Democratization of Corporate IT using IS Architecture Representation Framework (ISARF)

Kashif Saeed
University of North Texas
kashif.saeed@unt.edu

Anna Sidorova
University of North Texas
anna.sidorova@unt.edu

Abstract

Democratization, digital and data transformation, the need for responsible access to data and analytics resources, and the emergence of cloud computing are creating the need for understanding architectural aspects of IT for technical and business users. The combination of these trends, resulting in software-defined hardware, has blurred the line between software and hardware for IT developers. Similarly, the plug & play nature of serverless computing has allowed business users to manage their own cloud deployments. In spite of the numerous benefits, the combination of these new trends has created some challenges for IT management, one of which is to make the enterprise architecture more robust, accessible, and understandable. We, utilizing the representation theory and the theory of digital object, propose an IS architecture representation framework (ISARF) as a way to explain complex IT architectural concepts to software developers, business users, and executives. We posit that this framework can be utilized for the growing need for democratization of IT.

1. Introduction

Digital transformation redefines the role of digital objects in organizations by making them a critical element in how organizations organize and present themselves to customers in terms of business models, processes, and services [1]. In the digital transformation world, digital artifacts become defining characteristics of products and services, and the defining force in the design of business processes giving rise to so-called product-IT [2]. Product-IT is considered the revenue driver for organizations [2]. The management of product-IT is focused on innovation and business advantage, which requires fast turnaround and rapid path to transform the business. In contrast, enterprise-IT, in general, focuses on cost minimization, scalability, efficiency, and reliability rather than innovation and quick turnaround [2]. Product-IT developers and business users work closely with the customers, which requires them to not only understand the hardware and enterprise architecture concepts [2], but also understand the abstractions that the product provides to the customers. Therefore, the development and maintenance of product-IT relies on the democratization of IT for the deeper understanding of IT artifacts, enterprise architecture, components, integration, and abstractions in product-IT.

Democratization of IT, defined as citizen access of IT, is an emerging trend in information systems and practitioner literature. In simple terms, democratization of IT means making IT available to people, which can happen at a personal or corporate level. We, in this paper, focus on the democratization of IT at the corporate IT level. We contend that corporate IT has been democratizing IT to empower business users for several years, although it is a newer trend in the practitioner journals. For example, BI tools enable users to create their own reports and dashboards [3], which once used to be an IT responsibility. Similarly, analytics users are allowed to partially manage their own data discovery environments [4], which was once completely managed by IT. We argue that the recent trends in cloud computing, digital transformation, and data analytics are the reason behind the emergence of democratization in the practitioner journals. These emerging trends are pushing IT management to further the level of democratization that exists in IT.

The first trend is the IT investment in the adoption of cloud computing. According to the SIM IT trend study, the organizations surveyed, on average, delivered 44.8% of all IT services via cloud in 2019, up from 31.9% in 2016 [5]. Cloud computing allows organizations to provision hardware using software, often referred to as Infrastructure as Code (IaC) [6]. IaC uses high-level coding to automate the provisioning and customization of IT infrastructure. Traditionally, hardware engineers managed the hardware provisioning, whereas software developers were responsible for solution delivery from the software

URI: https://hdl.handle.net/10125/71326
978-0-9981331-4-0

perspective. The hardware-focused IT staff did not generally deal with software development or programming, whereas the software-focused IT staff did not deal with hardware, scaling, and enterprise architecture concepts. Thus, this divide between hardware and software in IT, which existed for decades, resulted in two silos – the software developers have little to no knowledge of hardware, and the hardware engineers have no in-depth knowledge of software development.

The emergence of cloud computing has exposed this divide between software and hardware. Cloud computing allows provisioning hardware using software by allowing developers to write code to automate the provisioning of hardware. The rise of DevOps [7] and the popularity of automation scripting tools like Ansible [8]–[10] and Terraform [11], [12] are examples of scripting tools used for automating and customizing the software provisioned hardware. This divide between hardware and software is creating a challenge for IT management, and there is an unprecedented need to cross-train resources in IT. There have been discussions in the computer science literature to incorporate these trainings at the corporate level [13], [14] and in academia [15], [16], however, the IS literature has not addressed this issue. We posit that the knowledge of enterprise architecture, hardware, and solution architecture, which once was limited to enterprise architects and hardware-specific IT employees, is now required for software developers and business users.

Another trend that is driving the calls for democratization of IT is the need for responsible access to data and analytics resources, and the cloud democratization of analytics and machine learning. The plug and play nature of serverless computing has enabled business users to build and deploy their own solutions in cloud [17], which requires an understanding of hardware and architectural concepts to effectively manage the resources and maintain responsible access to data.

The aforementioned trends converge to a common theme – making hardware and enterprise architectural concepts more visible and understandable to developers, product-IT, business users, and executives. Therefore, we argue that IT architecture cannot be hidden as separate from business architecture. This may be a new call in the context of democratization of IT, cloud computing, digital transformation, and user driven analytics, however, enterprise architecture have been used in the past by CIOs for making important decisions [18]. It can, however, be argued that CIOs may have the technical knowledge to understand the enterprise architecture concepts. Can the current enterprise architecture model explain the enterprise architectural concepts, especially the different levels of abstractions, to developers, product-IT, business users, and business executives?

In a case study conducted in 2018 on the effects of digital transformation on enterprise architecture [2], it is found that current enterprise architecture is not optimal for structuring product-IT. Additionally, the study [2] reported limitations in the existing enterprise architecture in representing business layers and customer focused interfaces. We agree with the findings in [2], in that, the existing enterprise architecture frameworks neither provides details of the different abstractions an IT artifact can provide for different users, nor breaks the information to the level of digital objects that make up the IT artifact. Lastly, the existing enterprise architecture frameworks are not grounded in IS theory, thus cannot explain the interplay between social and technical phenomena that shape the information systems in organizations. ISARF, on the other hand, builds on the representation theory and the theory of digital object, thus provides the socio-technical theoretical grounding needed to explain complex social and technical phenomena in information systems. Moreover, ISARF not only provides the details of different abstractions in an information system, it also has the ability to tie the abstractions with the social positioning of different IS users. Lastly, ISARF has the ability to view the architecture from a micro as well as macro level, which allows the users to skip the micro-level details if they desire.

Given the growing need for democratization of IT, the need for clearly articulating the business views and abstractions for digital transformation, the upcoming demand of business users deploying their own cloud solutions, the need to cross-train IT staff in architectural concepts, the broad-scale application of enterprise architecture in IS, and the lack of theoretical grounding of existing enterprise architecture frameworks, we propose IS architecture representation framework (ISARF) as a utility to make the IS architecture more visible and understandable. We contend that ISARF provides several advantages over the existing enterprise architecture frameworks, especially in explaining complex IT architectures to business users, executives, and developers who do not have background in hardware and IS architecture. While we are highlighting the merits of ISARF, we are not suggesting that ISARF is a replacement of the existing enterprise architecture frameworks, at least as of yet. Such claims require extensive empirical evidence and detailed evaluation of the merits of each framework. Rather, we suggest that ISARF provides several new benefits, and thus it can serve as an additional utility for IT management to succeed in today's digital and democratized IT landscape.

The paper proceeds as follows. The next section discusses the use of representation theory and the theory of digital object for the proposed IS architecture representation framework. Here we discuss the theory of digital object [19] and take the liberty to add some details that are not explicitly defined in the theory of digital object. Also, we use the structures defined in the representation theory to propose the architectural layers used in ISARF. Next, we provide the key propositions and tenets of IS architecture representation framework. Here we discuss the concept of architectural tiers and artifact delineation. Next, we discuss the application of ISARF to explain two information systems. We conclude by discussing the generalizability of ISARF to explain other information systems' architecture.

2. Background

Weber and Wand [20] describe their view of information systems in their discussion on the theory of deep structure, often referred to as representation theory in the IS literature, as: "We conceive of an information system as an object that can be studied in its own right, independently of the way it is deployed in its organizational and social context and the technology used to implement it" [8, Pg. 61]. Furthermore, "when modeling an information system, we are not concerned with the way it is managed in organizations, the characteristics of its users, the way it is implemented, the way it is used, the impact it has on such factors as quality of working life or the distribution of power in organizations, or the type of hardware or software used to make it operational. Instead, we are concerned only with information systems as independent artifacts that bear certain relationships to the real-world system they are intended to model. This view is not intended to denigrate the importance of deployment and technology issues to the successful development, implementation, and use of information systems. Rather, we seek to show that advantages accrue from decoupling the study of these issues from the study of certain other properties that can be identified when information systems are conceived as independent artifacts." [21]

Additionally, Weber and Wand distinguish between three "set of characteristics of the information systems object", as "surface-structure", "deep-structure", and "physical-structure" [20]. The surface-structure represents the interface between the information system and its users, for example the interactive dialogs used in a system or the reports generated by a system are referred to as surface-structure [20]. Deep-structure represent the meaning of the real-world system the information system models, or aims to model. For example, how an accounting system processes and posts the transactions to ledgers are deep-structure characteristics of an information system [20]. Lastly, the physical-structure of an information system represents how technology is used to implement the system. For example, the communication protocol or the data allocation on mass storage are examples of the physical structure [20].

Faulkner & Runde [19] theorize and define digital object in a way that is particularly instrumental for theorizing the IT architecture. According to the theory of digital object, a digital object is an object that has a bitstring as its component. Like all objects, a digital object must possess two characteristics: one, they endure – the state of persistence during their period of existence, and two, they are structured – the quality of maintaining their identity as an individual despite containing distinct parts [19].

Digital object can be classified as material or non-material. Material objects, like servers or computers, have a physical mode of being, while non-material objects, such as operating systems or software, do not have a physical mode of being [7, Pg. 6]. Material bearers can be accessed: ".. material bearers are vital to practical engagement with nonmaterial objects: to be accessed, a nonmaterial object must be borne on a material object" [7, Pg. 9].

The theory of digital object does not explicitly specify the following, but we would like to use the examples used in [19] to add that *material objects may also need non-material objects to be utilized.* As an example, you need the operating system (non-material object) to functionally utilize the resources on a server (material object). A server or a computer by itself, is nothing more than a box; one cannot even utilize the hard drive(s) on the computer or the server without the operating system (non-material object).

Additionally, we will also add that *one material object can be a bearer of multiple non-material objects. Conversely, the material object can be utilized by multiple non-material objects.* As an example, one server (material object) can have many operating systems (non-material objects), and each operating system (non-material object) can functionally utilize the resources on the server (material object). For example, a computer or a server can have Linux as well as Windows as operating systems, and each operating system will can functionally utilize the resources on the server or computer. See figure 1 for details.

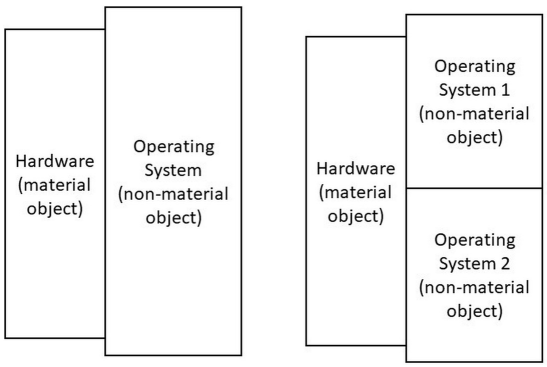

Figure 1. Non-material object(s) functional utilization of material object

We would also like to add that the software, by itself, is nothing more than a set of files or lines of code, which without the hardware cannot be executed. Similarly, the hardware, by itself, is like a metal (or plastic) box, which without the operating system cannot be accessed. The hardware and software, at least based on the technical advances till date, are used together to provide any kind of usable output.

Next, we present the IS architecture representation framework. We posit that this framework provides the theoretical grounding for IS architecture, and can serve as a utility to democratize IT architectural concepts.

3. Presenting IS architecture representation framework (ISARF)

ISARF takes the inspiration from the three structures defined in the representation theory. While the underlying views, assumptions, and premises[1] of the representation theory are different from our goal of understanding the architecture of complex information systems, we believe that the representation theory provides a conceptual basis for IT architecture. However, since the application of the three structures for ISARF is fundamentally different from the use by Weber, we rename the structures for the use of ISARF. This is an intentional deviation because we believe that the suggested new names are more relevant with the current state of digital enterprise architecture.

We are appropriating the notion of three structures (deep, physical, and surface) in the representation theory for dividing the components of a complex IS architecture into three tiers called *core tier*, *abstraction tier*, and *interaction tier,* in the order from innermost to the outermost. We define the core tier as the innermost structure of the IT system or component, which may include multiple layers of hardware and software (including OS). Additionally, we define the abstraction tier as the middle structure of the IT system or component comprising of software (or API) that represents the core tier. Lastly, we define the interaction tier as the outermost structure of the IT system or component that allows access to humans, software, or hardware. Table 1 maps the ISARF tiers with representation theory, digital objects, and IT artifacts. Figure 2(a) and 2(b) represent the ISARF tiers, which forms the foundational component of ISARF.

Table 1 – ISARF mapping with representation theory, IT Artifacts, and theory of digital object

IS architecture representation framework	**Representation Theory**	**IT Artifact (Digital Object) mapping**
Core tier	Deep Structure	Hardware (Material object) Software (Non-material object) Operating System (Non-material object)
Abstraction tier	Physical Structure	Software (Non-material object)
Interaction tier	Surface Structure	Human Software or API (Non-material object) Hardware (Material object)

[1]"The Fundamental Premise: A physical-symbol system has the necessary and sufficient properties to represent real-world meaning."
"Working Premise 1: An information system is an artifactual representation of a real-world system as perceived by someone, built to perform information processing functions."
"Working Premise 2: An information system is a state-tracking mechanism for the real-world system it is intended to model."
"Working Premise 3: A good information system is well decomposed." [20]

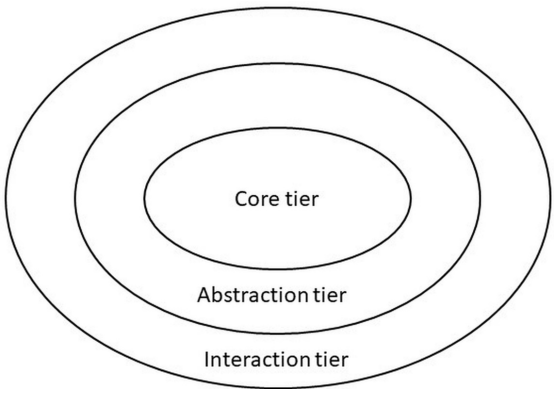

Figure 2a. Core, Abstraction, and Interaction tiers

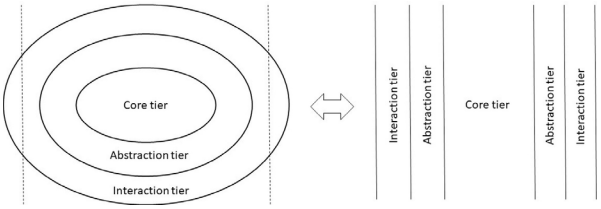

Figure 2b. Core, Abstraction, and Interaction tiers

3.1. Artifact Delineation

The creation and use of abstract object(s) from IT artifacts, which represent functional IT components (digital objects), is a foundational concept in understanding complex IT architectures. This phenomenon, however, is not represented in the traditional IT architecture design process. A traditional IT architecture diagram represents the integration of IT artifacts in an information system, without appropriating the different social positions and abstractions an IT artifact can take in an organization.

The concept of delineation is inspired by the computer science concept of abstraction, which is defined as simplifying something by hiding unnecessary details [22]. The Merriam-Webster definition of 'delineation' is to 'portray' or 'mark an outline of' something. We liked the definition because we are portraying the IT artifact by marking an outline for specific type of users in the system. So, delineation is abstracting an IT artifact and portraying it differently for different types of users in a system. An abstraction agent, generally a software, can create different abstractions of an IT artifact to serve different purposes. With this understanding, we define artifact delineation as *a process of creating abstract object(s) from one or more artifact(s) using an abstraction agent. The abstract object is created in the abstraction layer, where it can endure for as long as allowed by the abstraction agent. The abstraction layer can contain multiple abstract objects created by one or many abstraction agents.* Moreover*, the abstract object created by an abstraction agent can be utilized as a component for another process or system, with or without the inclusion of the original artifact or the abstraction agent as components of the process or system.*

If we utilize the terminologies used in the theory of digital object, artifact delineation can be rephrased as a process of creating abstract object(s) (material or non-material object) from one or more artifact(s) (material or non-material objects) using an abstraction agent (non-material object). The abstract object is created in the abstraction layer (non-material object), where it can endure as a digital object for as long as allowed by the abstraction agent. The abstraction layer can contain multiple abstract objects created by one or many abstraction agents. Moreover, the abstract object created by an abstraction agent can be utilized as a component for another process or system, with or without the inclusion of the original artifact or the abstraction agent as components of the process or system.

Let's discuss a simple IS example to build on our understanding. Let's assume that the IT department provided us a brand new laptop. The laptop, which is an IT artifact, has resources like RAM, CPU, and hard disk, which cannot be functionally utilized unless an operating system is installed on the laptop. Let's assume that the IT department installed Windows 10 as the operating system on the laptop. The operating system is the abstraction agent which provides the abstraction of the file system (the abstract object). Let's assume that you installed LINUX, another abstraction agent, on the same laptop. LINUX will provide the abstraction of another file system (abstract object) for the IT artifact and the resources therein.

3.1.1. User view of abstraction

Users can access the inner tiers of the ISARF through the interaction tier using a software, hardware, or a combination of software and hardware. Users' view of abstraction, or more specifically, the components in the core tier, including the original object, is controlled by the abstraction agent and the access level defined within. Different types of information systems users may have different access levels. For example, a business user may have read-only access to the components in the core tier through an abstraction in the abstraction tier. Alternatively, IT developers may have access to the core tier through the abstraction tier that allows them to access and modify the components in the core tier. There can be different abstraction layers for different kinds of users in the system, allowing users to view and interact with different components in the system. For example, a BI system may provide a management console, an abstraction created by the BI tool (abstraction agent), to be used by BI administrators. Similarly, the same BI tool may provide a different portal as an abstraction for the users to access their reports. Additionally, the abstraction agent can abstract an original object differently for different types of users. For example, a BI tool, which is an abstraction agent, can abstract a database table as a read-only table for a BI developer, but abstracts the same table as a set of objects for BI users who are responsible for creating reports using the BI tool.

3.1.2. Multi-layered IT architecture

IT architecture is often complex and can include multiple layers of hardware and software working together. Though the database architecture shown in Figure 4 is rather simplistic, you can still see the multi-layered nature of IT architecture from this example. For example, the core tier in Figure 4 contains the storage layer, the database server layer, the OS, and the DBMS software layer. Similarly, the abstraction tier contains multiple layers of abstractions created by different abstraction agents. Almost every IT system will have layers of software, hardware, and APIs working together. Complex IT systems may contain components, which are independent IT systems by themselves. For example, an ERP or a CRM system has DBMS, which is a separate information system by itself, as a component in the architecture. In spite of the complexity of the IT system, the atomicity of digital object and the utility of the three tiers inspired by reference theory allow ISARF to break down complex IT systems into components arranged in the core, abstraction, and interaction tiers. Thus, ISARF makes it easier for software developers, business users, and business executives to understand the architecture of complex information systems.

3.1.3. IT solutions involving multiple systems

Complex IT solutions involve multiple IT systems working together. These systems work together as the components of the larger IT system. As an example, AI systems comprise of components that can handle ingesting data from structured and unstructured data sources, processing data (data cleansing, merging, analyzing), and storing data. Solutions involving AI integrate these components to get the results they need. The individual components making up the complex IT system may be individual information systems, which communicate with one another using APIs through the interaction tier, as depicted in figure 3. These APIs, depending upon the user and API access level, may have access to the components in the abstract and core tiers of the system they are integrating with.

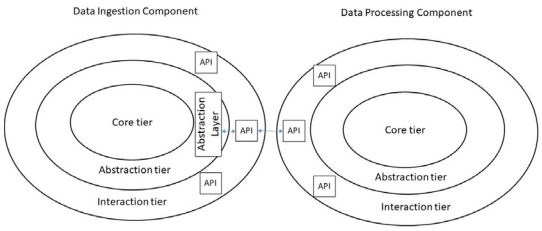

Figure 3. IT Solution involving multiple systems

Another example of different information systems working together as component of a larger information system is a data warehouse implementation. A data warehouse implementation uses a database management system, an integration system (ETL), and a BI system. The users of such complex information systems may not have the visibility or the understanding of the different components of the information system. The users of such complex information systems generally, but not always, appropriate the system based on the component they interact with. For example, if the data warehouse is not loaded properly, the users may see this as a BI system problem, even though the BI system is only reflecting the data that is available in the data warehouse. Similarly, an example of one information system interacting with the components of another information system is the access and ability of ETL developers to read and write to the database system through the ETL interface, which is possible because the credentials used to connect to the database system have read and write privileges to the database system.

In spite of the complexity and the number of information systems working as components of a larger information system, we posit that ISARF provides the atomicity to explain the component as well as the system as a whole.

4. Application of IS architecture representation framework (ISARF)

In this section, we apply IS architecture representation framework to a few examples to test its generalizability. We discuss DBMS architecture and computer virtualization using ISARF.

4.1. Use Case 1: Using ISARF to explain DBMS Architecture

We use database management system (DMBS) as the first example to explain information systems architecture using ISARF. We highlight that the DBMS architecture discussed in this example is vendor agnostic.

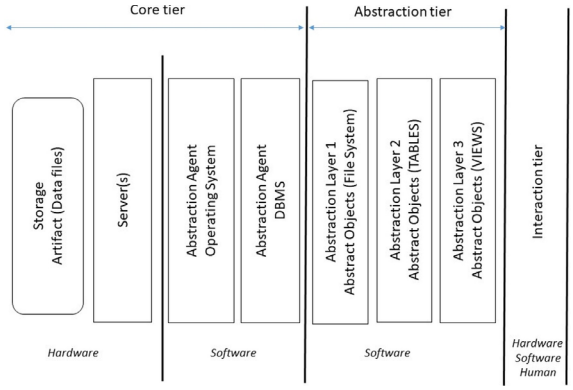

Figure 4. DBMS Architecture explained using ISARF

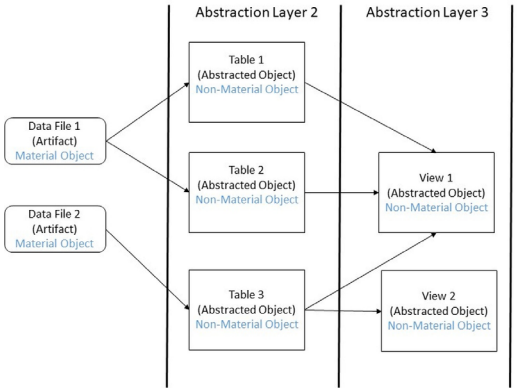

Figure 5. Association between Original Objects and Abstracted Objects

Figure 4 represents the multi-layered nature of database architecture. The core tier consists of the storage, the server(s), the operating system, and the database management system (DMBS). From an IT artifact perspective, the core tier contains both hardware and software, whereas from the digital object perspective, it contains both material and non-material objects. ISARF provides the flexibility of multiple abstraction agents, creating different abstractions in the abstraction tier. In our example, operating system serves as an abstraction agent to build the filesystem abstraction layer, whereas DBMS serves as another abstraction agent to build abstraction layer 2 and 3. Abstraction layer 3, which contains database views, uses the tables in abstraction layer 2 as the objects it abstracts from. This shows the multiple tiers of abstraction, that is, the database tables are abstract objects representing the data in storage (data files), whereas views are built on top of tables. It is also worth noting that an abstraction layer can have multiple objects – for example, there can be many tables and multiple views in the abstraction layer 2 and 3 respectively.

Additionally, many abstract objects can be created from one artifact, and one abstract object can be abstracted from many artifacts or abstract objects. In the example shown in figure 5, DBMS creates two abstract objects 'Table 1' and 'Table 2' from 'Data File 1', both residing in the same abstraction layer. DBMS also creates another abstraction layer, abstraction layer 3, which contains views. 'View 1', another abstract object in the abstraction layer 3, abstracts from 'Table 1', 'Table 2', and 'Table 3', all of which are abstract objects themselves. This example represents multiple levels of abstraction in information systems.

Table 2 below maps the components of the DMBS architecture to ISARF tiers, delineation components, digital objects, and IT artifacts. Moreover, table 3 maps the abstract objects in the delineation process to the abstraction agent they were created by, the IT artifacts they represent, and the kind of the digital object the IT artifact represents.

Table 2. DBMS architecture explained using ISARF

Components	**(IT) Artifact**	**Digital Object**	**ISARF tier**	**Delineation component**
Storage	Hardware	Material bearer of Non-Material Objects	Core tier	Not part of this delineation
Data Files in Storage	Files	Non-material Object	Core tier	Artifact (to be abstracted)
Server(s)	Hardware	Material Object	Core tier	[1] Not part of this delineation
Operating System	Software	Non-material bearer of Non-material Objects	Core tier	Abstraction Agent
DBMS	Software	Non-material bearer of Non-material Objects	Core tier	Abstraction Agent
File System	Software	Non-material bearer of Non-	Abstraction tier	Abstract Object

		material Objects		
Database Tables	Object	Non-material Object	Abstraction tier	Abstract Object
Database Views	Object	Non-material Object	Abstraction tier	Abstract Object

Note: [1] If the database architecture shown in Figure 4 uses virtual server(s), the virtual server(s) will represent abstract objects created by an abstraction agent (Hypervisor), which is not part of the database system.

Table 3. DBMS architecture components explained using ISARF

Abstract Object	**Abstraction Agent**	**(IT) Artifact**	**Artifact Object Type (digital object)**
Tables	DBMS	Data Files in Storage	Material Object
Views	DBMS	Tables	Non-material Object
File System	Operating System	Server(s)	Material Object

4.2. Use Case 2: Using ISARF to explain computer virtualization architecture

We use computer virtualization as our second use case for testing ISARF. Computer virtualization, generally referred to as 'virtualization', is defined as provisioning hardware (virtual machines or VMs) through software [9]. Figure 6 represents the architecture of computer virtualization explained using ISARF. Additionally, table 4 maps the components of the virtualization architecture to ISARF tiers, delineation components, digital objects, and IT artifacts.

Hypervisor, a software that represents the abstraction agent, creates multiple virtual machines representing the abstract objects in the abstraction layer marked as 'abstraction layer 1'. The physical server represents the artifact, the original object which is used by the abstraction agent for creating the abstract objects. These virtual machines exist in the abstraction tier, where they can endure as digital objects. More importantly, *the abstract object(s) (virtual machines) created by an abstraction agent (Hypervisor) can be utilized as a component for another process or system, with or without the inclusion of the artifact (original object, i.e. Physical server) or the abstraction agent (Hypervisor) as components of the process or system.*

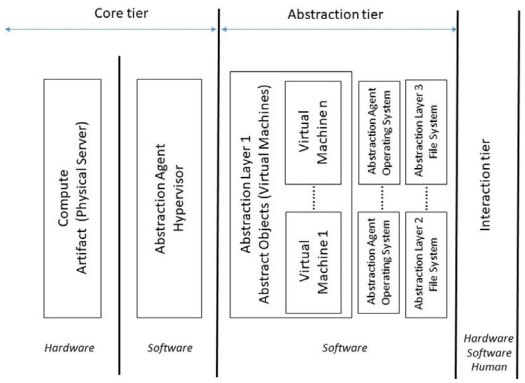

Figure 6. Virtualization explained using ISARF

Table 4. Virtualization explained using ISARF

Components	**(IT) Artifact**	**Digital Object**	**ISARF tier**	**Delineation component**
Physical Server	Hardware	Material Object	Core tier	Artifact (to be abstracted)
Hypervisor	Software	Non-material bearer of Non-material Objects	Core tier	Abstraction Agent
Virtual Machine(s)	Software provisioned Hardware	Non-Material bearer of Material and Non-Material Objects	Abstraction tier	Abstract Objects
Operating System	Software	Non-material bearer of Non-material Objects	Abstraction tier	Abstraction Agent
File System	Software	Non-material bearer of Non-material Objects	Abstraction tier	Abstract Object

5. Implications, Generalizability, and Conclusion

We, in this paper, have presented a generalizable theoretical framework that can be used as a utility to explain IS architectural concepts to software developers, business users, and business executives. Digital transformation, cloud computing, and business users' need for analytics are pushing the democratization of IT – a trend that requires software developers to provision hardware using IaC [6], product-IT and business users to understand the customer interfaces and

abstractions[2], and business users have to maintain their own cloud services for analytics [17]. While the existing enterprise architecture frameworks have limitations in effectively presenting the business view [2], ISARF provides an indispensable utility to organizations in making their digital transformation and democratization goals possible.

Moreover, enterprise architecture has played a key role in information systems success. It has been used in a variety of use cases, including but not limited to IT executives and CIOs decision making [18], as a framework for system quality analysis [23], organizational benefits [24], IT alignment [25], business value assessment [26], managing IT [27], and risk management [28]–[30]. As compared to the traditional enterprise architecture, we posit that ISARF provides several distinct advantages. One, ISARF provides a much more atomic and granular view to enterprise architecture, which provides deeper understanding at the digital object level. Two, ISARF is built using representation theory and the theory of digital object, which provides the theoretical grounding for the framework. Three, ISARF uses the socio-technical paradigm of information systems, which allows it to represent different social positions a user can have. Lastly, ISARF incorporates the concept of delineation, which highlights the different kinds of socio-technical abstractions in information systems architecture. Given these distinct advantages of ISARF, we posit that ISARF can be utilized in many IS applications including the ones mentioned afore.

As for future research, we can use ISARF in an experimental setup to evaluate its effectiveness in explaining different parts of the IS architecture to different audiences. Moreover, a case study or experiment can be conducted for the use of ISARF in different organizations to evaluate its effectiveness in explaining complex IS architectural concepts to business users, executives, and software developers who have no prior background in hardware or IS architectural concepts. The goal of such case study or an experiment would be to evaluate the possible added value for the business that this framework can provide. Additionally, a pretest-posttest experiment can be conducted with software developers to help them understand architectural concepts using the framework.

Lastly, IS researchers can explore several use cases of ISARF in explaining technical components, system architecture, integration of systems, and different levels of abstractions for different users in an organizational setup. IS research is often criticized for oversimplifying or ignoring the system aspects of IT [31]. We contend that ISARF can help IS research by highlighting the system implementation details for systems like big data, BI, ERP, CRM, etc. Additionally, complex IS infrastructures involving serverless computing and cloud computing can also be explained effectively using ISARF. We argue that ISARF will not only allow IS researchers to effectively present and explain different information system abstractions, but also tie these abstractions to different users in the organization, which enables interesting discussions on socio-technical usage and the design of user-specific interfaces in information systems.

6. References

[1] M. E. Porter and J. E. Heppelmann, "HBR.ORG SPOTLIGHT ON MANAGING THE INTERNET OF THINGS How Smart, Connected Products Are Transforming Competition," 2014. Accessed: Sep. 22, 2020.

[2] K. Julia, S. Kurt, and S. Ulf, "How Digital Transformation affects Enterprise Architecture Management-a case study," *Int. J. Inf. Syst. Proj. Manag.*, vol. 6, no. 3, pp. 5–18, 2018, doi: 10.12821/ijispm060301.

[3] S. K.-I. Management and undefined 2008, "Business intelligence the self-service way," *search.proquest.com*, Accessed: Jul. 02, 2020.

[4] A. Ali-Eldin, M. Salem, and M. M. Zaghloul, "Towards a Self-service Data Analytics Framework Towards a Self-service Data Analytics Framework General Terms," *Artic. Int. J. Comput. Appl.*, vol. 80, no. 9, pp. 975–8887, 2013, doi: 10.5120/13893-1840.

[5] L. Kappelman, V. Johnson, … C. M.-M. Q., and undefined 2020, "The 2019 SIM IT Issues and Trends Study.," *search.ebscohost.com*, Accessed: Jun. 29, 2020.

[6] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero, and D. A. Tamburri, "DevOps: Introducing infrastructure-as-code," in *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017*, Jun. 2017, pp. 497–498, doi: 10.1109/ICSE-C.2017.162.

[7] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Softw.*, vol. 33, no. 3, pp. 94–100, May 2016, doi: 10.1109/MS.2016.68.

[8] N. Singh, S. Thakur, … H. C.-2015 1st I., and undefined 2015, "Automated provisioning of application in IAAS cloud using Ansible configuration management," *ieeexplore.ieee.org*, Accessed: Jul. 08, 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7375087/.

[9] P. Masek, M. Stusek, J. Krejci, … K. Z.-… 22nd conference of, and undefined 2018, "Unleashing full potential of ansible framework: University labs administration," *ieeexplore.ieee.org*, Accessed: Jul. 08, 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8468270/.

[10] L. Hochstein and R. Moser, *Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way*. 2017.

[11] Y. Brikman, *Terraform: Up & Running: Writing Infrastructure as Code*. 2019.

[12] D. Ivanova, P. Borovska, and S. Zahov, "Development of PaaS using AWS and Terraform for medical imaging analytics ARTICLES YOU MAY BE INTERESTED IN Scalable framework for adaptive in-silico knowledge discovery and decision-making out of genomic big data AIP Conference," *aip.scitation.org*, vol. 2048, p. 60019, Dec. 2018, doi: 10.1063/1.5082133.

[13] M. Mazzara, A. Naumchev, L. Safina, A. Sillitti, and K. Urysov, "Teaching DevOps in Corporate Environments An experience report." Accessed: Jul. 02, 2020. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-06019-0_8.

[14] M. Mazzara, A. Naumchev, L. Safina, A. Sillitti, and K. Urysov, "Teaching devops in corporate environments: An experience report," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11350 LNCS, pp. 100–111, doi: 10.1007/978-3-030-06019-0_8.

[15] … C. J.-W. on S. E. A. of and undefined 2018, "A Proposal for Integrating DevOps into Software Engineering Curricula," *Springer*, Accessed: Jul. 02, 2020. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-06019-0_3.

[16] C. Jones, "A proposal for integrating devops into software engineering curricula," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11350 LNCS, pp. 33–47, doi: 10.1007/978-3-030-06019-0_3.

[17] Q. Pu, U. Berkeley, and S. Venkataraman, *This paper is included in the Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19). Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure*. 2019.

[18] P. Johnson, M. Ekstedt, E. Silva, and L. Plazaola, "Using enterprise architecture for cio decision-making: On the importance of theory." Accessed: Jun. 26, 2020. [Online]. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2:509306.

[19] P. Faulkner, J. R.-M. Quarterly, and undefined 2019, "Theorizing the Digital Object.," *search.ebscohost.com*, Accessed: Jun. 25, 2020.

[20] Y. Wand and R. Weber, "TOWARD A THEORY OF THE DEEP STRUCTURE OF INFORMATION SYSTEMS." Accessed: Jun. 25, 2020. [Online]. Available: https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1043&context=icis1990.

[21] R. W.-J. of I. Systems and undefined 1987, "Toward a theory of artifacts: A paradigmatic base for information systems research."

[22] T. C. Ae and G. Shute, "Abstraction in Computer Science," *Springer*, 2007, doi: 10.1007/s11023-007-9061-7.

[23] P. Närman, P. Johnson, and L. Nordström, "Enterprise Architecture: A Framework Supporting System Quality Analysis," *ieeexplore.ieee.org*, 2007, doi: 10.1109/EDOC.2007.39.

[24] G. Shanks, M. Gloet, I. Asadi Someh, K. Frampton, and T. Tamm, "Achieving benefits with enterprise architecture," *J. Strateg. Inf. Syst.*, vol. 27, no. 2, pp. 139–156, Jun. 2018, doi: 10.1016/j.jsis.2018.03.001.

[25] R. V. Bradley, R. M. E. Pratt, T. A. Byrd, C. N. Outlay, and D. E. Wynn, "Enterprise architecture, IT effectiveness and the mediating role of IT alignment in US hospitals," *Inf. Syst. J.*, vol. 22, no. 2, pp. 97–127, Mar. 2012, doi: 10.1111/j.1365-2575.2011.00379.x.

[26] M. Meyer, M. Helfert, B. Donnellan, and J. Kenneally, "Applying design science research for enterprise architecture business value assessments," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012, vol. 7286 LNCS, pp. 108–121, doi: 10.1007/978-3-642-29863-9_9.

[27] W. F. Boh and D. Yellin, "Using enterprise architecture standards in managing information technology," *J. Manag. Inf. Syst.*, vol. 23, no. 3, pp. 163–207, Dec. 2006, doi: 10.2753/MIS0742-1222230307.

[28] H. Jonkers and D. A. C. Quartel, "Enterprise architecture-based risk and security modelling and analysis," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016, vol. 9987 LNCS, pp. 94–101, doi: 10.1007/978-3-319-46263-9_6.

[29] F. Innerhofer-Oberperfler and R. Breu, "USING AN ENTERPRISE ARCHITECTURE FOR IT RISK MANAGEMENT." Accessed: Jul. 07, 2020. [Online]. Available: https://pdfs.semanticscholar.org/a41a/e6f15069246eca92fe02dd17505fcd881327.pdf.

[30] J. R. Getter, "Enterprise Architecture and IT Governance A Risk-Based Approach," 2007. Accessed: Jul. 07, 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4076825/.

[31] W. J. Orlikowski and C. Suzanne Iacono, "Research Commentary: Desperately Seeking the 'IT' in IT Research-A Call to Theorizing the IT Artifact," *Inf. Syst. Res.*, vol. 12, no. 2, p. 121, 2001, doi: 10.1287/isre.12.2.121.9700.