electronics

MDPI

Article

Ontology Development for Creating Identical Software Environments to Improve Learning Outcomes in Higher Education Institutions

Predrag Stolic [1,*], Danijela Milosevic [2], Zoran Stevic [1,3] and Ilija Radovanovic [3,4]

[1] Technical Faculty in Bor, University of Belgrade, Vojske Jugoslavije 12, 19210 Bor, Serbia
[2] Faculty of Technical Sciences, University of Kragujevac, Svetog Save 65, 32102 Cacak, Serbia
[3] School of Electrical Engineering, University of Belgrade, Bulevar Kralja Aleksandra 73, 11120 Belgrade, Serbia
[4] Innovation Center of the School of Electrical Engineering in Belgrade Ltd., Bulevar Kralja Aleksandra 73, 11120 Belgrade, Serbia
* Correspondence: pstolic@tfbor.bg.ac.rs

Abstract: Students engage in remote learning within a diverse computer environment. While virtual machines can address the challenges posed by heterogeneity, there remain unresolved issues, particularly related to the complexity of software management. An imperative is to discover an automated solution that facilitates the creation of consistent software environments for educational purposes. This paper introduces ontology engineering principles as a means to tackle the complexities associated with software management. A suitable ontology is developed using OWL syntax, integrating knowledge pertaining to the required software within a specific academic domain. The practical applicability of this knowledge is enabled through the implementation of dedicated SPARQL queries within a Python program. The effectiveness of the automated solution in achieving identical software environments is verified through testing, conducted in both controlled laboratory settings and by students themselves, thus simulating authentic teaching scenarios. The solution not only adheres to the principles of reusability but can also be adapted or integrated into existing ontologies. Furthermore, it presents an opportunity to create automated and self-adjusting virtual machines, offering significant potential for educational and other domains.

Keywords: heterogeneous computer environment; higher education; Linux; ontology; OWL; Python; software management; SPARQL

Citation: Stolic, P.; Milosevic, D.; Stevic, Z.; Radovanovic, I. Ontology Development for Creating Identical Software Environments to Improve Learning Outcomes in Higher Education Institutions. *Electronics* **2023**, *12*, 3057. https://doi.org/10.3390/electronics12143057

Received: 20 June 2023
Revised: 4 July 2023
Accepted: 7 July 2023
Published: 12 July 2023

1. Introduction

Modern computing encompasses various tasks that require the utilization of different resources, depending on the complexity of the desired solution. In [1], the authors highlight the presence of a heterogeneous computing environment, which is a characteristic of contemporary computing due to the diverse range of resources employed at different levels. This heterogeneity arises from variations in individual computer components, as well as the characteristics of computers and computer systems. When considering larger sets of computing resources organized in accordance with modern concepts such as fog computing, edge computing, and cloud computing, special attention must be given to each local node. The exceptional heterogeneity observed at the lowest level of each node carries forward and manifests itself at higher levels [2]. Consequently, heterogeneity has become one of the challenges to overcome in the design and implementation of reliability, scalability, security, and privacy [3]. In addressing these challenges, one should focus on developing approaches that facilitate the efective connection of heterogeneous devices in various environments [4].

To cope with the distinct heterogeneity of the computer environment, several key concepts need to be applied. First and foremost, proper management is essential to handle

the unpredictability, variability, and dynamism exhibited by resources [5]. Additionally, adequate connectivity between heterogeneous devices must be ensured [6]. Lastly, software solutions should be adapted to effectively function in heterogeneous conditions [7]. While there have been programming and modeling adaptations already [8], addressing heterogeneity in the domain of higher education introduces additional challenges.

The digital transformation of higher education, particularly accelerated by the COVID-19 pandemic, has brought forth new obstacles for students using personal computer resources for learning [9]. Another significant issue identified in [10] is the obsolescence of equipment and software, which amplifies existing risks and introduces new ones. Thus, it is crucial to minimize risks to an acceptable level. Virtualization offers a viable solution by allowing easy and rapid installation, reinstallation, adjustment, and repair of virtual machines, along with software adaptation, without imposing complex operations or additional efforts on students [11]. It is worth noting that overcoming the heterogeneity of the computer environment does not have to be an expensive process. For instance, solutions based on free and open source software [12] can reduce expenses, and many of these solutions are cross platform.

Addressing the challenges related to software environments in heterogeneous conditions raises several crucial questions, as highlighted by the authors in [13]. These questions revolve around the intricate sharing and reuse of acquired knowledge, given that each software possesses specific characteristics that complicate mutual communication. Redundancy also becomes a concern due to the presence of extensive shared knowledge among different software, necessitating the acquisition of knowledge from scratch for each new software considered. Utilizing ontologies can effectively address these challenges by providing a suitable model composed of classes, attributes, and relationships [14]. Through proper specification, conceptualization, and implementation [15], ontologies establish common semantics [16] within the heterogeneous computer environment, facilitating seamless knowledge exchange and straightforward reuse [17]. Principles based on ontologies have successfully overcome a wide range of problems in software engineering and information systems implementation, leading to the development of new approaches [18]. Moreover, existing methodologies have been supplemented with ontologies, resulting in integrated approaches [19]. Ontologies have proven to be valuable tools in information management [20] and configuration management [21]. Their significance becomes evident when dealing with challenges stemming from environmental variability, offering essential techniques for identification [22] and creating suitable responses to environmental changes [23]. Furthermore, ontologies enable the synthesis of acquired knowledge within organized repositories [24], promoting the principle of reuse.

Ontological engineering principles cover many aspects of software in traditional computing, and a certain number of published papers can be found in the literature [25]. However, there is still significant potential for further advancement, especially when considering systems designed for higher education, which exhibit a high degree of heterogeneity in the computer environment. As emphasized in [26], establishing a suitable computing environment and information system poses challenges due to rapid technological development and exceptional manifestations of heterogeneity. However, the principles of ontological modeling provide a strong foundation for overcoming these issues.

When implementing "learning from home", researchers have primarily focused on various learning styles, diverse assessment methods, engagement, motivation, and other characteristics. Technological studies have mainly aimed to enhance e-learning systems by expanding their capabilities, improving interoperability, and introducing new collaboration features. However, during practical teaching tasks, students also utilize software solutions outside the e-learning system within a highly heterogeneous computer environment. This complexity adds two challenges to the teaching process: achieving uniformity throughout the teaching process and enabling effective software management. These challenges place an additional burden on students and teaching staff (e.g., lecturers and technical personnel), diverting their attention from the core teaching tasks. Although there are numerous

software management solutions available, it is crucial to develop a solution that considers specific aspects of the academic domain. Therefore, the following chapters present a solution that facilitates the simple and automated installation of the necessary software for teaching, which minimizes the involvement of students and teaching staff. This solution is based on constructing a suitable ontology using OWL syntax and implementing appropriate SPARQL queries to obtain relevant knowledge, enabling the realization of consistent software aspects on different student machines. Such a solution can be easily incorporated into most programming languages, and in this paper, the Python programming language was utilized. The proposed solution not only automates certain software aspects of the teaching process, but also offers broader applicability. Based on the developed ontology, software vendors can easily incorporate their installation procedures intended for academic use, academic institutions can exchange knowledge about the software they use, and the complexity of the solutions employed remains low, requiring minimal additional technical knowledge for implementation. While the presented solution is specifically tailored for the academic world, its potential for application extends to other domains as well.

2. Literature Background

The significance of having adequate technical infrastructure for the teaching process is discussed in [27], where the authors highlight that the success or failure of distance education is heavily influenced by the infrastructure provided. When implementing a learning system primarily based on remote learning from home, the infrastructural challenge becomes even more complex due to the utilization of students' personal computer resources, which differ significantly in terms of hardware. In educational institutions, computer laboratories equipped for practical teaching in courses requiring computer usage are well-defined homogeneous computer environments. Detailed knowledge of the hardware resources in these laboratories exists, as they were set up based on the appropriate technical requirements and specifications. Each computer laboratory represents a known homogeneous computer environment, enabling a precise understanding and predictability of the behavior of individual computers within them. However, in the context of learning from home using personal computer resources, the environment becomes highly heterogeneous, with numerous unknowns concerning the computer hardware used and its behavior during different teaching activities. To ensure that this heterogeneity does not introduce additional problems in the teaching process, it is essential to find an appropriate solution to overcome the complexity it presents.

One effective solution to address these challenges is to introduce virtualization concepts during the teaching process. Virtualization involves creating an abstraction layer that enables the creation of virtual hardware over physical hardware [28]. Authors in [29] provide valuable insights by stating that "implementing virtualization technology is a way to relax the physical hardware constraints and increase a system's flexibility". Virtualization, with the appropriate abstraction, has the potential to transform the noted heterogeneity into a more homogeneous environment. Various virtualization techniques are employed today, and Figure 1 presents a graphical comparison of some of their fundamental characteristics [30].

In the domain of higher education, the prevalent solutions for virtualization implementation utilize type 2 hypervisors. Type 2 hypervisors, also known as Virtual Machine Managers (VMMs) are software installed on the existing computer operating systems (host systems). They enable the creation and management of virtual machines (VMs) using virtual hardware. Each VM operates in an isolated environment with its own operating system (guest system). Standard operating systems commonly used on physical computers (such as Windows, Linux, macOS, and others) can be utilized without the need for specialized or dedicated operating systems. This flexibility allows us to seamlessly use almost all applications in a virtual environment, just as we would in a physical environment. Consequently, VMs are ideal for educational purposes and student use, as they eliminate the need for additional knowledge or training in application usage. Figure 2 illustrates a

graphical representation of the fundamental layers in the virtualization architecture, which is extensively employed in the field of higher education.

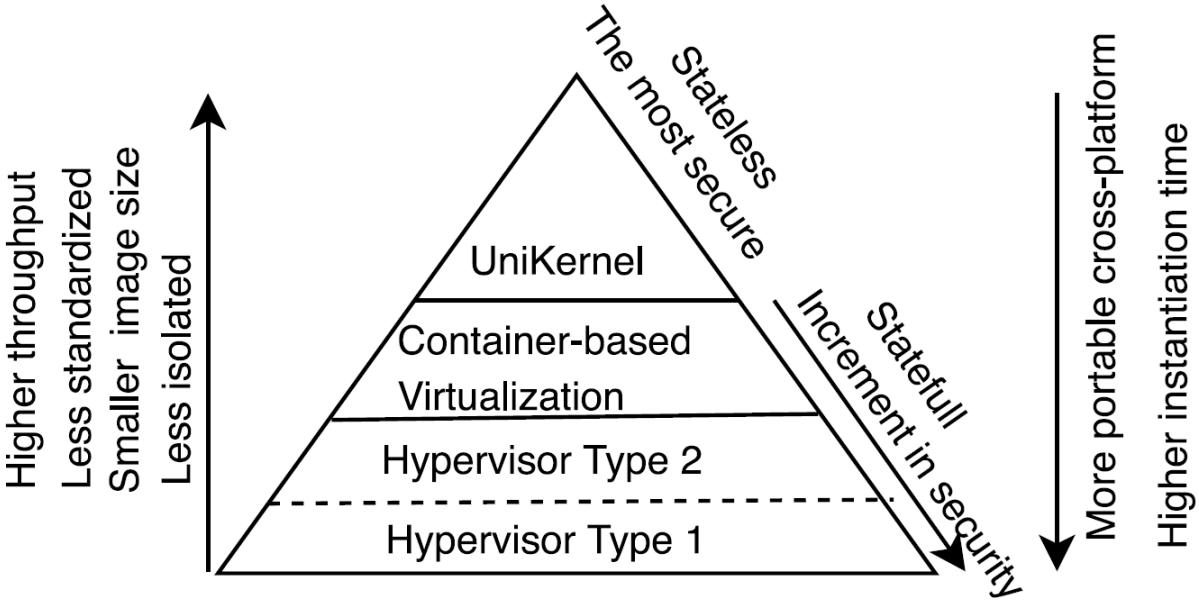

Figure 1. Comparison of various virtualization techniques [30].

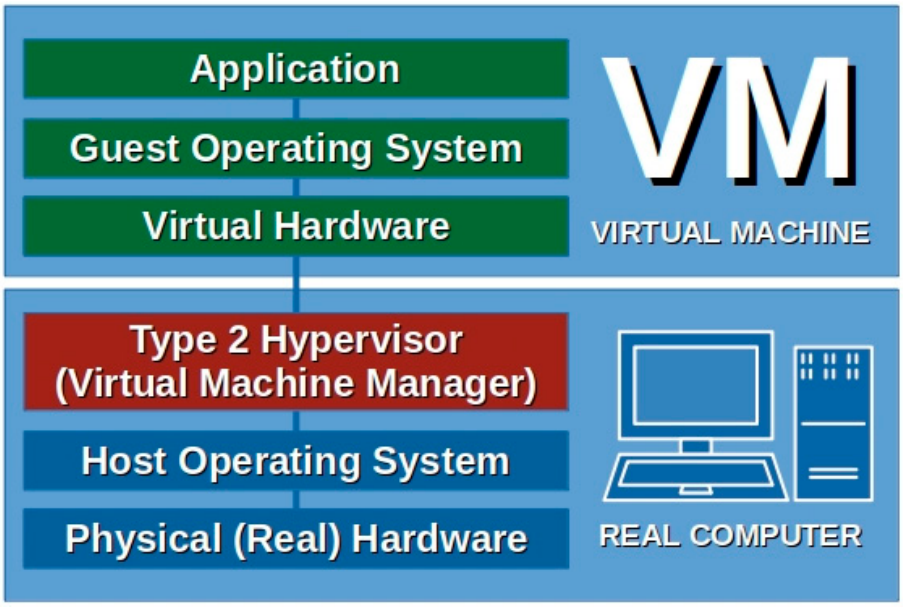

Figure 2. Layers in virtualization architecture based on type 2 hypervisor.

Although the concept of virtualization has been known for many years, its application in the field of education is still limited [31]. Students often find themselves having to make decisions and spend significant time on activities unrelated to the learning process itself and the actual teaching activities, but rather on preparing the virtual machine for the teaching process [32]. While the challenges of heterogeneity have been largely overcome by using virtual hardware, ensuring sufficient uniformity in student usage, problems still persist in the software domain regardless of the virtualization techniques employed.

The crux of the software-related problems lies in the fact that all tasks related to the installation and configuration of necessary software within the virtual machine are carried out by the end user, in this case, the student. Due to this individualistic approach, variations in the interpretation of procedures, execution of tasks, software selection, and other factors

result in a lack of uniformity, deviating from the principle of homogeneity. Moreover, providing a pre-configured virtual machine by the technical staff of the educational institution would not be an ideal solution either. In the first scenario, a single virtual machine would be implemented for the entire institution, encompassing all study levels, programs, and courses. However, this would be resource-intensive, containing software that may not be required by all students, and result in a confusing user experience. The second scenario where individual virtual machines are created for each course would create a large number of virtual machines, burdening the technical staff and resources of the institution (possible oversizing of the system at certain moments, an increase in the level of infrastructure complexity, and additional economic efforts). Both scenarios shift the focus of interest from achieving learning outcomes to the technical provision of conditions, adding a significant burden to all stakeholders involved in the teaching process and potentially hindering the achievement of learning outcomes. It is also assumed that all stakeholders possess sufficient technical knowledge in various domains of computer technology (operating systems, software engineering, virtualization, etc.). However, in practice, there may be instances of lacking basic knowledge of computer technology, or inadequate knowledge, leading to misinterpretation and insufficient information when making crucial decisions and implementing key actions. The tendency for each user to individually carry out the virtual machine installation process based on their own understanding and experience further complicates achieving uniformity throughout the teaching process. This may introduce new unknowns and additional problems that need to be addressed, needlessly increasing the complexity of the teaching process.

To minimize these challenges, ontological engineering can be employed to represent and apply the necessary knowledge in overcoming software-related problems in virtual machine usage. The developed ontology enabling effective software management for virtual machines in higher education will be presented below.

In the literature, there are several approaches to determining the type of ontology based on the subject of classification, as well as various methodologies for ontology development. Without delving into an in-depth analysis, the ontological approach employed here can be classified as an application-dependent methodology, specifically an application ontology [33]. In ontology engineering, several languages have been developed for successful ontology development. For the subject ontology, the advantages offered by the OWL 2 Web Ontology Language [34] will be utilized due to its wide adoption and proven usability [35]. Many ontological solutions in the software domain are implemented using OWL, making it a suitable choice [36]. Additionally, the OWL/XML format, will be used to present the ontology in a more comprehensible format [37], which is better suited for further use in programming languages and applications.

3. Problem Statement

Since the ontology developed within the virtual machine aims to facilitate the installation of appropriate software for teaching activities, it is named StudentSoftwareOntology (the current version of ontology is 1.0), also referred to as StudSoftOnto or SSO for brevity. Table 1 presents the current metrics calculated for the SSO ontology within the Protégé development environment [38].

The complete ontology development process can be carried out using various environments and software tools. In this case, the Protégé 5.6.1 Desktop System, a free and open source software, was employed [39]. Protégé fully supports ontology development using the OWL language [40] and is widely recognized as the most popular environment for ontology development [41]. Within Protégé, the OWLViz 5.0.3 [42] and OntoGraph 2.0.3 [43] plugins allow for adequate visualization of classes and their relationships. Ensuring the consistency of ontologies during development is of utmost importance. Specialized reasoning software, known as reasoners [44], is employed for this purpose. Various programs are available, and in this ontology development project, FaCT++ 1.6.5 [45] and

HermiT 1.4.3.456 [46] were used within the Protégé development environment to assess ontology consistency.

Table 1. Current StudentSoftwareOntology metrics calculated in Protégé software.

Metrics	
Axiom	486
Logical axiom count	344
Declaration axioms count	142
Class count	41
Object property count	25
Data property count	43
Individual count	35
Class axioms	
SubClassOf	39
DisjointClasses	8
Object property axions	
SubObjectPropertyOf	24
InverseObjectProperties	12
ObjectPropertyDomain	25
ObjectPropertyRange	25
Data property axioms	
SubDataPropertyOf	42
DataPropertyDomain	42
Individual axioms	
ClassAssertion	35
ObjectPropertyAssertion	28
DataPropertyAssertion	63

3.1. Identified Classes

The topmost class, owl:Thing, is a predefined class in OWL2 that serves the purpose of providing an appropriate class hierarchy [47]. Under this class, two major subclasses have been defined: CourseRelevant and SoftwareRelevant, as depicted in Figure 3.

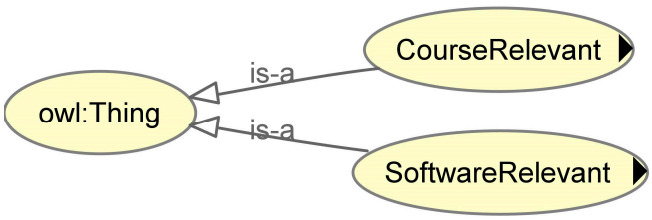

Figure 3. Representation of two major classes in StudentSoftwareOntology using OWLViz plugin in Protégé environment.

To effectively address the problem outlined in the previous chapter, which involves overcoming the heterogeneity of the computer environment in higher education, it is necessary to combine knowledge from two domains. Firstly, knowledge from the domain of higher education is required. This knowledge is organized within the subclasses of the CourseRelevant class, representing the academic domain. Secondly, since we are addressing software-related issues that arise when tackling the complexity of a heterogeneous computer environment, knowledge of the software domain is also essential. The SoftwareRelevant class and its corresponding subclasses facilitate the representation of this software domain knowledge. Therefore, in the hierarchy, the CourseRele-

vant and SoftwareRelevant classes are positioned immediately below the highest-level owl: Thing class.

As mentioned earlier, we will now delve into the details of the CourseRelevant and SoftwareRelevant subclasses, which represent separate domains.

Figure 4 illustrates the key subclasses within the CourseRelevant class. These subclasses were identified by referring to the system of higher education in the Republic of Serbia, as outlined in the Law on Higher Education [48]. The institution itself (class Institution) plays a crucial role in carrying out activities related to academic education. It implements study programs at different levels, referred to as LevelOfStudies class, following the national classification of study levels within the Serbian educational system [49]. Study programs are organized into various study modules (Module class). Each course (Course class) attended by students is associated with a specific module within the study program, and each course is offered in a particular semester (Semester class). The subclasses within CourseRelevant provide more detailed implementation, as depicted in Figure 4.

Figure 4. CourseRelevant subclasses hierarchy using OWLViz plugin in Protégé environment.

Institutions that are legally authorized to offer study programs at different levels (as observed in the higher education model of the Republic of Serbia) include academies (Academy class) with their colleges (College class) and universities (University class) with their institutes (Institute class) and faculties (Faculty class). Despite any institutional affiliations (e.g., a faculty or institute belonging to a university), all institution classes are placed at the same hierarchical level. This is because while they share common characteristics and interests, they should be viewed separately when modeling the teaching process.

The study level subclasses include first, second, and third level studies (LevelI, LevelII, and LevelIII classes), along with special short study programs (ShortStudyProgram class).

First level studies encompass undergraduate academic studies (UndergraduateAcademic class), as well as bachelor (BachelorApplied class) and specialist (BachelorApplied class) applied studies. Second level studies include master (MasterAcademic class) and specialized (SpecializedAcademic class) academic studies, as well as master applied studies (MasterApplied class). Third level studies currently only comprise doctoral academic studies (DoctoralAcademic class).

Similar to the previous approach, we can also identify key subclasses within the SoftwareRelevant class, as shown in Figure 5.

Figure 5. SoftwareRelevant subclasses hierarchy using OWLViz plugin in Protégé environment.

The software itself (Software class) serves as the primary element in this domain. Each software has specific requirements (Requirements class) that must be met for successful implementation, and it is distributed under a specific license (License class). Additional components (AdditionalResources class) may accompany the software to enhance its functionality. To initiate the installation and perform necessary actions, appropriate commands (Command class) or sets of commands are provided. For this system, a Linux distribution is used as the guest operating system within the virtual machine due to its adaptability and

redistributability [50]. Software on Linux systems is typically obtained from repositories (Repository class) during installation [51].

The presented solution categorized software into three types: application software (ApplicationSoftware class) representing the most commonly used software by students; system software (SystemSoftware class), encompassing operating system upgrades and software for system settings; and programming software (ProgrammingTools class) designed for coding, development, testing, debugging, and related tasks. Three types of requirements are defined: minimum (Minimal class), representing the essential hardware and software criteria for normal installation and operation; optimal (Optimal class), reflecting hardware and software criteria for comfortable usage; and recommended (Recommended class), based on user experience from students and technical staff. Additional components (AdditionalResources class) that are not mandatory during installation, but can be added if needed include examples (ExampleFile class), configuration files (ConfigurationFile class), and instructions (ManualFile class) to assist the end user (student) in completing tasks with the provided software. The commands for software installation have been specialized to separate different phases, such as initialization and starting the installation (InstallationCommand class), post-installation commands (PostInstallationCommand class) to add additional functionality in the future, and configuration commands (ConfigurationCommand class) for customization (e.g., changing the file directory).

3.2. Identified Object Properties

In the previous section, we discussed the identification of classes for creating a model to represent the necessary knowledge. However, to make the model usable in real-world scenarios, it is essential to establish appropriate connections between classes, which are realized through object properties in the ontology.

Figure 6 illustrates the object properties as implemented in the Protégé environment. All object properties are defined as sub-properties of owl:topObjectProperty, which serves as the topmost object property in OWL2, providing a hierarchy for object properties.

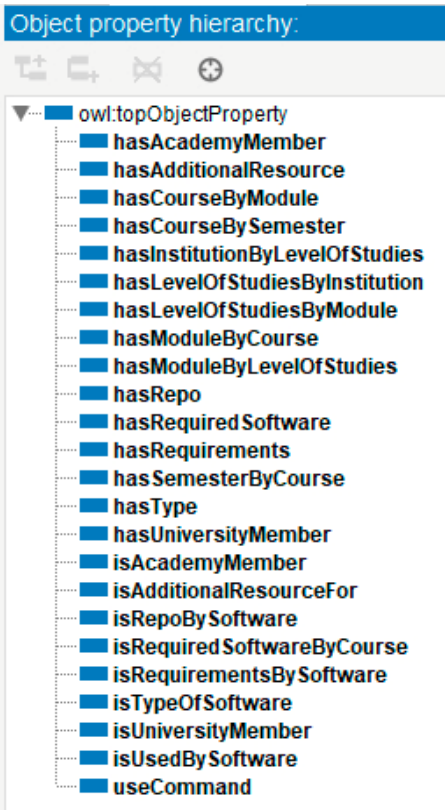

Figure 6. Object properties hierarchy as it is presented in Protégé environment.

It is worth noting that out of the 24 identified object properties, we will focus on explaining only 12 of them, as the other 12 represent inverse object properties, as indicated in Table 1.

Figure 7 showcases the main connections using a visualization generated by the OntoGraf 2.0.3 plugin in the Protégé 5.6.1 desktop environment, based on Graphviz 2.38 software [52].

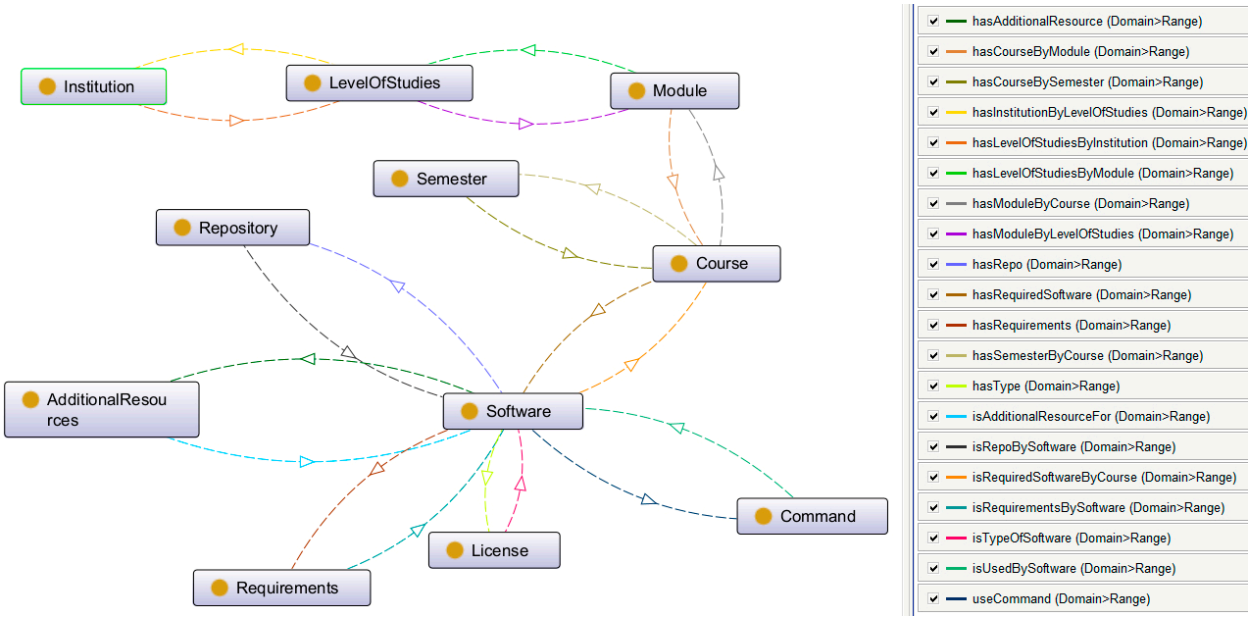

Figure 7. Graphical representation of major relations using OntoGraf plugin in Protégé environment.

Based on the graph representation in Figure 7, we can observe that for the classes related to the CourseRelevant domain (upper half of the graph), the following relationships apply: Institution has LevelOfStudies, which has Modules, and Modules have Courses in specific Semesters.

Similarly, for the classes related to the SoftwareRelevant domain, the following relationships apply: Software has Requirements, Software has License, Software has Command, Software has Repository and Software has AdditionalResources.

Applying the principle of ontological engineering, which connects the subject with the corresponding object through the appropriate predicate, we define the connections for subclasses of the CourseRelevant class in Table 2, and for subclasses of the SoftwareRelevant class in Table 3.

Table 2. Subject, predicate, and object for CourseRelevant subclasses.

Subject (Domain)	Object Property (Predicate)	Object (Range)
Institution	hasLevelOfStudiesByInstitution	LevelOfStudies
LevelOfStudies	hasModuleByLevelOfStudies	Module
Module	hasCourseByModule	Course
Course	hasSemesterByCourse	Semester

Table 3. Subject, predicate, and object for SoftwareRelevant subclasses.

Subject (Domain)	Object Property (Predicate)	Object (Range)
Software	hasRequirements	Requirements
Software	hasType	License
Software	useCommand	Command
Software	hasRepo	Repository
Software	hasAdditionalResource	AdditionalResources

The previous tables show the main predicates graphically represented in Figure 7. In addition, there are specific predicates related to subclasses of the Institution class that define certain institutional affiliations, as shown in Table 4.

Table 4. Subject, predicate, and object for Institution subclasses.

Subject (Domain)	Object Property (Predicate)	Objects (Ranges)
Academy	hasAcademyMember	College
University	hasUniversityMember	Institute, Faculty

To establish a connection between CourseRelevant and SoftwareRelevant domains and obtain relevant knowledge using the ontology, we connect the subject (domain) Course from CourseRelevant and object (range) Software from SoftwareRelevant using the hasRequiredSoftware predicate (object property). This results in the statement:

Course hasRequiredSoftware Software.

By achieving this complete connection within the ontology, we can effectively utilize the ontology to represent the desired knowledge. It is important to note that each predicate has a corresponding inverse predicate, enabling the establishment of inverse connections, although they are not discussed separately in this context.

3.3. Identified Data Properties

In the previous sections, we discussed the identification of classes and the establishment of connections in ontology. However, to make the ontology comprehensive, it is crucial to introduce attributes that provide the necessary characteristics for each instance.

For example, if we have an instance called Faculty_1 belonging to the Institution class, we will only know about its existence. However, we would want to know details such as the institution's name, location, and state. These pieces of information are stored in the corresponding attributes, highlighting the importance of defining attributes alongside classes and connections.

Attributes in the ontology are referred to as data properties. They allow us to store specific information about instances of classes. Figure 8 illustrates the data properties used in the ontology.

Data properties are defined as sub-properties of owl:topDataProperty which serves as the topmost data property in the ontology hierarchy.

Given the extensive list of data properties displayed in Figure 8, it is not necessary to delve into each one individually. However, it is worth noting that a total of 42 data properties have been identified, as indicated in the metrics shown in Table 1. These data properties are based on four data types, with xsd:string being the most commonly used, followed by xsd:anyURI, xsd:integer, and xsd:decimal. For the purpose of this ontology, no additional user-defined data types were needed, and only primitive data types defined within XSD (XML Schema Definition) were utilized [53].

It is important to emphasize that the existing ontology can easily accommodate the introduction of new data properties alongside the existing ones, allowing it to adapt to changes in the environment it serves. This flexibility enables the ontology to react to modifications in key software properties or similar changes, without requiring alterations to classes and connections.

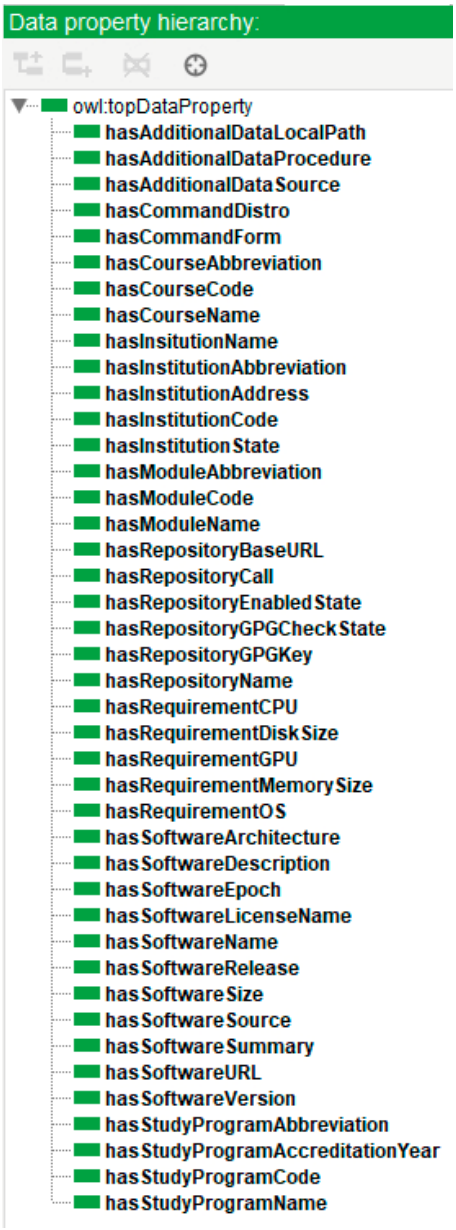

Figure 8. Data properties hierarchy as it is presented in Protégé environment.

3.4. Predefined Individuals and Reasoning

In ontologies, individuals represent instances of classes. While most individuals are provided as input by users, there are also pre-defined individuals that are created during the ontology design phase. In the observed ontology, individuals for the Semester class were included since they are standardized and unique within the educational system, with no variations across institutions. Additionally, certain individuals were created for the License class, as there are well-known and defined types of licenses that remain unchanged and are used as they are.

When entering individuals from the user's side, it refers to an authorized person from the higher education institution (such as technical staff or teaching staff) who enters the necessary details for each individual element through appropriate forms and automated functions. This process ensures accurate entry of the individual's name, assignment to the appropriate class (individual type), and establishment of object property assertions and data property assertions.

These defined elements provide the necessary knowledge for the end user, the student, to apply them correctly during the software installation process, as discussed in previous sections. To ensure the adequacy and absence of deficiencies in the ontology, two reasoners, FaCT++ 1.6.5 and HermiT 1.4.3.456, were employed during the ontology design. Figure 9 shows the log record generated within the ontology development environment, indicating that both reasoners started without any errors.

```
INFO  15:48:11  ------------------------------ Running Reasoner ------------------------------
INFO  15:48:11  Pre-computing inferences:
INFO  15:48:11      - class hierarchy
INFO  15:48:11      - object property hierarchy
INFO  15:48:11      - data property hierarchy
INFO  15:48:11      - class assertions
INFO  15:48:11      - object property assertions
INFO  15:48:11      - same individuals
INFO  15:48:11  Ontologies processed in 33 ms by FaCT++
INFO  15:48:11
INFO  15:48:33  ------------------------------ Running Reasoner ------------------------------
INFO  15:48:33  Pre-computing inferences:
INFO  15:48:33      - class hierarchy
INFO  15:48:33      - object property hierarchy
INFO  15:48:33      - data property hierarchy
INFO  15:48:33      - class assertions
INFO  15:48:33      - object property assertions
INFO  15:48:33      - same individuals
INFO  15:48:33  Ontologies processed in 456 ms by HermiT
INFO  15:48:33
```

Figure 9. Protégé log details related to running reasoners.

As depicted in the figure, the reasoner performs several tasks to check the ontology's consistency. It examines the defined hierarchies, including class hierarchy, object property hierarchy (predicates), and data property hierarchy (attributes). The reasoner also processes and analyzes fact statements, including class and object property assertions, and performs analysis of individuals. In the Protégé environment, any inconsistencies encountered by the reasoner are reported in the error log, providing clear and unambiguous information about the nature of the inconsistency. Additional details about specific inconsistencies can be obtained within the environment itself. However, it is important to note that reasoners are not infallible tools for validating ontologies. While they can check for consistency, they cannot verify if the ontology accurately represents the desired semantics in its entirety.

4. Results and Discussion

In this paper, formal ontology was developed to facilitate knowledge representation in a specific domain. The ontology allows for the formulation of statements about the domain, which can later be used for reasoning and analysis based on these statements and their components [54]. These statements are derived from the classes, object properties (connections), and data properties (attributes) defined within the ontology. Individual instances and their roles within the domain can be defined based on this knowledge [55]. Description logic, a low-level knowledge representation technique [56], is employed as the basis for the ontology. It is worth noting that ontologies can be used effectively for encoding various types of data for further analysis [57].

With the realized ontology realized, the necessary knowledge from the academic and software domains can be effectively connected to obtain usable knowledge related to software management in higher education. The ontology clearly establishes the connection between the course being studied and the software required for its implementation. The course is precisely defined within the ontology, specifying the educational institution, study program, module, and semester in which it is implemented, enabling accurate determination of the student's software needs. On the other hand, software is described through its attributes, such as name, epoch, version, release, and architecture, following the representation of software in Linux. The relationship between software and the repository from which it can be obtained is also defined, along with the necessary commands for installation. In order to avoid potential unwanted situations in the sense of trying to

install software in an inadequate environment, the relationship between software and requirements is also defined.

However, integrated knowledge alone would not hold much significance if it cannot be accessed in a purposeful manner, extracting the relevant information needed for specific actions or tasks. For this purpose, dedicated software is implemented to analyze the knowledge within the ontology and synthesize usable information, which can be further processed based on input parameters. The analysis of ontology for specific inputs is performed using the appropriate implementation developed using the SPARQL query language [58], which is suitable for data querying and integration into applications regardless of the programming language used [59].

SPARQL queries, error elimination, and testing were conducted using the Snap SPARQL Query 6.0.0 plugin within the Protégé 5.6.1 desktop environment. The HermiT 1.4.3.456 reasoner, a Description Logic (DL) reasoner [60] based on hypertableau calculus and supporting OWL [61], was employed for query execution. The HermiT reasoner provides support for SPARQL queries through an appropriate wrapper [62], enabling query answering and reporting any inconsistencies [63]. By utilizing the reasoner during querying, queries can be simplified, and additional information that may not be accessible through regular querying can be retrieved.

Figure 10 provides an example of a realized SPARQL query and the information obtained from its execution.

```
prefix : <http://stolic.rs/onto/StudentSoftwareOntology#>

select ?faculty ?studies ?module ?course ?software ?commandform ?name ?epoch ?version ?release ?architecture
where
{
?facultystr a :Faculty .
?facultystr :hasInstitutionCode "TFB" .
?facultystr :hasLevelOfStudiesByInstitution ?studiesstr .
?studiesstr a :UndergraduateAcademic .
?studiesstr :hasStudyProgramName "Rudarsko inzenjerstvo" .
?studiesstr :hasModuleByLevelOfStudies ?modulestr .
?modulestr :hasModuleCode "RTOR".
?modulestr :hasCourseByModule ?coursestr .
?coursestr :hasCourseName "Procesna merna tehnika" .
?coursestr :hasRequiredSoftware ?softwarestr .
?softwarestr :useCommand ?command .
?command a :InstallationCommand .
?command :hasCommandForm ?commandstr.
?softwarestr :hasSoftwareName ?namestr .
optional {?softwarestr :hasSoftwareEpoch ?epochstr} .
?softwarestr :hasSoftwareVersion ?versionstr .
?softwarestr :hasSoftwareRelease ?releasestr .
?softwarestr :hasSoftwareArchitecture ?architecturestr .
bind(strafter(str(?facultystr),"#") as ?faculty)
bind(strafter(str(?studiesstr),"#") as ?studies)
bind(strafter(str(?modulestr),"#") as ?module)
bind(strafter(str(?coursestr),"#") as ?course)
bind(strafter(str(?softwarestr),"#") as ?software)
bind(str(?commandstr) as ?commandform)
bind(str(?namestr) as ?name)
bind(str(?epochstr) as ?epoch)
bind(str(?versionstr) as ?version)
bind(str(?releasestr) as ?release)
bind(str(?architecturestr) as ?architecture)
}
```

Execute

?faculty	?studies	?module	?course	?software	?commandform	?name	?epoch	?version	?release	?architecture
Technical_faculty_in_Bor	Rudarsko_inzenjerstvo	RTOR	Procesna_merna_tehnika	Octave	sudo dnf -y install	octave	6	6.4.0	5.fc36	x86_64
Technical_faculty_in_Bor	Rudarsko_inzenjerstvo	RTOR	Procesna_merna_tehnika	PSPP	sudo dnf -y install	pspp		1.6.2	4.fc36	x86_64

Figure 10. Execution of SPARQL query in Protégé environment.

As shown in the provided code, the names of the classes, connections, and attributes previously mentioned are used. To narrow down the selection to specific instances of classes and determine their interdependencies, we utilize SPARQL language queries. We introduce variables, define conditions, and use SPARQL functions to manipulate the data accordingly. It is important to note that additional mechanisms are employed to handle missing values.

SPARQL queries alone are only part of the software solution. The goal is to automatically interpret and synthesize the data into appropriate commands that can be applied to the virtual machine without requiring direct interaction from the student. We would not want environment installation, data interpreting, or commands integrated by the students themselves. To achieve this, a dedicated application was developed in the Python programming language (compiler version 3.10.12). Python has been widely used in ontological engineering [64], and the owlready2 library was chosen for its compatibility with OWL2 syntax and support for SPARQL queries [65].

The owlready2 0.39 library, which includes the HermiT reasoner [66], was used for software implementation. The HermiT reasoner does not require explicit inclusion or additional settings, making it straightforward to implement SPARQL queries generated from the Protégé development environment. This ensures the consistency of the obtained data between the software and the testing environment.

The information obtained from the ontology is easily processed in Python using lists. Python offers various libraries for system engineering, and the os Python standard library [67] (version is the same as the version of the Python compiler) was used in this specific case. It allows the implementation of appropriate shell commands within the Linux distribution on which the virtual machine operates. This enables the transfer of synthesized knowledge from the ontology into the actual operation of the virtual machine.

The input data are provided through a list of elements gathered from the existing higher education institution information system for the corresponding user, in this case, the student. In the example depicted in Figure 10, the student is studying at the Technical Faculty in Bor (TFB), pursuing an undergraduate academic study program called Mining Engineering (Rudarsko inzenjerstvo), within the Recycling Technologies and Sustainable Development module (RTOR). The specific course being studied is Process Measurement Technique (Procesna merna tehnika). Based on these input data, the corresponding SPARQL query retrieves information about the software used in the course. Attributes such as software name, epoch, version, release, and architecture are read, and using the commandform structure for installing software on Linux, obtained from the ontology, the appropriate installation command is generated for each software instance. These commands are then executed in the virtual machine's operating system using system commands provided by the os Python standard library.

For the example shown in Figure 10, two software items, Octave and PSPP, are found based on specified criteria. The realized software knows that, based on the values obtained from the query, the installation command for Octave in the virtual machine will be as follows:

```
?commandform+" "+?name+"-"+?epoch+":"+?version+"-"+?release+"."+?architecture.
```

Also, for the installation of the PSPP software, the realized software knows that, based on the values gathered by the query from the ontology, the command in the virtual machine will be as follows:

```
?commandform+" "+?name+"-"+?version+"-"+?release+"."+?architecture.
```

As evident from the above, the implemented software correctly recognizes the information obtained from the ontology and generates the appropriate command based on the obtained data structure for forwarding to the virtual machine. It successfully identifies various features of the Linux environment used within the virtual machine and effectively handles missing values. This ensures that users do not install incorrect software, apply incorrect commands, install the wrong version, or attempt to install incompatible software based on architecture and other factors.

Testing Results

To test the proposed solution, a Fedora 36 XFCE Linux distribution was chosen, and the testing was conducted in two different scenarios: laboratory conditions and real conditions. Both scenarios utilized the Oracle VirtualBox 7.0.8 r156789 (Qt5.15.2) type 2 hypervisor for virtual machine management. The software used in the mining engineering teaching process at the Technical Faculty in Bor [68], which is well documented [69], and includes verified test examples [70], was employed for testing.

In the first scenario, two clean virtual machines were created. Several software applications were installed on one virtual machine using conventional methods. The basic ontology was then transferred to that virtual machine, and the relevant data related to the previously installed software was entered using appropriate methods. After completing the data entry into the ontology, the developed software was transferred to the second virtual machine. The second virtual machine then accessed the ontology on the first virtual machine, executed appropriate queries, extracted relevant information, and performed the necessary actions for software installation on itself. Consequently, the software installation on the second virtual machine was fully automated without any user intervention. After the installation procedures, both virtual machines were restarted, and the installed software was launched on both virtual machines. Identical behavior and the absence of errors were observed on both virtual machines. Although there were slight delays in response times at certain moments due to resource sharing, these were attributed to the virtual machines running on the same computer. Overall, the first test was successfully conducted, demonstrating that the proposed solution effectively accomplishes its tasks under controlled conditions.

The second test simulated a real teaching environment within a highly heterogeneous computer setup. Seventeen students participated in the test, using their personal computers at home. Similarly, a clean virtual machine was created, and students downloaded it onto their local computers to install within the hypervisor. The ontology with relevant data was hosted on a designated within the educational institution, accessible exclusively from that location. Students downloaded the appropriate software and, in conjunction with the ontology, enabled the uniform installation of software on their virtual machines. Once the virtual machines were started and the software on the host system was initiated, further interaction with the students was unnecessary. The software connected to the ontology retrieved relevant information and executed the necessary actions for software installation on each virtual machine. It is important to note that the software was started on the host system, while the installation was performed on the guest system (virtual machine) facilitated network virtualization resources. After the installation procedures, the virtual machines were automatically restarted. Each student was then required to start the installed software and perform a series of pre-prepared tests, such as running specific codes and conducting calculations. Students were instructed to note any observed errors, deviations from the expected behavior, or behaviors they considered to be illegitimate. An identical series of tests were conducted on the reference virtual machine, and the results from each student were compared to the results obtained on the reference machine. The results were identical across all tests, and none of the students reported any errors or deviations from the standard behavior.

The only instance of non-standard behavior was reported by students using Windows as their host operating system. They encountered an issue with the antivirus software flagging the ontology software. Subsequent analysis determined that this was a false positive report triggered by the fact that the software was not digitally signed. Therefore, these reports were dismissed.

Based on the above, it can be concluded that the developed software, ontology, and applied methods successfully fulfilled their intended tasks without any errors or deviations from the expected behavior.

5. Conclusions

In recent years, higher education has undergone significant digital transformation, including the adoption of remote teaching processes where students use their personal digital resources. While this approach offers many benefits, it also introduces challenges, particularly due to the highly heterogeneous computer environment students use. This heterogeneity adds complexity to the teaching process that needs to be effectively managed.

One way to address the computer environment's heterogeneity is by utilizing virtual machines in higher education institutions. Virtual machines provide a level of abstraction that can create a more homogeneous environment for students. However, challenges still exist even with virtual machines.

A notable observation is that implementing tens or even hundreds of different virtual machines to cover all modules and courses within a higher education institution is not practical. Managing such a large number of virtual machines, or a smaller number of complex ones, would strain the institution's resources and staff. Additionally, students would need to perform additional activities, such as software installation and configuration, alongside their regular coursework. This diversion of focus can hinder learning outcomes.

To overcome these potential problems, the paper proposes an automated approach for virtual machine implementation in teaching processes specifically regarding the preparation of software used within the virtual machine. Instead of deploying multiple virtual machines, only one basic virtual machine is implemented for the entire institution and then locally upgraded using built-in automation mechanisms. By leveraging ontological engineering principles, the complexity and resources required for the process are significantly reduced. Notably, the virtual adapts to the student, relieving both students and staff of unnecessary burdens.

The proposed solution underwent successful testing in laboratory conditions and real-world scenarios, confirming its usability. Furthermore, the solution demonstrates sustainability, as it can be easily upgraded and adapted, aligning with modern reuse principles. It enables knowledge sharing not only within a single institution but also among different institutions.

The proposed solution aligns with the FAIR principles, which emphasize the improvement of findability, accessibility, interoperability, and reuse of digital assets [71]. The ontology obtained from this solution can be easily shared and accessed on the internet, ensuring findability and accessibility. In terms of interoperability, ontological engineering principles provide a high potential for integrating and combining the obtained ontology with other ontologies. Furthermore, the resulting solution can be adapted to different environments, accommodating changes in its concepts, connections, and attributes. While the current solution targets virtual machines based on the Linux system, with suitable adjustments and the adoption of relevant rules and mechanisms, it can be applied to other operating systems as well.

Importantly, the solution described in this paper represents the first step towards automated virtual machines for higher education, aiming to achieve software and hardware unification across students' computers. Traditionally, more emphasis has been placed on hardware, while software issues were left to individual users. This approach shifts the focus to automated processes without introducing additional complexity to the teaching process. It offers various benefits to participants in the teaching process, including students' uninterrupted focus on learning and reduced workload for teaching staff. The solution also facilitates knowledge transfer between academic institutions and simplifies software installation procedures, benefiting both software vendors and academic institutions.

Future iterations will likely involve reusing the obtained ontology in other academic education domains, connecting it with existing ontologies, and creating new ontologies to address specific challenges through synergies in the academic education domain. Given the alignment with current and future trends in the digital transformation of higher education, it is expected that the solution presented in this paper will have broad applicability and contribute to the evolving landscape of education.

Author Contributions: Conceptualization, P.S. and D.M.; methodology, P.S. and D.M.; software, P.S. and I.R.; validation, D.M. and Z.S.; formal analysis, P.S., D.M., and Z.S.; investigation, P.S., D.M., Z.S. and I.R.; resources, P.S., D.M. and Z.S.; data curation, P.S. and D.M.; writing—original draft preparation, P.S. and D.M.; writing—review and editing, P.S., D.M., Z.S. and I.R.; visualization, P.S. and Z.S.; supervision, D.M. and Z.S.; project administration, P.S. and Z.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data supporting reported results are included within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Du, X.; Tang, S.; Lu, Z.; Wet, J.; Gai, K.; Hung, P.C.K. A Novel Data Placement Strategy for Data-Sharing Scientific Workflows in Heterogeneous Edge-Cloud Computing Environments. In Proceedings of the 2020 IEEE International Conference on Web Services (ICWS), Beijing, China, 19–23 October 2020; pp. 498–507. [CrossRef]
2. Verba, N.; Chao, K.M.; Lewandowski, J.; Shah, N.; James, A.; Tian, F. Modeling industry 4.0 based fog computing environments for application analysis and deployment. *Future Gener. Comput. Syst.* **2019**, *91*, 48–60. [CrossRef]
3. Javadzadeh, G.; Rahmani, A.M. Fog Computing Applications in Smart Cities: A Systematic Survey. *Wireless Netw.* **2020**, *26*, 1433–1457. [CrossRef]
4. Kumar, V.; Laghari, A.A.; Karim, S.; Shakir, M.; Brohi, A.A. Comparison of fog computing & cloud computing. *Int. J. Math. Sci. Comput.* **2019**, *1*, 31–41. [CrossRef]
5. Ghobaei-Arani, M.; Souri, A.; Rahmanian, A.A. Resource Management Approaches in Fog Computing: A Comprehensive Review. *J. Grid Comput.* **2020**, *18*, 1–42. [CrossRef]
6. Li, Q.; Kumar, P.; Alazab, M. IoT-assisted physical education training network virtualization and resource management using a deep reinforcement learning system. *Complex Intell. Syst.* **2022**, *8*, 1229–1242. [CrossRef]
7. Lattner, C.; Amini, M.; Bondhugula, U.; Cohen, A.; Davis, A.; Pienaar, J.; Riddle, R.; Shpeisman, T.; Vasilache, N.; Zinenko, O. MLIR: A compiler infrastructure for the end of Moore's law. *arXiv* **2020**, arXiv:2002.11054. [CrossRef]
8. Lattner, C.; Amini, M.; Bondhugula, U.; Cohen, A.; Davis, A.; Pienaar, J.; Riddle, R.; Shpeisman, T.; Vasilache, N.; Zinenko, O. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), Seoul, Republic of Korea, 27 February–3 March 2021; pp. 2–14. [CrossRef]
9. Bogdandy, B.; Tamas, J.; Toth, Z. Digital Transformation in Education during COVID-19: A Case Study. In Proceedings of the 2020 11th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), Mariehamn, Finland, 23–25 September 2020; pp. 000173–000178. [CrossRef]
10. Alashhab, Z.R.; Anbar, M.; Singh, M.M.; Leau, Y.B.; Al-Sai, Z.A.; Alhayja'a, S.A. Impact of coronavirus pandemic crisis on technologies and cloud computing applications. *J. Electron. Sci. Technol.* **2021**, *19*, 100059. [CrossRef]
11. Qasem, Y.A.M.; Abdullah, R.; Jusoh, Y.Y.; Atan, R.; Asadi, S. Cloud Computing Adoption in Higher Education Institutions: A Systematic Review. *IEEE Access* **2019**, *7*, 63722–63744. [CrossRef]
12. Luchini, G.; Alegre-Requena, J.V.; Funes-Ardoiz, I.; Paton, R.S. GoodVibes: Automated thermochemistry for heterogeneous computational chemistry. *F1000Research* **2020**, *9*, 291. [CrossRef]
13. Rocha, A.R.; Travassos, G.H.; de Oliveira, K.M.; Villela, K.; Santos, G.; de Menezes, C.S. Ontologies in Software Development Environments. In *Engineering Ontologies & Ontologies for Engineering*; Almeida, J.P.A., Guizzardi, G., Eds.; Nemo: Helsinki, Finland, 2020; pp. 23–35, ISBN 978-1393963035.
14. Hao, Y.; Yu, X. Ontology-based Software Trustworthy Requirements and Behavior Modeling. In Proceedings of the 2021 International Conference on Networking, Communications and Information Technology (NetCIT), Manchester, UK, 26–27 December 2021; pp. 460–466. [CrossRef]
15. Alsanad, A.A.; Chikh, A.; Mirza, A. A Domain Ontology for Software Requirements Change Management in Global Software Development Environment. *IEEE Access* **2019**, *7*, 49352–49361. [CrossRef]
16. Eito-Brun, R.; Gómez-Berbís, J.M.; de Amescua Seco, A. Knowledge tools to organise software engineering Data: Development and validation of an ontology based on ECSS standard. *Adv. Space Res.* **2022**, *70*, 485–495. [CrossRef]
17. Bhatia, M.P.S.; Kumar, A.; Beniwal, R.; Malik, T. Ontology driven software development for automatic detection and updation of software requirement specifications. *J. Discret. Math. Sci. Cryptogr.* **2020**, *23*, 197–208. [CrossRef]

18. Rocha, R.; Bion, D.; Azevedo, R.; Gomes, A.; Cordeiro, D.; Leandro, R.; Silva, I.; Freitas, F. A Syntactic and Semantic Assessment of a Global Software Engineering Domain Ontology. In Proceedings of the 22nd International Conference on Information Integration and Web-based Applications & Services (iiWAS '20), Chiang Mai, Thailand, 30 November–2 December 2020; pp. 253–262. [CrossRef]
19. Mejhed Mkhinini, M.; Labbani-Narsis, O.; Nicolle, C. Combining UML and ontology: An exploratory survey. *Comput. Sci. Rev.* **2020**, *35*, 100223. [CrossRef]
20. Yang, L.; Cormican, K.; Yu, M. Ontology-based systems engineering: A state-of-the-art review. *Comput. Ind.* **2019**, *111*, 148–171. [CrossRef]
21. Tebes, G.; Peppino, D.; Becker, P.; Matturro, G.; Solari, M.; Olsina, L. A Systematic Review on Software Testing Ontologies. In *Quality of Information and Communications Technology*; Communications in Computer and Information Science; Piattini, M., Rupino da Cunha, P., García Rodríguez de Guzmán, I., Pérez-Castillo, R., Eds.; Springer: Cham, Switzerland, 2019; Volume 1010. [CrossRef]
22. Triandini, E.; Kristyanto, M.; Rishika, R.; Rawung, F. A Systematic Literature Review of The Role of Ontology in Modeling Knowledge in Software Development Processes. *IPTEK J. Technol. Sci.* **2021**, *32*, 159–175. [CrossRef]
23. Sikos, L.F. AI in digital forensics: Ontology engineering for cybercrime investigations. *WIREs Forensic Sci.* **2021**, *3*, e1394. [CrossRef]
24. Sharma, S.; Raja, L.; Pallavi Bhatt, D. Role of ontology in software testing. *J. Inf. Optim. Sci.* **2020**, *41*, 641–649. [CrossRef]
25. Agbaegbu, J.; Arogundade, O.T.; Misra, S.; Damaševičius, R. Ontologies in Cloud Computing—Review and Future Directions. *Future Internet* **2021**, *13*, 302. [CrossRef]
26. Nesterenko, O. Technological Trends & Software Engineering Education: A Systematic Review Study. *Probl. Program.* **2022**, *3–4*, 107–116. [CrossRef]
27. Affouneh, S.; Khlaif, Z.N.; Burgos, D.; Salha, S. Virtualization of Higher Education during COVID-19: A Successful Case Study in Palestine. *Sustainability* **2021**, *13*, 6583. [CrossRef]
28. Goel, G.; Tanwar, P.; Bansal, V.; Sharma, S. The challenges and Issues with Virtualization in Cloud Computing. In Proceedings of the 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 3–5 June 2021; pp. 1334–1338. [CrossRef]
29. Bermejo, B.; Juiz, C.; Guerrero, C. Virtualization and consolidation: A systematic review of the past 10 years of research on energy and performance. *J. Supercomput.* **2019**, *75*, 808–836. [CrossRef]
30. Mansouri, Y.; Babar, M.A. A review of edge computing: Features and resource virtualization. *J. Parallel Distrib. Comput.* **2021**, *150*, 155–183. [CrossRef]
31. Huang, A. Teaching, Learning, and Assessment with Virtualization Technology. *J. Educ. Technol. Syst.* **2019**, *47*, 523–538. [CrossRef]
32. Wazan, A.S.; Kuhail, M.A.; Hayawi, K.; Venant, R. Which Virtualization Technology is Right for My Online IT Educational Labs? In Proceedings of the 2021 IEEE Global Engineering Education Conference (EDUCON), Vienna, Austria, 21–23 April 2021; pp. 1254–1261. [CrossRef]
33. Stancin, K.; Poscic, P.; Jaksic, D. Ontologies in education–state of the art. *Educ. Inf. Technol.* **2020**, *25*, 5301–5320. [CrossRef]
34. OWL 2 Web Ontology Language Quick Reference Guide (Second Edition), W3C Recommendation 11 December 2012; World Wide Web Consortium. Available online: http://www.w3.org/TR/2012/REC-owl2-quick-reference-20121211/ (accessed on 10 June 2023).
35. Khair, A.M.M.; Meziane, F. The Use of Ontologies In Software Elicitation. *Humanit. Nat. Sci. J.* **2021**, *2*, 427–441. [CrossRef]
36. Qaswar, F.; Rahmah, M.; Raza, M.A.; Noraziah, A.; Alkazemi, B.; Fauziah, Z.; Hassan, M.K.A.; Sharaf, A. Applications of Ontology in the Internet of Things: A Systematic Analysis. *Electronics* **2023**, *12*, 111. [CrossRef]
37. Popereshnyak, S.; Vecherkovskaya, A. Modeling Ontologies in Software Testing. In Proceedings of the 2019 IEEE 14th International Conference on Computer Sciences and Information Technologies (CSIT), Lviv, Ukraine, 17–20 September 2019; pp. 236–239. [CrossRef]
38. Protégé. Available online: https://protege.stanford.edu/ (accessed on 10 June 2023).
39. Musen, M.A. The Protégé project: A look back and a look forward. *AI Matters* **2015**, *1*, 4–12. [CrossRef]
40. Ageed, Z.S.; Ibrahim, R.K.; Sadeeq, M.A. Unified Ontology Implementation of Cloud Computing for Distributed Systems. *Curr. J. Appl. Sci. Technol.* **2020**, *39*, 82–97. [CrossRef]
41. Rahayu, N.W.; Ferdiana, R.; Kusumawardani, S.S. A systematic review of ontology use in E-Learning recommender system. *Comput. Educ. Artif. Intell.* **2022**, *3*, 100047. [CrossRef]
42. OWLViz. Available online: https://protegewiki.stanford.edu/wiki/OWLViz#Documentation (accessed on 12 June 2023).
43. OntoGraph. Available online: https://protegewiki.stanford.edu/wiki/OntoGraf (accessed on 12 June 2023).
44. Husáková, M.; Bureš, V. Formal Ontologies in Information Systems Development: A Systematic Review. *Information* **2020**, *11*, 66. [CrossRef]
45. FaCT+ Reasoner. Available online: https://owl.cs.manchester.ac.uk/tools/fact/ (accessed on 12 June 2023).
46. HermiT OWL Reasoner. Available online: https://www.hermit-reasoner.com/ (accessed on 12 June 2023).
47. Sengupta, K.; Hitzler, P. Web Ontology Language (OWL). In *Encyclopedia of Social Network Analysis and Mining*; 2014. Available online: https://corescholar.libraries.wright.edu/cse/184 (accessed on 12 June 2023).

48. Republic of Serbia. Law on Higher Education. 2021. Available online: https://www.paragraf.rs/propisi/zakon_o_visokom_obrazovanju.html (accessed on 14 June 2023).
49. Republic of Serbia, Ministry of Education. Types and Scopes of Studies. 2023. Available online: https://prosveta.gov.rs/prosveta/visoko-obrazovanje/vrste-i-obim-studija/ (accessed on 14 June 2023).
50. What is Linux. Understanding Linux. Red Hat Inc. 2023. Available online: https://www.redhat.com/en/topics/linux/what-is-linux (accessed on 14 June 2023).
51. Repositories. Ubuntu Documentation. Canonical Ltd. 2022. Available online: https://help.ubuntu.com/community/Repositories (accessed on 14 June 2023).
52. Graphviz. Available online: https://graphviz.org/ (accessed on 14 June 2023).
53. XML Schema Part 2: Datatypes, 2nd ed.; World Wide Web Consortium. 2004. Available online: https://www.w3.org/TR/xmlschema-2/ (accessed on 15 June 2023).
54. Stepan, G.; Pascal, H.; Andreas, A. Knowledge Representation and Ontologies. In *Semantic Web Services*; Studer, R., Grimm, S., Abecker, A., Eds.; Springer: Berlin, Germany, 2007. [CrossRef]
55. Tenorth, M.; Beetz, M. Representations for robot knowledge in the KnowRob framework. *Artif. Intell.* **2017**, *247*, 151–169. [CrossRef]
56. Gayathri, R.; Uma, V. Ontology based knowledge representation technique, domain modeling languages and planners for robotic path planning: A survey. *ICT Express* **2018**, *4*, 69–74. [CrossRef]
57. Robinson, P.; Haendel, M. Ontologies, Knowledge Representation, and Machine Learning for Translational Research: Recent Contributions. *Yearb. Med. Inform.* **2020**, *29*, 159–162. [CrossRef] [PubMed]
58. SPARQL 1.1 Overview. World Wide Web Consortium. 2013. Available online: https://www.w3.org/TR/sparql11-overview/ (accessed on 15 June 2023).
59. DuCharme, B. *Learning SPARQL*; O'Reilly Media Inc.: Sebastopol, CA, USA, 2011.
60. Almendros-Jiménez, J.M.; Becerra-Terón, A. Type Checking and Testing of SPARQL Queries. In *Actas de las XVII Jornadas de Programación y Lenguajes (PROLE 2017)*; Durán, F., Ed.; Sistedes: San Sebastian, Spain, 2017; Available online: https://hdl.handle.net/11705/PROLE/2017/002 (accessed on 15 June 2023).
61. Horrocks, I.; Motik, B.; Wang, Z. The HermiT OWL Reasoner. In Proceedings of the 1st International Workshop on OWL Reasoner Evaluation ORE 2012, Manchester, UK, 1 July 2012.
62. Glimm, B.; Horrocks, I.; Motik, B.; Stoilos, G.; Wang, Z. HermiT: An OWL 2 Reasoner. *J. Autom. Reason.* **2014**, *53*, 245–269. [CrossRef]
63. Almendros-Jiménez, J.M.; Becerra-Terón, A.; Cuzzocrea, A. Detecting and Diagnosing Syntactic and Semantic Errors in SPARQL Queries. In Proceedings of the Workshops of the EDBT/ICDT 2017 Joint Conference, Venice, Italy, 21–24 March 2012.
64. Lamy, J.B. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artif. Intell. Med.* **2017**, *80*, 11–28. [CrossRef]
65. Lamy, J.B. *Ontologies with Python: Programming OWL 2.0 Ontologies with Python and Owlready2*; Apress: Berkeley, CA, USA, 2021. [CrossRef]
66. Lamy, J.B. Reasoning. Owlready2 Documentation. 2019. Available online: https://owlready2.readthedocs.io/en/latest/reasoning.html (accessed on 17 June 2023).
67. Os–Miscellaneous Operating System Interfaces. Python Software Foundation. 2023. Available online: https://docs.python.org/3/library/os.html (accessed on 17 June 2023).
68. Stolic, P.; Ivaz, J.; Petrovic, D.; Stevic, Z. Advantages of Mining Engineering Curriculum Realization Using Solutions Based on Free Software. In Proceedings of the 52nd International October Conference on Mining and Metallurgy-IOC 2021, Bor, Serbia, 29–30 November 2021; pp. 221–224, ISBN 978-86-6305-119-5.
69. Stolic, P.; Milosevic, D. Alternative Software Solutions for Ensuring the Continuity of the Teaching Process in Emergency Situations. In Proceedings of the 8th International Scientific Conference Technics and Informatics in Education, Cacak, Serbia, 18–20 September 2020; pp. 196–203, ISBN 978-86-7776-247-6.
70. Stolic, P.; Stanimirovic, Z.; Stanimirovic, I.; Jaric, M.; Radovanovic, I.; Stevic, Z. Application of open source solutions in the realization of low-cost teaching laboratories. In Proceedings of the XXIV International Scientific-Practical Conference "Modern Information and Electronic Technologies", Odesa, Ukraine, 29–31 May 2023; pp. 36–39.
71. Wilkinson, M.; Dumontier, M.; Aalbersberg, I.; Appleton, G.; Axton, M.; Baak, A.; Blomberg, N.; Boiten, J.-W.; da Silva Santos, L.B.; Bourne, P.E.; et al. The FAIR Guiding Principles for scientific data management and stewardship. *Sci. Data* **2016**, *3*, 160018. [CrossRef] [PubMed]