code accomplishes. Another difference is that Smalltalk uses dynamic data types, having the data type be determined during runtime, while Java has static data types, having the data type be determined at the time of declaration [4]. In Smalltalk everything is an object whereas in Java certain aspects of code, like primitive data types, are not considered objects. Programs in Smalltalk are “built on three basic operations: the message sending to the object, the object specification, and the object returning as a value.” [4]. This means that objects in Smalltalk communicate with one another to accomplish a task given by the program, where as Java invokes methods on objects to accomplish tasks given by programs.

**Applications of the Object-Oriented Paradigm**

Object-oriented programming languages like Smalltalk excel in modelling large and complex systems. They can use many different objects that each serve a different purpose in the system, making it much easier to keep track and manage all the entities in this paradigm as opposed to using other programming paradigms. With the feature of inheritance, object-oriented programming is also particularly useful for problems where objects/entities can be categorized hierarchically, which makes it useful for model building, computer-based simulations, and as a tool for knowledge representation in expert systems [5].

Developers from Servio Logic Corporation have proposed an idea to implement Smalltalk80 as a potential approach to coding their database systems. With an object-oriented language, objects can be represented as real-world entities that the databases are also trying to represent. This would increase the potential efficiency and the functionality of a database system, allowing entities to have functions of their own and hold certain values that other database systems could not account for. For instance, a regular database would only be allowed to represent attributes in the form of a set of limited types such as integer, real, string, and some special types like date while an object-oriented database would be able to represent a data type like “employee number” with a non-standard ordering [6]. An object-oriented database can avoid arbitrary limits that are imposed on other database systems, such as number of fields in a record, number of files, number of relations in a query, number of indexable fields, number of records in a file, and depth of repeating groups [6]. In other database systems, every instance of a certain type of entity must be identical in structure. Every attribute of an entity must follow the exact structure and must have all the same fields as other entities of that type and the only way that these databases systems can account for variability is by allowing certain fields to be nullable. This does not adequately support the complexities and variations that occur in real data [6]. For example, if an entity employee had 3 fields for the employee’s name, one for their first name, one for their middle name, and one for their last name, the database would not be able to account for the scenario where an employee might have an extra middle name. Using an object-oriented database would allow the addition of such an attribute without needing to restructure the entire database [6].

**The Logic Paradigm**

As Carl Hewitt states, “Logic Programming can be broadly defined as ‘using logic to infer computational steps from existing propositions’” [2]. Logic programs are composed of a set of statements called Horn clauses, stating what is true about a problem domain. A Horn clause is a sentence which can be written in the form:

“A0 ← A1 ∧ . . . ∧ An where n ≥ 0” [7].

Each Ai is an atomic formula of the form *p*(t1, … ,tm) where p is a predicate symbol and the ti are terms [7]. Each term is either a constant symbol, variable, or a composite term [7]. The backwards arrow “←” stands for “if” and the “∧” symbol stands for “and” [7]. A­0 is the conclusion of the clause and “A1 ∧ . . . ∧ An” is the body of the clause. The terms A1, …, An inside the body of the clause are called conditions. If n = 0, then the body of the clause is equivalent to true, making the clause “A0 ← true”, which can be abbreviated to “A0”, making the clause a fact. If n != 0, then the clause is considered a rule. So, if there are no conditions in the body of the clause then the entire clause is considered as a fact, and if there exists one or more conditions in the body of the clause then the clause is considered as a rule.

Prolog is one of the first and most widely used logic programming languages in the field [7]. However, Prolog itself is not a pure logic programming language as it allows the use of some imperative programming whenever using logic programming proves to be more inconvenient. Prolog allows impure predicates, which means that checking truth values for select predicates have the possibility of producing side effects, possible side effects can include modifying a global state or producing output on the terminal. Although Prolog is an impure logic language, there is a subset of Prolog called Pure Prolog. Pure Prolog still uses the same environment and tools as Prolog does, but programs written in Pure Prolog must adhere to the standard logic programming paradigm rules, meaning that programs are restricted to only using horn clauses and these clauses must not produce any side effects. Pure logic programming languages are not used as often as their impure counterparts because pure logic programming tends to be much more inefficient than impure logic programming. Paulson and Smith states that pure logic programs are “completely pure, regardless of the consequences for efficiency” [8].

**Applications of the Logic Paradigm**

Logic programing languages like Prolog are well suited for dealing with fields that involve decision making including computational linguistics, database research, artificial intelligence, and knowledge systems [5]. The ability for logic languages to use deductive reasoning through the means of using first-order logic and a given knowledge base makes them especially useful when working with computational linguistics and artificial intelligence, where both fields are constantly required to make decisions or inferences for machines/mechanisms to act upon. Speech recognition, a subfield of computational linguistics, often requires the software to make assumptions whenever the software can not understand what the user is saying. The software will decide what is best to put down on text based on the given knowledge base and the sound the user has provided.

A poster from the Logic Programming and Nonmonotonic Reasoning conference was presented on the topic of using defeasible logic programming to give a robot the ability to perform deductive reasoning. Defeasible logic programming slightly differs from logic programming, with the addition of a new set of predicates, being defeasible rules. Defeasible rules are rules that only hold for certain scenarios, while regular rules hold for all scenarios. So instead of a set of facts and rules, defeasible logic programs consist of a set of facts, strict rules, and defeasible rules. The developers have created a scenario in which a robot is in an enclosed space, many boxes of different sizes (small, medium, and big) are scattered around in the enclosed space, and a designated spot is placed in a corner where the boxes need to be moved to. Since the robot has no way of knowing the status of the battery that is powering it, the robot must make decisions to find the best way to store all of boxes in the designated spot [9]. To apply defeasible logic programming in this situation, the three sets mentioned above can be translated into more meaningful concepts. First, facts in this case will be used for representing perceptual information about the environment. These statements will define what is considered close to the robot, what is far away from the robot, and what size a certain box is. Strict rules will represent information that cannot change regardless of changes made to the environment. These statements include rules to help compare and determine all the possible scenarios when observing any 2 boxes. Finally, defeasible rules will represent the robot’s preferences about which box to select in different situations. These are a list of conditions that cover every single possible scenario that the programmers think that the robot will encounter during execution.

Using these sets of predicates, the developers demonstrate how a robot in this given environment will use deductive reasoning to find the optimal order of moving a set of boxes to a designated area. The robot will use the given facts and strict rules to decide which set of defeasible rules to use, depending on the surrounding environment that it is currently in. Logic programs use a set of predicates, in the form of facts and rules, and logic to conclude what to do and is the reason why the developers decided that using a logic programming approach would be best for this scenario. The developers have presented an application of logic programming where cleaning tasks are given to real and simulated robots [9].

**Combining the Two Paradigms Together**

Both object-oriented and logic programming have their benefits and specialize in certain fields of programming, but like any other paradigm, there are many things that each programming paradigm can not do. Object-oriented programs lack the ability to use deductive reasoning using a knowledge base, while logic programs lack the ability to provide the steps on how to perform a certain task that needs to be completed. On top of not being able to easily compute certain operations, logic languages like Prolog lack core features that the common object-oriented languages have, “such as the inexistence of standard libraries, standard foreign language interfaces, and powerful encapsulation mechanisms” [10]. Combining these two paradigms together would resolve the issue stated above, giving a means to deductive reasoning through the features of logic programming and providing the much-needed features from object-oriented programming. In such a paradigm that integrates both logic and object-oriented programming, “Objects will allow programmers to work with the same set of concepts in the successive phases of application development” and “Logic programming will allow programmers to represent the knowledge of each object.” [10]

A possible application with this multi-paradigm is the ability to represent clauses as objects. Representing clauses as objects would allow greater flexibility in creating, maintaining, and modifying large knowledge bases. First, these objects can store information other than the clause itself, like the creation date, the author, what assumptions were made for a clause, and what other clauses may have been used to derive this clause [5]. Koschmann and Evans states that “these kinds of documentation are important when creating and maintaining a large knowledge base” [5] since there are going to be a great number of clauses, there will be a high chance that clauses will hold some similarities and these other attributes will help distinguish one clause from another in knowledge bases. In addition to keeping documentation of each clause, objects can also store information “describing the deductive relationships between clause object” [5], enabling the ability to create truth-maintenance systems that keep track of the assumptions made to come to any given conclusion.

**Conclusion**

Both the object-oriented paradigm and the logic paradigm have their benefits and proved that they excel in solving problems in the fields they specialize in. Although logic programming is not as popular as object-oriented programming, the need for logic-based approaches to problems increases as more complex concepts like artificial intelligence, speech recognition, and many other decision-based tasks are introduced to the field. Although there do exist languages that incorporate both logic and object-oriented programming, the same drawbacks that hold logic programming back also holds these languages back from becoming mainstream languages.

**References:**

[1] P. Paetau, “On the Benefits and Problems of the Object-Oriented Paradigm Including a Finnish Study”, Swedish School of Economics and Business Administration, Finland, 2005.

[2] C. Hewitt, “Resolution, Planner, Prolog, and the Japanese Fifth Generation Project,” *Middle History of Logic Programming*, Dec. 2009.

[3] M. A. AbdelGawad, “NOOP: A mathematical model of object-oriented programming,” thesis, 2011.

[4] T. Bublik, “Comparison of Java and Smalltalk Programming Languages,” pp. 1–12, 2015.

[5] T. Koschmann and M. W. Evens, “Bridging the gap between object-oriented and logic programming,” *IEEE Software*, vol. 5, no. 4, pp. 36–42, 1988.

[6] G. Copeland and D. Maier, “Making Smalltalk a Database System,” *Proceedings of the 1984 ACM SIGMOD international conference on Management of data - SIGMOD '84*, 1984.

[7] R. Kowalski, “Logic Programming,” *Computational Logic*, pp. 523–569, 2014.

[8] L. C. Paulson and A. W. Smith, “Logic Programming, Functional Programming, and Inductive Definitions,” *Extensions of Logic Programming*, pp. 283–309, Mar. 2001.

[9] E. Ferretti, M. Errecalde, A. J. García, and G. R. Simari, “An Application of Defeasible Logic Programming to Decision Making in a Robotic Environment,” *Logic Programming and Nonmonotonic Reasoning*, pp. 297–302, 2007.

[10] P. Moura. “Logtalk - Design of an Object-Oriented Logic Programming Language,” PhD thesis, Department of Informatics, University of Beira Interior, Portugal, Sept. 2003.