STM JOURNALS

International Journal of
Data Structure Studies

Volume 2, Issue 2, 2024
July–December
DOI (Journal): 10.37591/IJDSS

https://journals.stmjournals.com/ijdss

Review IJDSS

Data Representation and Abstraction in Computer Systems: An Interactive Study

V. Basil Hans*

Abstract

Data abstraction simplifies data, while data representation stores or encodes it. Data abstraction in programming involves constructing a data type that hides data representation. This lets consumers concentrate on the data's primary features rather than its implementation. Data abstraction is common in object-oriented programming and database administration. Computers store data in binary format. The smallest binary unit is a bit, or "binary digit". Bytes typically have eight bits. Data abstraction with abstract data types (ADT) is ubiquitous in programming. A stack ADT may offer push, pop, and peek methods to add, delete, and analyze stack components without knowing the stack's internals. Front-end applications can query data regardless of storage location via data abstraction. This lets developers switch back-end databases without rewriting huge parts of code. Bank account information can be protected using data abstraction. Data abstraction prevents users from seeing how an ATM dispenses money or creates receipts. Computer systems need data representation and abstraction to manage and interpret information. This article explores data representation and abstraction in modern computers through interactive research. Data representation includes binary, hexadecimal, and other data kinds like integers, floating points, and characters. However, abstraction hides low-level features to simplify complex data structures and processes and allow users to engage with a system through higher-level interfaces. This study stresses how these notions affect computer system design and functionality, including hardware, operating systems, and software development. It addresses how procedural, data and control abstraction improve system performance, modularity, and scalability. The paper discusses real-world instances of abstraction making complicated systems like file systems and databases easier to operate. Interactive features like case studies and simulations highlight how data is represented and abstracted in different computing environments, emphasizing its importance in system design and user engagement. Data representation and abstraction are crucial to computing, revealing how they underpin modern computer architecture and processes, making technology more efficient and user-friendly.

Keywords: Data abstraction, data representation, data structures, interactive tools, binary number, abstract data type (ADT)

***Author for Correspondence**
V. Basil Hans
E-mail: vhans2011@gmail.com

Research Professor, Department of Management & Commerce, Srinivas University, Mangaluru, Karnataka, India

Received Date: October 20, 2024
Accepted Date: October 25, 2024
Published Date: November 07, 2024

Citation: V. Basil Hans. Data Representation and Abstraction in Computer Systems: An Interactive Study. International Journal of Data Structure Studies. 2024; 2(2): 22–31p.

INTRODUCTION

Data representation and abstraction underpin modern computer systems in computer science. Data and interaction management have become crucial as software and hardware systems have become more complicated. Abstraction simplifies and manages data representations, enabling user-computer interaction.

The encoding of data for computer storage and processing is called data representation. Computing uses binary digits (bits), although the data may be

integers, characters, or floating-point values. The representations of these data types greatly affect a system's performance, correctness, and efficiency. ASCII for text and IEEE 754 for floating-point integers demonstrate how data types are encoded for computation and system communication.

We study how to declaratively express combinations of data structures with complex sharing in a provably accurate code according to Hawkins et al. (2011) [1]. Our method specifies abstract data types (ADT) by using relational algebra and functional relationships. We provide a language of decomposition that allows users to express concrete representations for relations and demonstrate that operations on concrete representations correctly execute their relational specification. Our compiler synthesizes data representations that are easy to integrate into existing systems, resulting in simpler, more correct, and comparable-performing codes.

However, data abstraction hides the deep information of these representations from users, generating simpler models. Systems encapsulate complex processes and data so that consumers and developers may engage with them without understanding their mechanics. This allows for modular system design, where each abstraction layer can focus on certain functions without being swamped by low-level details.

Cima et al. (2023) [2] noted that data abstraction is relevant in numerous settings outside data service providers. Three are listed here. Abstraction can be used in ontology-based data management to validate whether mapping covers the required data services at the global schema level. Abstractions provide the semantics of open datasets and APIs published by organizations, which unlock the potential of open data. Finally, abstraction can be used to create a semantic-based source profile approach, another data-preparation task, to describe the structure and content of a data source in business jargon.

Neumann [3] stated that everything important for the construction and understanding of digital computers can be clarified by simple propositions that are easily understood. We discuss data as they are used in digital computers, emphasizing the fundamental issues surrounding data representation and use, leaving the technology that supports this representation implicitly.

The goal is to promote comprehension, help disperse the fog, and increase the understanding of people who now or will in the future use digital computers. For this to be useful, we must practice what we have preached.

An interactive study of data representation and abstraction in computer systems aims to clarify how these principles are implemented across hardware and software layers, and how they affect system performance, scalability, and ease of use.

This study will also use real-world examples and interactive case studies to show how data representation and abstraction form the backbone of modern computing. By understanding these principles, we can appreciate the intricacies of computer systems and how they have evolved to handle the ever-increasing complexities efficiently and elegantly.

DATA REPRESENTATION BASICS

Data, Items, and Structures

This study examined data in the context of computer-based information systems and linked them to the principal components that are manipulated, transferred, and stored in a computer-based information system: data items, data aggregates, and data structures.

Tsakalidis (1990) [4] stated that data structuring involves examining concrete implementations of commonly used ADTs. An abstract data type consists of a set and collection of operations that can be performed on its elements. For instance, in the dictionary data type, the set represents the power set of universe *U*, with operations including the insertion and deletion of elements, as well as membership

testing. A typical use of dictionaries is in symbol tables within compilers. In the case of priority queues, the set is again the power set of an ordered universe *U*, and the operations include inserting elements and locating and removing the minimal element from the set. There are three standard types of complexity analysis in data structures: worst-case, average-case, and amortized. The worst-case analysis determines the maximum time bounds for individual operations. Average-case analysis assumes a probability distribution over the operations of the abstract data type and computes the expected cost of operations based on this distribution. On the other hand, amortized analysis examines the worst-case cost across a sequence of operations.

Data Structure Classes

The reader is familiar with two classes of data structures in common use: scalar and composed data structures. Scalar data structures are not composed of other data structures. The only examples are individually valued primitive and abstract data structures because primitive data structures are generally implemented using scalar data structures. Abstract data structures are built upon other abstract data structures until they reach the lowest level and meet primitive data structures. The data structures can be divided into two classes. An occurrence data structure is designed to hold information about the actual occurrence of an abstract object; such a structure is often associated with some of the time the system provides a service or support to the objects described by the data structure. On the other hand, a definition data structure holds information that describes some or all the structural characteristics of a collection of objects. It is a separate data structure that includes instances of one or more occurrence data structures. Both classes can combine the use of primitive and scalar data types.

BINARIES

In the binary representation, information is represented using only two digits: 0 and 1. A scalar data structure comprises individual elements and is not composed of other data structures.

A typical computer system uses two types of memory: volatile memory (RAM) and nonvolatile memory (hard disks, flash memory, etc.). Volatile memory is short-term memory that loses its content when it is turned off.

In digital logic, binary forms have core functions in data processing. Digital circuits use binary number systems with only two states, 0 and 1, for mathematical computations for storing and transmitting information. The sign and magnitude, one's complement, and two's complement are methods for representing both positive and negative numbers.

Digital systems, such as computers, use binary numbers to perform arithmetic, store and retrieve data, and control and organize tasks. All the information must be encoded in a sequence of 0s and 1s, which circuits the process.

The decimal number is seven, and its equivalent 8-bit binary representation is 0111. When complex data and operations are maintained in hardware, they exist solely in basic on-off states in binary systems [5]. The binary coding of numbers uses positional notation to represent values with only two digits: '0' and '1.' Each consecutive position corresponds to the binary radix power of base 2. B0 is the rightmost digit and can have only two values: 0 or 1. B-1 is the next highest significance digit to the right of the point.

Hex Notation

Hexadecimal numerals, often known as base 16 or hex, are represented by digits 0-9 and A-F, representing values of 0-15 in decimal.

In the hexadecimal representation, we utilized base 16. In this scheme, 16 possible legal digital values are represented. Modern computer systems have adopted binary systems because of their design simplicity. For the convenience of interpretation of data and its transmission purposes, the binary or

decimal numbers are converted into equivalent hexadecimal numbers. Each hexadecimal digit had a decimal value between 0 and 15. A string of hexadecimal digits represents a sequence of 4-bit binary groups. For instance, the 7-bit ASCII representation of the letter 'C' is 1000011. 1000011 may be split into two 4-bit groups: 1000 to > 8 and 011 to > 3, meaning 43. Here, a new number system with base 16 was introduced. This system is called the hexadecimal number system. To define a new number of systems, we require 16 numbers. The 16 numbers are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and letters A, B, C, D, E, F. In this context, digits 0 through 9 are equivalent to those in the decimal number system, where A represents 10, B represents 11, C represents 12, D represents 13, E represents 14, and F represents 15. Each digit in the hexadecimal number system corresponds to 4 bits in the binary number system. Each digit represents one of four unique binary digits. The hexadecimal digit 0 corresponds to the 4-bit binary sequence 0000; 1 corresponds to 0001; 2 corresponds to 0010; 3 corresponds to 0011; 4 corresponds to 0100; 5 corresponds to 0101; 6 corresponds to 0110; 7 corresponds to 0111; 8 corresponds to 1000; 9 corresponds to 1001; A corresponds to 1010; B corresponds to 1011; C corresponds to 1100; D corresponds to 1101; E corresponds to 1110; and F corresponds to 1111.

A binary number can be segmented into 4-bit groups, each capable of representing values from 0000 to 1111, providing 16 possible combinations ranging from 0 to 15. Given a base value of 16, the highest value for a single digit was 15.

Converting a binary number to a hexadecimal number, and vice versa, is straightforward. Table 1 shows a representation of these numbers in each system. Furthermore, the digits to the left of the hexadecimal point have weights of $16^0, 16^1, 16^2$ and so etc. Likewise, the positions to the right have weights of $16^{-1}, 16^{-2}$, and so on [6].

COMPUTER SYSTEM DATA ABSTRACTION

Only the general, interesting, and relevant characteristics of an object were considered. Detailed characteristics with many minute details are ignored. For example, a car's steering wheel is round, made of soft materials, and colored.

Data abstraction involves turning an entity into simpler elements by ignoring the properties of its components. Thus, an automobile company belongs to some categories, such as automobile producers, Indian companies, and top-ranking Indian companies. Automobile companies operate at various levels of abstraction. At each level, one can get different properties pertaining to automobiles that can be obtained. Similarly, on a computer, the term data refers to the program-controlled movement of informational items to and from memory and other devices. This term is also used to refer to an informational item, the value of which is used in processing. A data item can be a name, number, or any other type of information. The data are processed using either logical instructions or I/O instructions and are often represented as alphanumeric characters, numeric, or binary values to be manipulated by the programmer. Data abstraction provides the concept of data hiding, which allows a programmer to ignore certain characteristics of data while working with certain data types, functions, and mechanisms. Object-oriented programming is a software paradigm based on the data abstraction concept of computer systems.

Statter and Armoni (2017) [7] stated that a prior study explored a framework for teaching abstraction in computer science (CS) within an introductory course designed for 7th-grade students. This framework was found to be effective for developing CS abstraction skills.

Table 1. Binary numbers.

Hexadecimal number	0	1	2	3	4	5	6	7
4-bit binary number	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal number	8	9	A	B	C	D	E	F
4-bit binary number	1000	1001	1010	1011	1100	1101	1110	1111

The current paper presents a study that examines the teaching of abstraction in CS to 7th-grade students from a gender perspective. Specifically, they compared the CS abstraction abilities of boys and girls enrolled in an introductory CS course using the Scratch environment. They also investigated how teaching strategy differentially impacted the CS abstraction skills of boys and girls. The results revealed a notable advantage for girls regarding various aspects of CS abstraction.

Furthermore, the framework established in their previous study as effective for 7th-grade students showed even greater effectiveness in this context for girls. This enhancement was evident in girls' superior overall performance compared to boys in the course utilizing this framework, as well as in their deeper and more consistent understanding of CS. The authors suggested that this framework may positively influence girls' self-efficacy, potentially encouraging and motivating them to continue pursuing CS education beyond the introductory course. In other words, this framework has the potential to address the issue of female underrepresentation at various stages of CS education, thus contributing to expanding the CS pipeline for women.

Abstract Data Types

An abstract data type is an abstraction of a data structure that specifies only the operations that are applied to the data structure without specifying how they are implemented. The separation of the concept of data from its implementation, as required by ADTs, provides significant programming advantages. The same data structure can be used for storing different types of information as long as the operations to be performed on the structure are similar. Most modern programming languages, including object-oriented languages, use this separation, and significant strides have been made to popularize this sound method of structuring data. In the following section, we consider some common built-in abstract data types.

Stacks: A stack is a one-dimensional list of objects such that objects are added or removed from only one end of the list, called the top. Because only the top object can be removed, objects stored at the top of the stack are removed before those stored below. This property is known as last-in-first-out. Stacks are extensively used as supporting storage in computers and are easy to implement using typical data structures, such as arrays or linked lists. Consequently, many of the algorithms we consider require a stack, and their textual descriptions may not directly suggest this.

Abstract data types are abstractions of data structures that simply provide the interface to which they must comply, not the implementation specifics or programming language [8].

Before understanding the abstract data type model, it is essential to grasp the concepts of abstraction and encapsulation.

- *Abstraction:* This method conceals internal workings from the user while only presenting essential information.
- *Encapsulation:* The technique of combining the data and member functions into a single unit is referred to as encapsulation.

Figure 1 illustrates the ADT model, which includes both public and private functions as well as the data structures utilized in our programs. Initially, encapsulation was implemented, encapsulating all the data within a single unit, referred to as ADT. Subsequently, abstraction occurs, highlighting the operations that can be performed on the data structure and specifying the data structures in use.

Data Structures

A data structure is a specialized format that organizes and manipulates data items. Additions and removals are the two main operations; however, other operations include printing, searching for, and changing the value of an item.

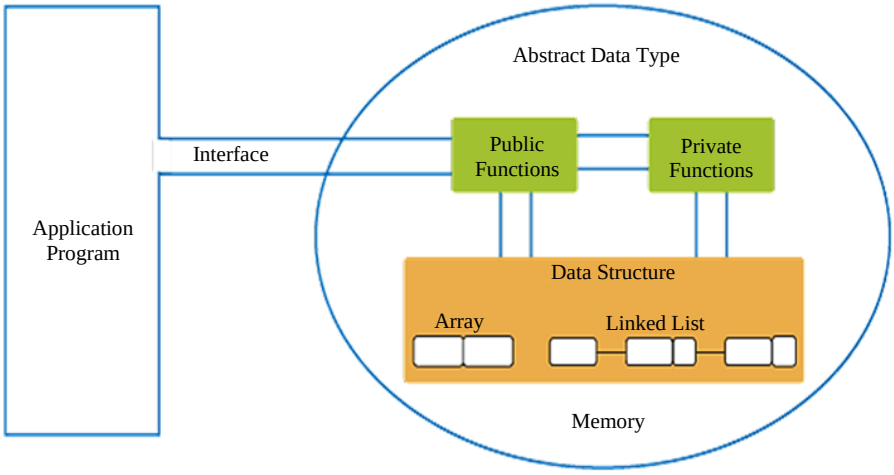

Figure 1. Abstract Data Type (ADT) model.

INTERACTIVE DATA PRESENTATION TOOLS

Traditionally, data representation has been taught through definitions and abstract illustrations. To make the subject lively, more interactive procedures, such as changing a number to a binary representation and performing exponential computations using constant multiplications, were introduced in the classroom. As the interest of the students has to be sustained throughout the work of the interactive features and at the same time to avoid frivolous use of valuable computer time, the interactivities must be designed carefully. We formulated the following principles to guide us in this task: avoiding all forms of direct teaching through interactivity. The interactive feature can be used only to illustrate definitions, convert numbers from one representation to another, convert a program in one language to another, verify a theorem in the case of simple problems, generate examples for guessing the solution, tutor-student learning to solve problems and interact with the student to test their understanding. To illustrate the definition of a number-based conversion, it is necessary to show the expansion of a number in the form shown in the equation. The user is driven through an exploratory phase between these two events. The initial screen, where the user is shown the number to be converted, the format to which it is to be converted, and certain parameters that are to be defined, is an essential interactive tool that avoids a cookbook method in which the student invokes the rules every time from the computer. In the exploratory phase, the student can request one of a series of tests on randomly selected numbers to gain confidence in the applicability of the method or to accumulate a set of examples without which practice is impossible.

Descartes (previously known as IRIS) is a software system designed to facilitate the visual exploration of spatially referenced data such as demographic, economic, or cultural information related to geographical entities such as countries, districts, or cities. It provides two integrated services: automated data presentation on maps, and interactive map manipulation tools.

The system incorporates general knowledge of map design to enable automated mapping, selecting appropriate presentation methods based on the characteristics of the variables being analyzed and their relationships when multiple variables are chosen. With Descartes' cartographic expertise, non-cartographers can obtain accurate presentations of their data, and the automation of map creation allows users to save valuable time, which can be better utilized for data analysis and problem-solving. Exploratory data analysis requires highly interactive, dynamic data displays. We strive to develop various interactive techniques for map manipulation that can enhance the expressiveness of maps and thus promote data exploration. We are convinced that a technique can be made especially productive if it is directed towards a particular presentation method: it can utilize the peculiarities of this method and support those analytical operations that best fit the method [9].

Graphics

Data representation is the basis of computing and handling information. Representing data with visual forms is the most straightforward method for facilitating the human processing of data. Data representation is generally preceded by a reduction in the total amount of data produced, which makes handling manageable. This is achieved through proper computing techniques to produce a package of meaningful information for human consumption, from which decision-making is facilitated. Hence, the effective representation of abstract data using a combination of known mathematical concepts can assist in achieving computational goals. Therefore, designing mathematical algorithms for efficient handling is essential. The focus is on using the available software tools to perform data transformation before attempting to understand the implementation aspects of the tools and algorithms. In essence, it is believed that designing smart tools is beyond the scope of computer applications. While graphical representations, such as bar charts, histograms, scatter diagrams, pie charts, and area diagrams, can help in providing a visual indication of the nature of data and can lead to certain inferences not easily obtainable through textual descriptions of the data, the selected graphical representation method should reflect the purpose behind the process.

Graphical representations convey to their readers a general view of the information they carry, thus removing readers' dependence on the original data for further processing. They are normally graphical displays of qualitative or descriptive data. In contrast, quantitative data, such as open-ended or sensitive numerical data, should be rendered as speech, tabular, or textual explanations. Graphical representations enable errorless data transmission owing to the minimum usage of declarative variability and instantaneous perception. Generally, design experts or graphic artists can plan on details, such as boundaries between the informative area and the decorative area, line thickness, line type, dash patterns, color, and fill patterns. They can pay attention to not forgetting the texture, patterns, or granularity of the graphical representations. The use of human understanding of graphical representations can make the interactions more functional. The most common bi-dimensional graphical representations can be classified into positioned or non-positioned representations. Statistical graphics functions allow the replacement of colors or patterns with other symbol features, such as sizes, varieties, or grayscale. These representations are useful for exploring bivariate relations. The use of other visual elements, such as connecting lines, in addition to the symbol-bivariate relation, helps in reinforcement. The occurrence of categorical disparity split across the levels of a third conditioning variable should be avoided. Misleading emphasis on grand averages also indicates inappropriate use of bar charts and pie diagrams.

Simulations

A simple, icon-based, relative file simulation system illustrates relative file organization. A disk simulation system using a tree and a list of file allocation table approaches illustrates disk initialization, format, disk chaptering, tree approach to disk management, and Abstract Data Type (ADT) approach. Finally, a project-oriented simulation system explains that computer simulation provides an in-depth study of a real-life situation and can be used to illustrate a concept or an idea. A well-presented computer simulation.

APPLICATIONS

This study discusses data representation and abstraction. We have software tools that can execute and visualize algorithms with complex computer data structures. It provides instructions for experiments and case studies that require students to perform imaginative input preparation, as well as the mechanical operations required to make algorithms work. We believe that studying and implementing known data representation and abstraction methods at an accessible level of granularity can be a valuable experience for computer science students learning important software concepts. We aimed to contribute to this learning process. In this study, we introduce a variety of explorations in which software students apply algorithmic knowledge by defining and testing data representations. Our objective was not to help computer science students solve domain-specific problems using known methods. Instead, our approach uses domain-specific data types as illustrative targets for generic

programming methods. Important concepts in introductory computer science courseware include formal descriptions of elementary data types, modules, procedural abstractions, data representations, and data abstractions. The Renderer model is a software system that is explicitly used to illustrate and demonstrate imperative and generic data representation methods for students in university-level computer science courses.

While there is broad acknowledgment of the need for data models with more complex semantics, no single approach has achieved widespread acceptance [10]. Existing data models often fail to provide direct support for relationships, data abstraction, inheritance, constraints, unstructured objects, or dynamic application properties.

FUTURE RESEARCH

Current data models do not adequately support relationships, data abstraction, inheritance, constraints, unstructured objects, or the dynamic properties of applications. Although there is widespread acknowledgment of the need for data models with richer semantics, no single approach has gained universal acceptance.

Possible future study and application areas in data representation and abstraction include developing technology, growing data complexity, and efficient system architecture.

Quantum Computing and New Data Representation

- *Research focus:* Quantum computing simultaneously introduces qubits that can represent data in multiple states. Future research could explore how classical data representation can interact with quantum systems and the abstraction layers required for efficient programming.
- *Application:* Programming languages and compilers that abstract quantum states and operations for quantum software.

Machine Learning and AI Data Abstraction

- *Research focus:* AI and machine learning are rapidly processing massive datasets using high-level abstractions. New ways to abstract and represent data that can be dynamically optimized for machine learning algorithms can improve speed, precision, and resource efficiency.
- *Application:* Developing abstraction methods to integrate and manipulate big datasets, making AI systems easier for non-experts to use, and boosting model performance.

Internet of Things (IoT)/Edge Computing

- With the rise of IoT devices and edge computing, data representation and abstraction must be optimized for devices with limited processing power and memory. Research can focus on lightweight abstractions and more efficient data formats for distributed systems.
- *Application:* Building frameworks to abstract device data and synchronize with centralized or cloud infrastructure for IoT systems.

Energy Efficiency Data Representation Optimization

- *Research focus:* Energy consumption in computing systems, especially data centers, is becoming more important. Energy-efficient data representation methods, such as approximations or low-precision formats, can reduce computation and storage requirements without affecting accuracy.
- Applying energy-saving abstractions to system and software development environments promotes green computing.

Adaptive Dynamic System Data Abstraction

- *Research focus:* Dynamic data abstractions could allow adaptive systems, such as autonomous vehicles and smart cities, to adjust their level of detail or abstraction based on processing needs or resources.

- *Application:* Auto-adjusting the data format and abstraction levels for real-time decision-making systems to improve performance and efficiency.

High-Level Blockchain and Decentralized System Abstraction

- *Research focus:* Blockchain and decentralized systems require secure and transparent data management. Abstraction techniques, such as encryption and distributed consensus mechanisms, can help developers work with blockchain technology without dealing with its complexities.
- *Application:* Creating higher-level languages or libraries that abstract blockchain algorithms and protocols for easier acceptance and development.

Human-Centered Data Abstraction

- *Focus:* Research into new abstraction layers that facilitate human-system interaction, such as visual programming environments or better user interfaces that hide complexity.
- *Application:* Making complex data structures and algorithms easier to understand using straightforward tools to boost data science, engineering, and healthcare productivity.

Data Representation for Security and Privacy

- *Research focus:* As cybersecurity becomes more important, data representation methods that protect data integrity, confidentiality, and privacy may be studied. This can include formats that support secure computations and abstractions that protect sensitive data during processing.
- *Application:* Creating abstraction layers for secure multiparty computation, homomorphic encryption, or differential privacy to handle data securely without disclosing critical information.

Augmented reality (AR)/Virtual reality (VR) Data Abstraction

- *Focus:* AR and VR systems process and interact with enormous volumes of sensory and visual input. Research can enhance data abstraction and representation to improve real-time processing and interaction.
- *Application:* Create abstractions that allow developers to build more complex and realistic virtual environments without overloading processing power, resulting in smoother and more immersive AR/VR experiences.

Advanced Abstract System Compiler Design

- Further integration of data representation and abstraction concepts could lead to compilers that dynamically optimize data representation based on the workload or architecture of the system.
- *Application:* Building compilers that optimize code and data structures for cloud, edge, and mobile computing environments.

Cross-disciplinary Data Abstraction Applications

- *Research focus:* Data abstraction can help bioinformatic researchers simplify biological data and gain new insights.
- *Application:* Data abstraction to better manage massive biological datasets such as DNA sequencing data or model complicated neurological systems.

Data representation and abstraction are vital to the future of computing and technology. Exploring these pathways may improve system efficiency, scalability, and usability.

CONCLUSION

Data Representation and Abstraction

This study emphasizes that data representation and abstraction are fundamental concepts in computer systems. Data representation converts real-world information into binary, hexadecimal, or various data types (integers, floating-point numbers, etc.) that computers can process efficiently. Abstraction

streamlines complex systems by concealing unnecessary details and understanding these principles is crucial for creating efficient and robust systems. Proper data representation optimizes the memory usage, processing speed, and storage. Abstraction reduces complexity, making systems easier to maintain, scale, and extend, without delving into low-level details.

Interactive Learning

This study's interactive tools and simulations show how data representations affect performance, functionality, and precision. Abstraction through layers of software (such as operating systems or programming languages) allows users to interact with systems without deep hardware or low-level operational knowledge.

Impact on Programming and System Development

This study shows that modern programming languages and software systems use data abstraction to manage complex computing tasks, such as objects in object-oriented programming (OOP) or data structures in functional programming.

Conclusion and Future Directions

Developers and system architects must understand data representation and abstraction to design more efficient, scalable, and user-friendly computer systems. As AI, machine learning, and quantum computing grow, more sophisticated abstractions and representations may be required.

This study demonstrated the applicability of both concepts in computer systems and suggested future research and applications.

REFERENCES

1. Hawkins P, Aiken A, Fisher K, Rinard M, Sagiv M. Data representation synthesis. Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, NY, USA: Association for Computing Machinery; 2011. p. 38–49. DOI: 10.1145/1993498.1993504.
2. Cima G, Console M, Lenzerini M, Poggi A. A review of data abstraction. Front Artif Intell. 2023;6:1085754. DOI: 10.3389/frai.2023.1085754. PubMed: 37426303.
3. Von Neumann J, Kurzweil R. The Computer and the Brain. New Haven, Connecticut, US: Yale University Press; 2012.
4. Mehlhorn K, Tsakalidis A. Data structures. In: Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity. Amsterdam, Netherlands: Elsevier; 1990. pp. 301–341. DOI: 10.1016/B978-0-444-88071-0.50011-4.
5. GeeksforGeeks. (2028). Binary representations in digital logic [Online]. GeeksforGeeks. Available from: https://www.geeksforgeeks.org/binary-representations-in-digital-logic/.
6. Awati R. (2022). Hexadecimal [Online]. WhatIs. TechTarget. Available from: https://www.techtarget.com/whatis/definition/hexadecimal.
7. Statter D, Armoni M. Learning abstraction in computer science: A gender perspective. Proceedings of the 12th Workshop on Primary and Secondary Computing Education. New York, NY, USA: Association for Computing Machinery; 2017. p. 5–14. DOI: 10.1145/3137065.3137081.
8. Javatpoint. (2024). Abstract data type in data structure. [online] Javatpoint. Available from: https://www.javatpoint.com/abstract-data-type-in-data-structure.
9. Andrienko GL, Andrienko NV. Interactive maps for visual data exploration. Int J Geogr Inf Sci. 1999;13:355–374. DOI: 10.1080/136588199241247.
10. Peckham J, Maryanski F. Semantic data models. ACM Comput Surv. 1988;20:153–189. DOI: 10.1145/62061.62062.

