PADS: Design and Implementation of a Cloud-based, Immersive Learning Environment for Distributed Systems Algorithms

Yogesh D. Barve, Prithviraj Patil, Anirban Bhattacharjee and Aniruddha Gokhale, *Senior Member, IEEE*

Abstract—As distributed systems become more complex, understanding the underlying algorithms that make these systems work becomes even harder. Traditional learning modalities based on didactic teaching and theoretical proofs alone are no longer sufficient for a holistic understanding of these algorithms. Instead, an environment that promotes an immersive, hands-on learning of distributed systems algorithms is needed to complement existing teaching modalities. Such an environment must be flexible to support the learning of a variety of algorithms. The environment should also support extensibility and reuse since many of these algorithms share several common traits with each other while differing only in some aspects. Finally, it must also allow students to experiment with large-scale deployments in a variety of operating environments. To address these concerns, we use the principles of software product lines and model-driven engineering, and adopt the cloud platform to design an immersive learning environment called the Playground of Algorithms for Distributed Systems (PADS). A prototype implementation of PADS is described to showcase use cases involving BitTorrent Peer-to-Peer file sharing, ZooKeeper-based coordination, and Paxos-based consensus, which show the benefits of rapid deployment of the distributed systems algorithms. Results from a preliminary user study are also presented.

Index Terms—Learning System, Feature model, Software Product Lines, Distributed Systems, Cloud.

1 Introduction

1.1 Complexities in Learning and Teaching Distributed Systems Algorithms

The design of large-scale networked and distributed systems must handle many complex issues, such as time synchronization, fault management, replication and replica synchronization, consensus among peers, concurrency control and race conditions, leader-election among nodes, deadlock avoidance, etc. Addressing these complex problems is further excarbated by the heterogeneity in distributed systems in terms of network topology (ring, star, mesh, etc.), node types (fixed vs mobile nodes, static vs dynamic nodes, physical vs virtual nodes), communication styles (client-server, peer-to-peer, publish-subscribe, etc.), and network types (Ethernet, WiFi, Satellite). The complexity of these design considerations and various accidental complexities make both the teaching and the learning of algorithms for distributed systems a daunting task for instructors and students, respectively.

From our experience both as students taking a course on Distributed Systems and as an instructor teaching such a course, we observed that existing teaching modalities, tools and techniques for understanding algorithms for distributed systems often rely on traditional approaches, such as didactic lecturing, simple proof sketches on the whiteboard, using theorem provers, and basic simulations or toy assignments in some programming language.

We believe that this approach incurs several difficulties for students including (1) often having to program the algorithms in programming languages they are not experts in, (2) analyzing these algorithms in simulators/emulators they are unfamiliar with, and (3) needing to deal with accidental complexities in order to deploy them on real hardware to realistically validate them or propose improvements and extensions to them. Due to a piecemeal approach in learning and implementing these algorithms (i.e., programming/learning algorithms individually), students (1) cannot analyze multiple algorithms at the same time to compare and contrast them, (2) cannot seamlessly switch between simulation, emulation and real deployment on hardware, and (3) consequently do not obtain a holistic view of distributed systems and how different algorithms work together in real world distributed systems.

The instructor faces a different set of challenges. For instance, the inherent asynchrony and scale of distributed systems makes it hard for an instructor to show students the multiple different execution traces that a distributed system can illustrate in its life time. By no means do we imply that existing teaching modalities are not needed; rather what we propose is that instructors require some way in which they can keep the students engaged after which proof sketches and theorem provers can be utilized. We believe that hands-on, immersive learning [1] where students are allowed to conduct different kinds of "what-if" analyses (e.g., tweaking certain parameters of the distributed algorithm or tweaking some steps in an algorithm) and letting the students observe the impact of their slight modifications may provide a significantly more engaging and rewarding learning experience

• *Yogesh D. Barve, Prithviraj Patil, Anirban Bhattacharjee and Aniruddha Gokhale are with the Department of Electrical and Computer Science, Vanderbilt University, Nashville, TN 37212, USA.*
E-mail: {yogesh.d.barve, prithviraj.p.patil, anirban.bhattacharjee, a.gokhale}@vanderbilt.edu

Manuscript received June 1, 2016; first revised November 8, 2016; second revised February 22, 2017.

for the student, who may then feel obliged to utilize theorem provers and associated tools to prove the correctness of the original and/or the modified algorithms. Unfortunately, we have not come across any readily available capabilities that fulfill these critical needs for distributed systems learning and instruction.

1.2 Solution Approach and Organization of Paper

To address these challenges we need a learning and technology-based approach that will alleviate the need for students to learn an unfamiliar programming language or a simulation tool to experiment with distributed systems. Secondly, such an approach should provide the student with intuitive and higher-levels of abstractions that are closer to the domain and semantics of distributed systems rather than having the student deal with low-level and mundane syntactic details of programming languages and simulation tools. Moreover, the approach should be extensible and enable the student to seamlessly move between simulation, emulation and real-world experimentation. Finally, the approach should promote maximal reuse so that students can build larger distributed systems by composing smaller but intuitive building blocks.

We surmise that these critical needs can be met using Software Product Lines (SPLs) [2] and model-driven engineering (MDE) [3] in the context of cloud platforms that will help improve teaching and learning of distributed systems algorithms. The key intuition behind applying SPL principles stems from the observation that these algorithms tend to share several common traits (e.g., communication paradigm, such as client-server versus publish/subscribe, or the model of computation, such as synchronous data flow or reactive asynchronous) while differing only in some aspects (e.g., the actual publish/subscribe technology used or the protocol encoding for messages).

Consequently, a collection of distributed systems algorithms can be viewed as variants of a product line. The challenge then lies in understanding and capturing the commonality and variability across these algorithms, and developing techniques needed to automate the synthesis of these variants so that the different dimensions of accidental complexities faced by the student can be substantially alleviated. MDE is used to realize these capabilities because it provides the user with intuitive, higher-level abstractions compared to programming languages to model the distributed systems algorithms and use generative techniques to almost completely automate the entire experimental setup including deployment and orchestration.

In the context of using MDE-based approaches, we have focused on visual modeling for imparting learning objectives to the students. Visual approaches for teaching have been found very effective in research projects, such as Betty's Brain [4], Code.org [5], NetsBlox [6, 7] and in our prior work on C3STEM [8, 9, 10, 11]. Both NetsBlox and C3STEM particularly make use of model-driven engineering techniques. To that end, our solution is a learning framework for distributed systems called the *Playground of Algorithms for Distributed Systems (PADS),* that reifies SPL principles by building on MDE with generative capabilities, feature modeling and teaching/learning tools and technologies.

For this paper we complement and build upon our previous work [12] and make the following contributions:

- Concrete details on the design and implementation of the PADS framework beyond that provided in [12] including description of the underlying web-based modeling environment that provides new features such as web-based and collaborative modeling and the learning outcomes addressed by PADS (Section 3).
- Deeper insights into the runtime experimentation and deployment using the PADS framework (Section 4).
- Providing early results from a user study of the PADS framework in a classroom setting (Section 5).

The rest of the paper is organized as follows: Section 2 surveys related research and compares and contrasts them with PADS; Section 3 delves into the design details of PADS focusing the MDE and SPL aspects; Section 4 focuses on the deployment and runtime infrastructure that accompanies PADS; Section 5 provides a validation of PADS including results from an initial user study; and finally Section 6 provides concluding remarks alluding to lessons learned and future work.

2 Related Work

In this section we compare PADS with related efforts along three key dimensions: existing work in teaching specific software for distributed systems algorithms, use of model driven engineering in the design of large-scale software systems in the context of educational learning systems, and visual learning aids.

Learning systems for distributed algorithms teaching: Authors in [13] present a comprehensive survey providing an overview of different tools, simulators and learning platforms available for teaching distributed systems. It outlines tools available for managing deployment, execution, discovery, monitoring and configuration of distributed systems. It also presents a list of algorithms that can be used for teaching and demonstrating intricate details of distributed algorithms.

ViSiDiA [14] is a framework for designing, simulating and visualizing distributed algorithms. It is developed using Java language frameworks. It provides implementations of different distributed systems like sensor networks and mobile agents. A user can specify their custom distributed algorithms by making use of framework specific Java API. Distal [15] is another framework that is specifically aimed at a certain class within the distributed systems algorithm, namely fault-tolerant systems. It is developed on top of the Scala programming framework. One can write pseudo code for the algorithm using its DSML to translate into an executable code. The executable can then be deployed on clusters for testing. However, it lacks integration with simulators that would facilitate quick testing and debugging of algorithms.

Another teaching and learning framework called FADA (Framework Animations of Distributed Algorithms) is presented in [16]. In FADA, the simulations are written using Java programming language using the visualization APIs

provided by the framework. It also provides a set of pre-assembled simulations for different algorithms which can be used as examples for demonstrating distributed algorithms to students.

The frameworks presented above have the following shortcomings compared to our approach. First, the distributed algorithms need to be written in a language which the framework supports. Secondly, the tools presented above do not support seamless translation of programming artifacts from simulation to real world deployment.

MDE in learning systems: Previous work has shown MDE, specifically, domain-specific modeling languages (DSMLs) have being effective tools [17] in developing teaching software systems. Students have also seen the benefits of rapid code generation based on MDE techniques. In [18], students were able to rapidly synthesize code artifacts using MDE to rapidly generate code, when changes where required to be made in platform configuration of robotics control code and mobile device.

An educational game design software framework is presented in [19]. It utilizes a model driven approach to describe educational game concepts. It presents an educational game metamodel that defines platform-independent educational game concepts. The framework aims to design educational games to motivate the students to get involved in the learning process thereby effectively conveying educational material.

SPL techniques were applied for design, development and support of a family of elearning systems [20] called TALES. It also highlighted some of the challenges involved in the development of large-scale educational system and how SPL helped it to gain 10-fold productivity boost in the developmental efforts. The educational systems were built as a part of Adult Literacy Programme (ALP) for teaching illiterates in India in 22 Indian languages. Unlike the work presented above our area of study is focused on a special topic within computer science which is the distributed systems algorithms. Our work leverages the MDE and the SPL techniques in the design of a learning framework for distributed algorithms.

Visual Programming based learning frameworks: Computational thinking (CT) in the recent times, has become focus of many researchers including our prior work in the field of educational computing [9, 21, 22]. CT can be broadly summarized as a problem solving process involving abstraction of problem into smaller separation of concern entities, designing algorithm to solve the problem, simulating the solution approach and verification of the effectiveness of the solution to build a better solution strategy [23]. CT based framework such as the C3STEM [9] engages K-12 students to solve complex real world problem like the traffic flow modelling using a visual programming interface to design, simulate and analyze the solution strategy. Yet another MDE/visual programming tool for learning is called NetsBlox [6, 7] which aims to provide a gentle introduction to distributed computing to K-12 students. PADS also uses the same MDE development environment as NetsBlox and in contrast to NetsBlox is focused on more rigorous distributed systems algorithms.

In [24], authors developed a visual programming toolkit called Alice, to teach object oriented programming to students. Their observation from the student subject study revealed that students learned new concepts in computer science effectively and also developed higher-order thinking skills using the tool. Another useful observation was that the drag-and-drop ability of designing new programs helped students from preventing making syntax errors in their program due to the auto-code generation. Another visual programming platform called code.org [5] has been very popular and serving millions of users worldwide in introducing basics of computer programming. The study in [5] also notes that students developed positive behavior towards programming and also improved their reflective thinking skills towards problem solving. Visual programming based teaching methodology has been found useful to expose students to new teaching concepts.

3 Design and Implementation of PADS

This section delves into the details of the design and implementation of the PADS framework. We first outline the key learning outcomes we intend to address via the framework. Next, we delve into the details of its model-based design including feature model representation and web based modeling environment. Lastly, we describe the roles and responsibilities of PADS's actors which includes students and instructors.

3.1 Underlying Philosophy Behind PADS

In the educational teaching domain, learning objects have been an extremely useful resource in imparting education to the subjects [25, 26, 27, 28, 29]. In the literature, learning objects have been defined as "*Learning Objects (LOs) are digital resources that can be used (and reused) to support the learning process*" [30]. The learning objects have the property of re-usability, and sharing of learning material. As an instructional artifact, the learning objects should be able to easily incorporate in the design of larger educational teaching context. Generative learning objects (GLO) are a class of Learning Objects (LO) from which LO-specific functional implementation properties can be generated [30]. Learning objects are appealing in the space of learning computer science concepts.

Use of learning objects is still not that popular in the area of educational tools for computer science due to difficulties in designing learning objects as reusable software artifacts [25]. In a recent study [31], the authors have incorporated GLO in the teaching of robot programming. Authors have incorporated different robot control programs which are treated as LOs and the code generated for the robot target platform is treated as GLO. Model-driven engineering (MDE) has been shown to be an effective software engineering technique for reusing and sharing of software artifacts. MDE combined with code generation facility allows us to bring the learning paradigms of LO and GLOs in the practical use for our distributed algorithms learning toolkit. In the case of distributed algorithms teaching toolkit, different algorithms models will form the LO, and their target execution environment specific program code will be GLO.

From the viewpoint of learning outcomes, we have focused on a subset of key learning outcomes outlines

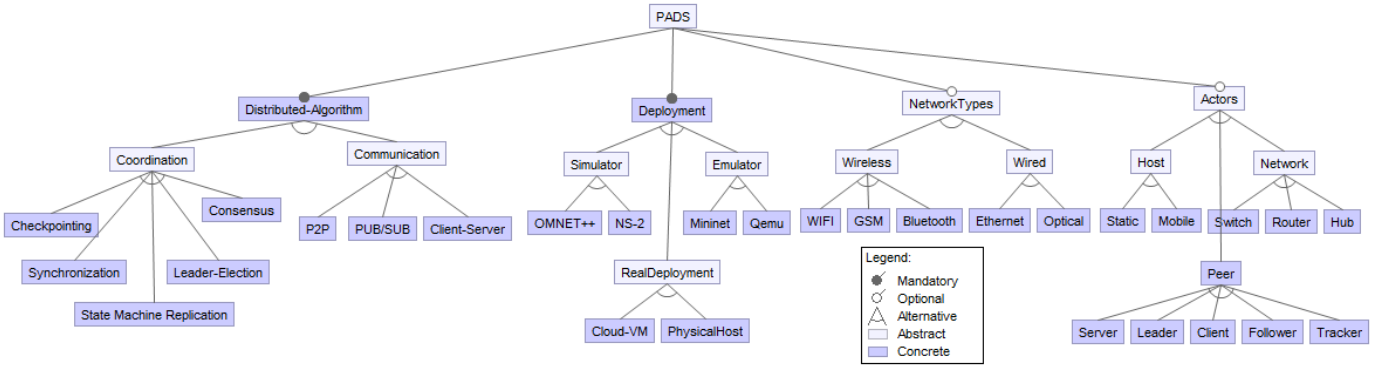

Fig. 1. Feature Model Diagram of Playground of Algorithms for Distributed Systems (PADS) Framework

by the Accredition Board for Engineering and Technology (ABET) engineering criteria [32]. Specifically, for an effective instructional method for educating students, addressing the learning outcomes becomes key in achieving student learning objectives in an institutional course. PADS tries to address the following **General Criteria 3**, student learning outcomes from the ABET engineering program.

- *(a) an ability to apply knowledge of mathematics, science, and engineering.* PADS is designed to enable students to apply the concepts they learn in Distributed Systems and model the algorithm to learn its behavior.
- *(b) an ability to design and conduct experiments, as well as to analyze and interpret data.* PADS is design to enable a student to setup one or more experiments to evaluate a distributed systems algorithm scalably without incurring mundane and error-prone activities stemming from setting up one or more experiments.
- *(c) an ability to design a system, component, or process to meet desired needs.* PADS enables a student to tweak different parameters and modify existing algorithms to understand the impact on the resulting behaviors.
- *(k) an ability to use the techniques, skills, and modern engineering tools necessary for engineering practice.* PADS enables students to use modern tools including MDE tools, emulation environments such as Mininet, virtualization techniques such as virtual machines and containers, and study a variety of networking issues among others as they study distributed systems.

We now present the design and implementation of the Playground of Algorithms for Distributed Systems (PADS), which is an extensible framework that manages a software product line of distributed algorithms used as an instructional and learning aid for distributed systems. It uses MDE and SPL techniques to integrate various distributed systems algorithms for teaching, and cloud platforms for deployment of experiments.

3.2 Feature Model Representation

For a successful SPL for PADS, we need to manage the commonalities and variabilities that are exhibited for realizing the development, implementation and demonstration of distributed algorithms. One of the well-known approaches for representing and managing these commonalities and variabilities is by the means of feature models [33]. Feature models provide proven techniques for improving reusability by specifying reuse rules. The feature model for PADS must capture the commonalities and different dimensions of variabilities incurred in the learning process [12]. To that end we have defined a conceptual feature model for PADS as shown in Figure 1 and described in [12].

3.3 Realizing the PADS Feature Model using Model-driven Engineering

The SPLs can be built using modular software. Changes in the feature configurations can be mapped to the changes in the software modules [34]. The design and development of modular software framework is a challenging task. We use model-driven engineering (MDE) techniques to codify the feature model by mapping it to metamodel(s) of a domain-specific modeling language (DSML) and use generative technologies, which are key artifacts of MDE, to automate the synthesis of product variants of our PADS product line. MDE helps to codify the necessary properties of problem domain which is decoupled from specific solution domain (e.g., simulator specific programming language, target hardware specific). MDE helps to integrate domain knowledge into the metamodels and model interpreters [35]. The MDE design approach helps to map different configuration possibilities for a specific instance of SPL based on where it is deployed. This is useful when one needs to deploy a distributed algorithm software product family onto different platforms/simulators, MDE can bring all software components and their configuration mapped to the platform specific source codes and scripts.

Our PADS framework can be hosted on virtual machines or containers deployed on the Openstack cloud, which is an open source cloud computing infrastructure [36]. In our current prototype, we use containers. Cloud computing provides on-demand access to large pool of shared resources for compute intensive simulations and network experimentation. Students and instructors access these virtual machines/containers using remote access client which could be either a web browser client or a desktop client.

3.4 Web Based Modeling Environment

In our previous work [37], we used the Generic Modeling Environment (GME) [38] as an environment to build and

develop meta-models and models, and model interpreters. Despite the widespread use of GME in designing the DSML as evident by the works presented in [39, 40], it falls short in meeting our requirement of online and collaborative design and development of PADS artifacts. As such we decided to look into the next generation of GME being developed at Vanderbilt University called the WebGME [41].

WebGME is an online browser-based modeling environment. It supports features of GME like creating of meta-models, models, model visualization and model interpreters. Apart from this, it supports the collaborative modeling, version controlling of models, user management, cloud-based infrastructure, and model executors that can run either on the client side web-browser or run in the resourceful cloud infrastructure depending on the user configuration.

WebGME[1] provides an environment to define the syntax and semantics of a DSML through metamodeling. Model interpreters can be defined associated with the metamodels that can provide additional semantics to the language which are not captured in a visual form as well as provide the generative capabilities needed for automation. The WebGME environment can be used to build model instances of a DSML. Thus, in our PADS framework, an instructor can extend existing metamodels for the collection of algorithms by providing a metamodel for a new algorithm. The students use the PADS framework to develop model instances and configure them for their experimental scenarios.

3.5 Roles and Responsibilities of PADS's Actors

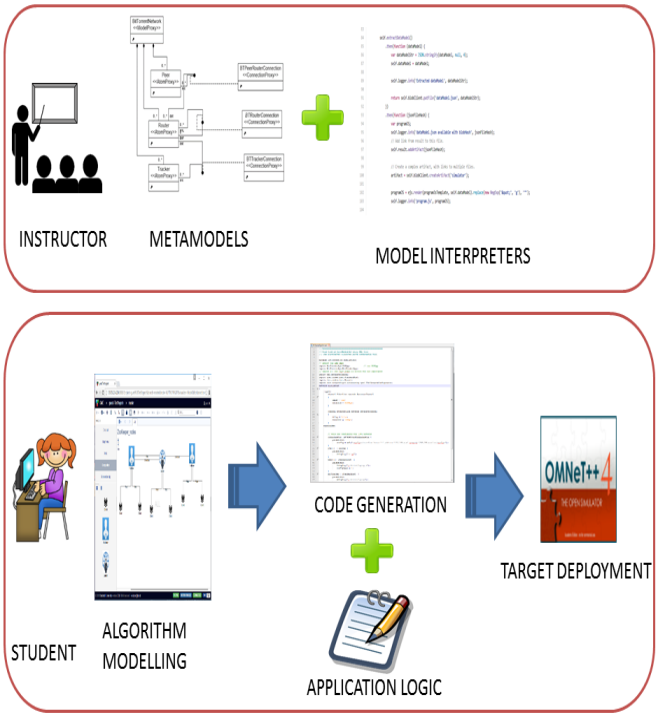

Fig. 2. Model-based Process for Distributed Algorithm Demonstration and Deployment

1. https://webgme.org/

Figure 2 shows the different phases that comprises the use of the PADS framework for teaching distributed systems algorithms. An instructor who is a domain expert with a knowledge of model driven engineering concepts, lays down the foundation aspects as to which different feature model entities are involved in the design and deployment runtime of the algorithms. Based on the specific distributed algorithm concepts, using the PADS builtin meta-model library, the instructor creates a specific meta-model. Now, the software artifacts associated with the meta-model and the necessary model interpreter logic is also written by the instructor. Though we believe this would involve some learning curve for the instructor to get familiar with the PADS meta-modelling and model interpreter programming, the potential benefits of the PADS to the student learning outweighs this limitation. Moreover, we believe that a repository of such individual algorithm-specific metamodels can be envisioned and reused on a large scale.

Based on the type of the deployment (e.g., simulator specific), the associated deployment logic generators are also required to be a part of the model interpreters. Once, this initial design activity is complete, the tool is now ready to be used by the students. Students can now start modelling the distributed algorithm, such as different actors in the distributed systems, their attribute properties, characterizing the communication medium properties used during the simulation, network topology of the distributed systems, runtime platform for distributed systems deployment. Once the student has modelled the distributed systems algorithm, code generation facility can be utilized to generate the boilerplate code for the algorithm. The boilerplate code consists of the information specified during the modelling of the experiment and deployment runtime specific gluecode. Based on the learning activity, the instructor can then ask the students to write the application logic code for a functional distributed systems algorithm using the generated boilercode. If the writing of the distributed algorithms is a too complex activity for the students' level of learning, the instructor may decide to provide the application logic code to the students. This phase helps the students in understanding the different programming constructs and the internals of the algorithm. Once the application code is ready for testing and deployment, the student can then invoke deployment specific set of model interpreters, which can then upload this application logic codes to the runtime execution platform for algorithm execution.

4 Runtime Architecture of PADS

The PADS framework utilizes cloud computing infrastructure to deploy distributed systems experiments. The cloud computing provides elastic infrastructure facility which enables on-demand access to the computing resources. Figure 3 gives an overview of the layered architecture consisting of distributed systems experimentation, user interactions, cloud deployment and experimentations, and user analysis visualizations interface components of the PADS framework. Next we describe this layered architecture and the functionalities it provides to provide a scalable PADS framework.

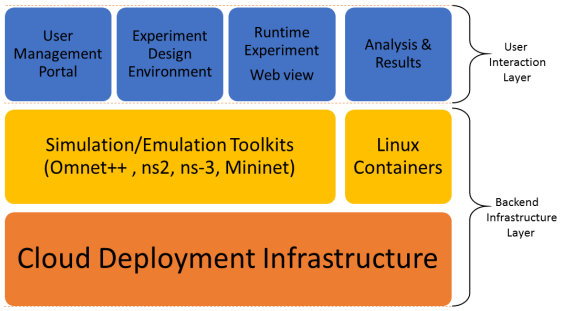

Fig. 3. Cloud based architecture of PADS framework

4.1 User Interaction Layer

The user interactions layer in the PADS framework provides following features: user/students/instructor access management and security, web front end to model and design the distributed algorithms, interaction and runtime visualization of the experiment deployed on the target simulator/emulator running on the cloud infrastructure, and performance monitoring logs of the completed experiment. User interactions primarily occur through the Internet enabled web browser. Next we will discuss each of the components of the user interaction layer.

1. User Management Portal: User management portal facilitates management of the user access privileges and login credentials to the PADS framework. User management portal lets an instructor manage login access for the students. Access rights to who can create, edit, and run experiments can be set as per the requirements of the distributed algorithm experiment for the students. If the instructor wants to give a 'read-only' access to the experiment scenario and prevent students from making any changes to the experiments, 'read-only' access can be set to the experiment project. Depending on the resource management and execution cost, the instructor may also want to set when and who can have access to the runtime infrastructure for deploying the distributed algorithms experiments. This gives the instructor a fine grain control over the PADS framework and various runtime components of the system.

2. Experiment Design Environment: This component provides a user with a web interface to design and orchestrate the distributed algorithm construction and specification. It provides various prebuilt artifacts to get quickly started with teaching and learning of the distributed systems concepts and algorithms. Some of the predefined algorithms include Client-Server model, BitTorrent, Paxos, Zookeeper, Chord, Pub-Sub. It also provides distributed systems specific actors like Server, Client, Leader, Tracker, Router, Wired and wireless interfaces, hub, switch. Users can build their own distributed systems algorithms representation using the prebuilt artifacts. Apart from designing the distributed algorithm experiment, the user can also specify the target simulation or emulation platform the user would like to deploy the experiment on. The current prebuilt target support is included for Omnet++, Mininet, ns-2. Due to the MDE approach as discussed earlier, new target platform support can be easily extended.

3. Runtime Experiment Webviewer: The runtime experiment webviewer provides a web enabled view of the simulator/emulation runtime software. The distributed algorithm that is ready to be executed is deployed on the target platform in the cloud infrastructure. The user may need to see different runtime properties offered by the simulator. To enable remote access to the runtime view of the simulator in the web browser, we use Virtual Network Computing (VNC) technology. noVNC is an HTML5-compliant VNC client that enables remote desktop control and viewer via a web browser interface. Figure 4 shows the Runtime Experiment Webviewer running an Omnet++ based distributed algorithm experiment displayed in the web browser.

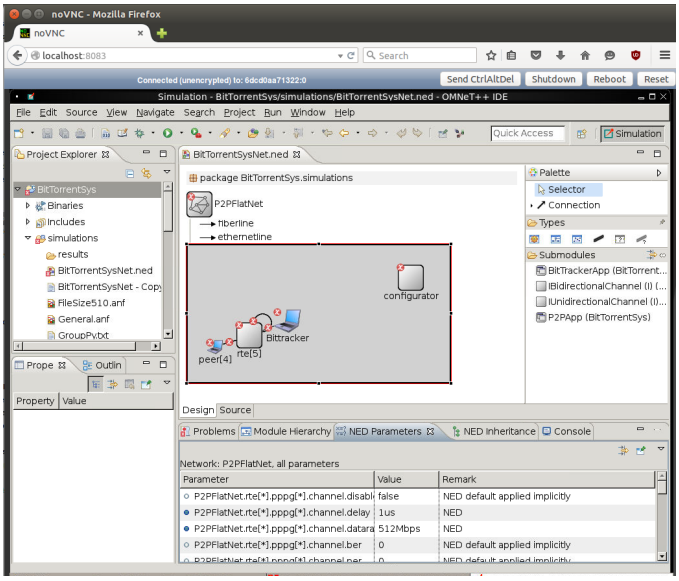

Fig. 4. Runtime Experiment Webviewer illustrating Omnet++ simulator accessed through noVNC client

4. Analysis and Results: Analysis of the experiment can give various insights into performance and effectiveness of the distributed algorithm that is constructed. One can tweak different algorithm-specific parameters to meet the user objectives for the distributed algorithms (such as optimization, best fit). After the experiments finish executing, there is a need to have a good performance and result aggregation and presentation component which the user can use to identify potential benefits or bottlenecks of the target distributed systems algorithm. The Analysis and Results components collects and makes available various experimental metrics, results and logs generated by the target execution component.

4.2 Backend Infrastructure Layer

This layer manages the deployment and runtime of the distributed algorithm experiments which are ready to be tested and run. The backend infrastructure deal mainly with providing elastic and extensible infrastructure to run the target simulators/emulators as configured by the user. To support running a large number of experiments which may consist of different simulation and emulation targets, one needs a large number of computing resources to provide satisfiable Quality of User Experience (QoE). Cloud computing

addresses these requirements and provides on-demand access to the pool of resources to carry out experiments. Moreover, the simulators/emulators that need to be run should be able to start instantaneously without considerably long startup times. We are currently using the Linux container technology to provide fast startup of simulators/emulators. Also, to manage a large pool or cluster of physical and virtual machines on which the experiments will be running, we will need resource management capability to deploy and manage this PADS infrastructure. Next we discuss each of the components of the backend infrastructure in detail.

1. Simulation/Emulation Target Toolkits: The experiments that the user orchestrates need to be executed on one of the simulators or emulators that it is designed for. The simulators are hosted in the cloud infrastructure. As such the users do not need to have any preinstalled applications (simulators/emulators) on their local development machines. Some of these targets need very high resources to execute. Running these target toolkits on the cloud relieves users from the need of owning resourceful machines to test and play with the distributed systems experiments. Simulator/emulators can be accessed through the Runtime Experiment Webviewer. Another benefit of running these tools in the cloud environment is the lower entry to barrier to play with the distributed systems algorithms as all the tools needed to run the experiment is already preinstalled and configured, and the user does not have to deal with complexities of the application installation process. OMNET++, Mininet, ns-2, ns-3 are some of the tools that are configured and ready for the users to experiment with.

2. Linux Containers: Linux containers provide a process-based virtualization that enables wrapping all the application dependencies in an isolated sandboxed environment. Linux containers provides many attractive features which are particularly suited for the PADS framework simulator/emulation deployment in the cloud environment. We are using Docker implementation to leverage Linux container technology. Some of the features provided by the Docker technology includes lightweight deployment, instant bootup time, and sandboxed environment. The simulation/emulation targets are wrapped into a Docker container that can be deployed when the experiment is ready to be executed. Docker container also contains a running VNC server which is used by the runtime experiment webviewer to control and view the simulator/emulator execution. As shown in the figure we have an Omnet++ experiment running which is accessed via the runtime experiment webviewer powered by noVNC HTML5 client software.

3. Cloud Deployment Infrastructure: The PADS framework leverages cloud computing infrastructure to execute and run the distributed systems algorithm experiments and evaluation. The simulation/emulation targets which are wrapped into Docker containers become the deployment artifacts that can be executed on the cloud computing infrastructure. To enable efficient resource utilization of the available infrastructure (physical and virtual machines) we are using a cluster management engine called Apache Mesos. Mesos provides easy abstraction of available computing environments such as physical machine hosts, virtual machine hosts into a unified resource pool. It allows fine grained resource procurement and scheduling required for the execution of simulation/emulation runtime targets. Based on the requirement of the individual targets such as CPU, RAM, number of cores and storage, the target is deployed on the computing host that can provide these resources. Mesos provides REST application protocol interfaces (API) to enable simulation/emulation runtime to be executed, deployed and managed from the user interaction layer.

5 Framework Validation

In this section we show using a preliminary user study the accrued benefits and effectiveness of PADS in terms of increasing user productivity, ease of use, usefulness in learning distributed systems algorithms, and users' interest in continuing to use modern tools for distributed systems algorithm study. In [12], we have also demonstrated how PADS can be extended to include a new algorithm and showed PADS' effectiveness in terms of the effort saved on the part of the instructor and learner in using the framework.

5.1 Preliminary User Study

To understand the effectiveness of the PADS tool in teaching distributed systems algorithms, a preliminary user study was conducted as a part of the Distributed Systems Principles (CS-6381) graduate level course offered at Vanderbilt University, USA.[2] The study was conducted in the Spring 2017 semester. The class comprised graduate-level students in Computer Science. A total of 17 students participated in the study. In the study, students were asked to work on tasks which consisted of creating networking topologies for a publish/subscribe distributed systems scenario. The Publish/Subscribe paradigm is a key part in information dissemination in distributed systems.

5.1.1 Task Assignment And Survey Questions

As part of the CS-6381 course, students are required to complete their assignments inside a Mininet network emulator and demonstrate scalable publish/subscribe using the ZeroMQ communication middleware. They are expected to showcase topic filtering, ownership strength of the publishers and history of topic samples in their assignment. Prior to our user study, students had completed this assignment wherein they were required to create application logic for a pub/sub system in Mininet environment. Thus, an assumption is made in this study that students have an understanding and hands-on experience in writing programs using the Mininet emulator and other associated technologies.

For the PADS formulation of this problem, we provide students with actors such as publisher, subscriber and broker. So the PADS framework contained these network actors in the distributed systems scenario to construct publish/-subscribe distributed systems study. Prior to using PADS, for the assignment students were required to manually create Python language code to generate the network topology and the deployment logic to place the different publishers, subscribers and brokers on the different nodes of the topology. These steps must be repeated for every new topology

2. An exemption was approved by the Institutional Review Board for this study.

under which they want to evaluate their algorithms, which are separately coded in Python per actor.

For our preliminary study, we have focused on relieving the student from the mundane, repetitive and error-prone tasks of writing code for generating the network topologies and deploying the actors. Instead, they are provided the PADS framework and asked to visually create the topologies while allowing the tool to generate the underlying code and deployment logic. Students are given two network topologies as shown in the Figures 5 and 6. An example of the code snippet generated from the PADS framework for one of the topologies is shown in Figure 7. Note that with more complex topologies, the logic for the topology generator becomes more complex and can be a cause of substantial effort spent in getting the topology right instead of spending time on learning and understanding the behavior of the algorithm being evaluated.

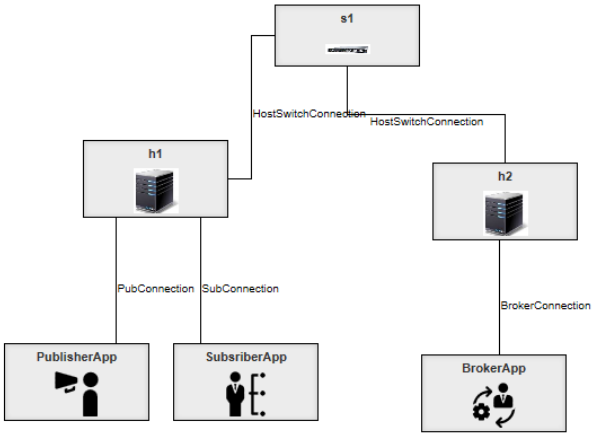

Fig. 5. Network Topology One for user studies

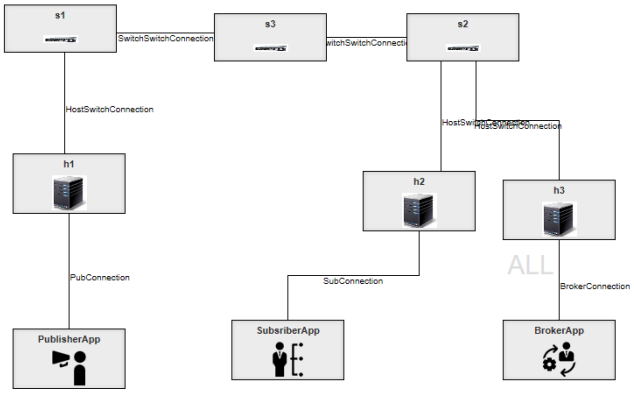

Fig. 6. Network Topology Two for user studies

To test the usefulness of the PADS tool in improving user productivity, students were first asked to manually create the network topology generator program required by Mininet. As a second step, the students were asked to

```python
#!/usr/bin/python

"""
This is a simple example that demonstrates multiple links
between nodes.
"""

from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.net import Mininet
from mininet.topo import Topo
from mininet.link import TCLink, TCIntf, Link

def runMultiLink():
    "Create and run multiple link network"
    topo = simpleMultiLinkTopo( n=2 )
    net = Mininet( topo=topo )
    net.start()
    CLI( net )
    net.stop()

class simpleMultiLinkTopo( Topo ):
    "Simple topology with multiple links"

    def __init__( self, n, **kwargs ):
        Topo.__init__( self, **kwargs )

        h1 = self.addHost('h1')
        h3 = self.addHost('h3')
        h2 = self.addHost('h2')

        s3 = self.addSwitch('s3')
        s2 = self.addSwitch('s2')
        s1 = self.addSwitch('s1')

        self.addLink(s1,s3,intf=TCIntf, params1 = { 'bw': 10 , 'delay' : '5ms' , 'loss' : 0}
        self.addLink(s3,s2,intf=TCIntf, params1 = { 'bw': 10 , 'delay' : '5ms' , 'loss' : 0}
        self.addLink(h2,s2,intf=TCIntf, params1 = { 'bw': 10 , 'delay' : '5ms' , 'loss' : 0}
        self.addLink(h3,s2,intf=TCIntf, params1 = { 'bw': 10 , 'delay' : '5ms' , 'loss' : 0}
        self.addLink(h1,s1,intf=TCIntf, params1 = { 'bw': 10 , 'delay' : '5ms' , 'loss' : 0}

if __name__ == '__main__':
    setLogLevel( 'info' )
    runMultiLink()
```

Fig. 7. Source code generated for user study network topology

use the PADS tool to create the same network topology. Students were asked to measure and report the approximate time required to complete the above tasks. At the end of the study, the users were asked to complete a survey form to report their experience in using the PADS framework.

The following questionnaire was created for the survey:

Q1: *Compare the time to complete tasks for Topology 1 and Topology 2 by completing Manually and then using PADS Framework.*

Q2: *Did PADS help you to avoid syntax errors in the topology generation process compared to writing of the topology file manually?* ***YES/NO***

Q3: *How easy was it to use PADS?: scale* ***(1-10)****, where 1 is lowest, 10 is highest.*

Q4: *How likely are you to use PADS in future assignments/experimentation?: scale* ***(1-10)****, where 1 is lowest, 10 is highest.*

Q5: *Is PADS useful in learning distributed systems algorithms?* ***YES/NO***

The above questionnaire was carefully crafted to assess approximately how the PADS framework meets the ABET's learning outcomes as discussed in Section 3.1. **Q1** tries to address the learning outcome **3.k**, which assesses whether student is able to use the PADS system to create network topology experiments efficiently and effectively. **Q2** addresses the learning outcome **3.b**, as to how students where able to use the PADS framework to design the system correctly thereby avoiding manual mistakes during the design and setting up of the experiments. **Q3** and **Q4** assesses the student's learning outcome **3.k**, as to how likely they are to use the PADS framework in future for solving distributed systems problems. **Q5** to tries to assess the learning outcomes **3.a** and

3.b, which demonstrate students' eagerness to use PADS framework in creating distributed systems algorithms and finding solutions to the associated problems.

5.1.2 Survey Results and Discussion

Based on the user study, the survey data was gathered and analyzed. Our findings are reported below.

Response to Q1→ Time to Complete: As can be seen in the box chart shown in Figure 8, comparison of time to completion for creating the topology files for the two test topologies using the manual approach and using the PADS tool is illustrated. The median value for time to completion using the manual approach is 7 and 9.5 minutes for the topologies 1 and 2, respectively. The 75th percentile value of the sample for time to completion using the manual approach resulted in 10.5 and 16.75 minutes for the topologies 1 and 2, respectively. Using the PADS approach, the median time required for completion for topologies 1 and 2 was 2 and 2 minutes, respectively. While the 75th percentile value of the sample for time to completion using the PADS approach is seen as 3.75 and 3.75, minutes respectively. Using these results, we can deduce that there is a substantial productivity gained using the PADS framework for constructing and creating the topology file for the distributed algorithms study.

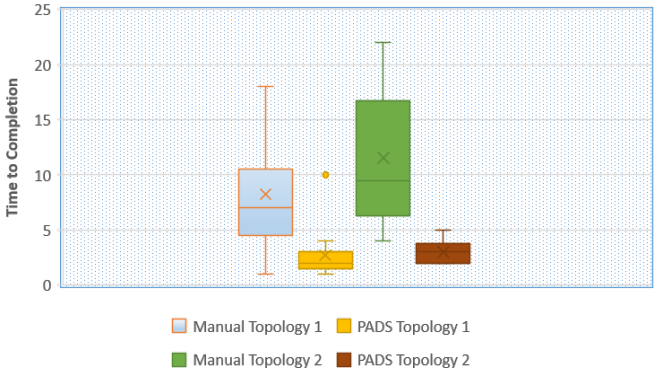

Fig. 8. Survey response to Question 1: Compare time to complete tasks for Topology 1 and Topology 2 by completing Manually and then using PADS Framework

Response to Q2→ Avoiding Manual Syntax Mistakes using PADS: As seen in Figure 9, 88 percent of the participants felt that the PADS helped them to avoid manual syntax mistakes in writing the topology description file. 12 percent of the participants did not answer the survey question. It can be seen that the PADS framework looks very appealing to the users in avoiding manual mistakes they might make while writing the topology file manually. The PADS autogenerates the topology file and takes care of making sure the right parameters are passed to the connections creation function calls supported by the underlying target platform based on the topology specification described by the user.

Response to Q3→ Ease of use: As seen in Figure 10, most of the respondents gave a very high rating for the ease of use criteria of the PADS framework. 35.71 percent of users rated 7 and 8 on the scale of 10 for ease of use.

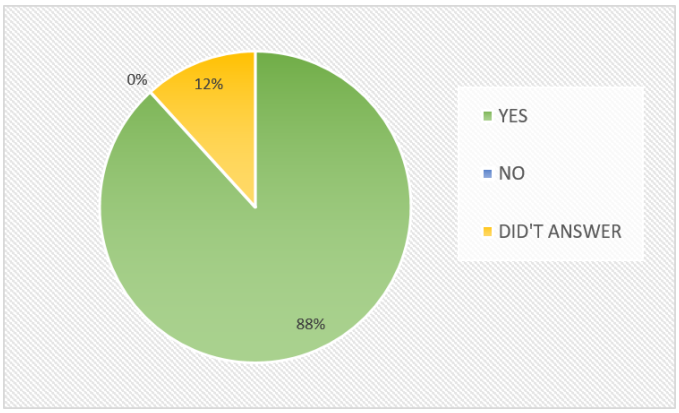

Fig. 9. Survey response to Question 2: Did PADS help you to avoid manual syntax errors compared to writing of the topology file manually?

While 21.43 percent and 7.14 percent rated it at 9 and 10 respectively on the scale of 10 for ease of use. This shows that our PADS framework is very easy to use for users to learn and play with distributed systems algorithms. The intuitive visual drag and drop environment helps the users to easily model different scenarios of the distributed systems for evaluation and study. Moreover, the support for the target emulator/simulator environment in PADS helps users to study distributed systems concepts in an environment which they are familiar with, which in this user study is Mininet.

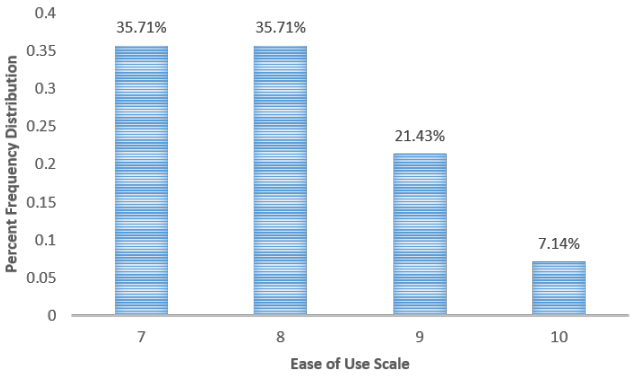

Fig. 10. Survey response to Question 3: How easy was it to use PADS?

Response to Q4→ Likely to continue using PADS tool: As seen in Figure 11, most of the respondents were excited to continue using the tool for future assignments and experimentations for learning distributed systems algorithms. 33.33 percent, 6.66 percent, 26.67 percent, 26.67 percent of the respondents rated 10, 9, 8 and 7 scores out of 10 rating respectively for likelihood in using PADS for future study purpose. While 6.67 percent of the participants rated 2 out of 10 score for using PADS in future for learning distributed systems algorithms. Overall, the trends show that students were able to experience the benefits offered by the PADS framework in the learning of the distributed systems algorithms. Due to this, the students developed great interest in working with the PADS tool for their future assignments and experiments.

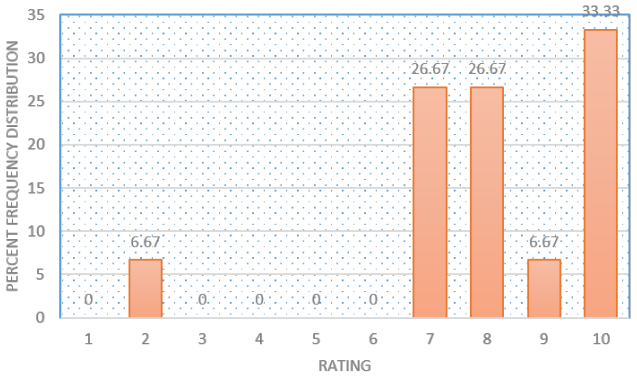

Fig. 11. Survey response to Question 4: How likely are you to use PADS in future assignments/experimentation?

Response to Q5→ Useful in Learning Distributed Systems Algorithms: As seen in Figure 12, 12 percent of the participants did not reply and 6 percent of the participants were not certain to the question if the PADS tool will be useful in the learning of the distributed systems algorithms. While, 82 percent of the participants were of the opinion that PADS tool is useful in learning distributed systems algorithms. This strengthens our belief that the PADS framework designed for learning distributed systems algorithms will be a very resourceful tool for the learners of distributed systems algorithms. More studies on more complicated scenarios are necessary in future to corroborate our claims.

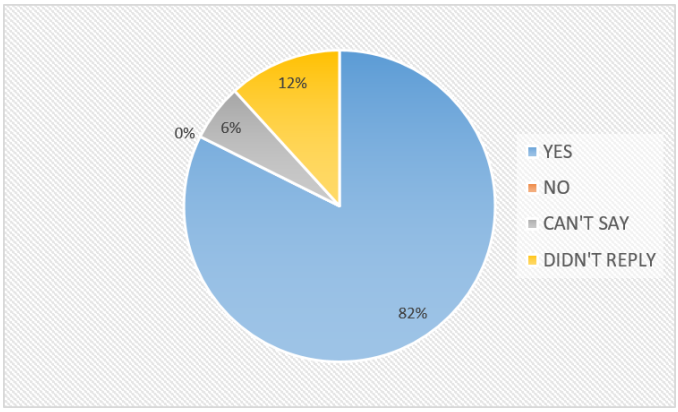

Fig. 12. Survey response to Question 5: Is PADS useful in learning distributed systems algorithms?

6 Concluding Remarks

This paper motivated the need for an integrated teaching framework used for experimenting with distributed systems algorithms. To that end, this paper describes the design, implementation and preliminary user evaluation of the *Playground of Algorithms for Distributed Systems (PADS)* framework, which provides intuitive, domain-specific modeling abstractions to capture various distributed systems algorithms, their components and requirements. Our preliminary user evaluation focused on understanding to what extent does PADS address the subset of learning outcomes we used from the ABET criteria. Our evaluation indicates that PADS prevents designers from making errors in the distributed systems algorithms test-bed setup and significantly simplifies system deployment by automating the generation of platform-specific metadata that faithfully implements the necessary execution dependency.

Ongoing and future work on PADS is focusing on framework extensibility so as to include more algorithms, reuse, building a repository of user models, conducting more user studies, and applying the framework beyond just distributed systems. PADS is available for download from:https://github.com/doc-vu/pads

Acknowledgments

This work is supported in part by NSF US Ignite CNS 1531079. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF.

References

[1] B. Hand, A. Cavagnetto, Y.-C. Chen, and S. Park, "Moving past curricula and strategies: Language and the development of adaptive pedagogy for immersive learning environments," *Research in Science Education*, vol. 46, no. 2, pp. 223–241, 2016.

[2] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[3] D. C. Schmidt, "Model-driven engineering," *COMPUTER-IEEE COMPUTER SOCIETY-*, vol. 39, no. 2, p. 25, 2006.

[4] K. Leelawong and G. Biswas, "Designing learning by teaching agents: The betty's brain system," *International Journal of Artificial Intelligence in Education*, vol. 18, no. 3, pp. 181–208, 2008.

[5] F. Kalelioğlu, "A new way of teaching programming skills to k-12 students," *Comput. Hum. Behav.*, vol. 52, no. C, pp. 200–210, Nov. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.chb.2015.05.047

[6] B. Broll, A. Lédeczi, P. Völgyesi, J. Sallai, M. Maróti, S. Wieden-Wright, A. Melo, and C. Vanags, "A Visual Programming Environment for Learning Distributed Programming," in *To Appear in the Proceedings of the 48th ACM Technical Symposium on Computing Science Education*. ACM, 2017.

[7] B. Broll, P. Völgyesi, J. Sallai, and A. Lédeczi, "NetsBlox: a Visual Language and Web-based Environment for Teaching Distributed Programming," https://netsblox.org/NetsBloxWhitePaper.pdf, 2016.

[8] A. Dukeman, F. Caglar, S. Shekhar, J. Kinnebrew, G. Biswas, D. Fisher, and A. Gokhale, "Teaching Computational Thinking Skills in C3STEM with Traffic Simulation," in *Human-Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data*, ser. Lecture Notes in Computer Science, A. Holzinger and G. Pasi, Eds. Springer Berlin Heidelberg, 2013, vol. 7947, pp. 350–357. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39146-0_33

[9] S. Shekhar, F. Caglar, A. Dukeman, L. Hou, A. Gokhale, J. Kinnebrew, G. Biswas, and D. Fisher, "An Evaluation

of a Collaborative STEM Education Framework for High and Middle School Students," in *Poster Paper at 121st ASEE Annual Conference, K-12 and Pre-Engineering Track*. Indianapolis, IN, USA: ASEE, Jun. 2014.

[10] A. Dukeman, L. Hou, S. Shekhar, F. Caglar, J. Kinnebrew, G. Biswas, A. Gokhale, and D. Fisher, "Modeling Student Program Evolution in STEM Disciplines," in *Poster paper at the 121st ASEE Annual Conference, K-12 and Pre-Engineering Track*. Indianapolis, IN, USA: ASEE, Jun. 2014.

[11] S. Basu, S. Shekhar, F. Caglar, J. Kinnebrew, G. Biswas, and A. Gokhale, "Collaborative Problem Solving Using a Cloud-based Infrastructure to Support High School STEM Education," in *ASEE Annual Conference K-12 and Pre-engineering Track*. Seattle, WA, USA: ASEE, Jun. 2015, pp. 26.359.1–26.359.21.

[12] Y. D. Barve, P. Patil, and A. Gokhale, "A cloud-based immersive learning environment for distributed systems algorithms," in *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, vol. 1. IEEE, 2016, pp. 754–763.

[13] F. J. De Hoyos Clavijo, "Learning about Distributed Systems," Master's thesis, Universitat Oberta de Catalunya, 2010.

[14] W. Abdou, N. Abdallah, and M. Mosbah, "Visidia: A java framework for designing, simulating, and visualizing distributed algorithms," in *Distributed Simulation and Real Time Applications (DS-RT), 2014 IEEE/ACM 18th International Symposium on*, Oct 2014, pp. 43–46.

[15] M. Biely, P. Delgado, Z. Milosevic, and A. Schiper, "Distal: A framework for implementing fault-tolerant distributed algorithms," in *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, June 2013, pp. 1–8.

[16] F. O'Donnell, "Simulation frameworks for the teaching and learning of distributed algorithms," *Ph.D. Thesis, University of Dublin, Trinity College. Department of Computer Science*, 2006.

[17] S. Mosser, P. Collet, and M. Blay-Fornarino, "Exploiting the internet of things to teach domain-specific languages and modeling: The arduinoml project," in *Proceedings of the MODELS Educators Symposium co-located with the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), Valencia, Spain, September 29, 2014.*, 2014, pp. 45–54.

[18] A. S. Gokhale and J. Gray, "Advancing model driven development education via collaborative research," in *Educators' Symposium*. Citeseer, 2005, p. 41.

[19] M. Jovanovic, D. Starcevic, M. Minovic, and V. Stavljanin, "Motivation and multimodal interaction in model-driven educational game design," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 41, no. 4, pp. 817–824, 2011.

[20] S. Chimalakonda and K. V. Nori, "What makes it hard to apply software product lines to educational technologies?" in *Product Line Approaches in Software Engineering (PLEASE), 2013 4th International Workshop on*. IEEE, 2013, pp. 17–20.

[21] S. Basu, G. Biswas, P. Sengupta, A. Dickes, J. S. Kinnebrew, and D. Clark, "Identifying middle school students challenges in computational thinking-based science learning," *Research and Practice in Technology Enhanced Learning*, vol. 11, no. 1, pp. 1–35, 2016.

[22] A. Repenning, D. Webb, and A. Ioannidou, "Scalable game design and the development of a checklist for getting computational thinking into public schools," in *Proceedings of the 41st ACM technical symposium on Computer science education*. ACM, 2010, pp. 265–269.

[23] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006.

[24] J. X. Zhang, L. Liu, P. Ordonez de Pablos, and J. She, "The auxiliary role of information technology in teaching: Enhancing programming course using alice," *International Journal of Engineering Education*, vol. 30, no. 3, pp. 560–565, 2014.

[25] V. Štuikys, *Smart Learning Objects for Smart Education in Computer Science: Theory, Methodology and Robot-Based Implementation*. Springer, 2015.

[26] S. Downes, "Learning objects: resources for distance education worldwide," *The International Review of Research in Open and Distributed Learning*, vol. 2, no. 1, 2001.

[27] R. P. Valderrama, L. B. Ocaña, and L. B. Sheremetov, "Development of intelligent reusable learning objects for web-based education systems," *Expert Systems with Applications*, vol. 28, no. 2, pp. 273–283, 2005.

[28] R. McGreal, *Online education using learning objects*. Psychology Press, 2004.

[29] T. K. Chiu and D. Churchill, "Design of learning objects for concept learning: Effects of multimedia learning principles and an instructional approach," *Interactive Learning Environments*, vol. 24, no. 6, pp. 1355–1370, 2016.

[30] R. Damaševičius and V. Štuikys, "On the technological aspects of generative learning object development," in *International Conference on Informatics in Secondary Schools-Evolution and Perspectives*. Springer, 2008, pp. 337–348.

[31] V. Štuikys, R. Burbaitė, K. Bespalova, and G. Ziberkas, "Model-driven processes and tools to design robot-based generative learning objects for computer science education," *Science of Computer Programming*, 2016.

[32] E. A. Commission *et al.*, "Criteria for accrediting engineering programs," *Accreditation Board for Engineering and Technology Inc*, 1999.

[33] D. Beuche, H. Papajewski, and W. Schröder-Preikschat, "Variability management with feature models," *Science of Computer Programming*, vol. 53, no. 3, pp. 333–352, 2004.

[34] J. White, J. H. Hill, J. Gray, S. Tambe, A. S. Gokhale, and D. C. Schmidt, "Improving domain-specific language reuse with software product line techniques," *Software, IEEE*, vol. 26, no. 4, pp. 47–53, 2009.

[35] G. Deng, D. C. Schmidt, A. Gokhale, J. Gray, Y. Lin, and G. Lenz, "Evolution in Model-Driven Software Product-line Architectures," in *Designing Software-Intensive Systems: Methods and Principles*, P. F. Tiako, Ed. Idea Group, 2007.

[36] The OpenStack Project. (2017) Openstack: The open source cloud operating system. [Online]. Available: http://www.openstack.org/software/

[37] F. Caglar, K. An, S. Shekhar, and A. Gokhale, "Model-driven performance estimation, deployment, and resource management for cloud-hosted services," in *Proceedings of the 2013 ACM workshop on Domain-specific modeling*. ACM, 2013, pp. 21–26.

[38] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi, "The generic modeling environment," in *Workshop on Intelligent Signal Processing, Budapest, Hungary*, vol. 17, 2001.

[39] K. An, T. Kuroda, A. Gokhale, S. Tambe, and A. Sorbini, "Model-driven generative framework for automated omg dds performance testing in the cloud," *ACM Sigplan Notices*, vol. 49, no. 3, pp. 179–182, 2014.

[40] K. An and A. Gokhale, "Model-driven performance analysis and deployment planning for real-time stream processing," in *Proceedings of the 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS13)*, 2013, pp. 21–24.

[41] M. Maróti, R. Kereskényi, T. Kecskés, P. Völgyesi, and A. Lédeczi, "Online collaborative environment for designing complex computational systems," *Procedia Computer Science*, vol. 29, pp. 2432–2441, 2014.

Yogesh Barve Yogesh Barve is currently a Ph.D. student in the Department of Electrical Engineering and Computer Science at Vanderbilt University, Nashville, TN, USA. His current research interest is in the area of model based deployment of distributed simulations, checkpointing based resilience techniques for cloud applications. Yogesh Barve obtained his B.E (Electronics and Telecommunication) from Goa University, Goa, India, 2006; MS (Electrical Engineering) from Tennessee Technological University, Tennessee, USA, 2010.

Prithviraj Patil Prithviraj Patil received a Ph.D. from Department of Electrical Engineering and Computer Science at Vanderbilt University, Nashville, TN, USA. He has also obtained M.Tech in Computer Science from IITB Mumbai, India. His research interests lies in the area of software defined networks and cloud-based distributed applications. Currently Prithviraj is working for the MathWorks, MA.

Anirban Bhattacharjee Anirban Bhattacharjee received his Bachelors degree in Computer Science and Engineering from West Bengal University of Technology, India. He is currently a Ph.D. student at Vanderbilt University, USA. His research interests are model-based designing of composite applications and their automated efficient deployment in the distributed systems and cloud environment.

Dr. Aniruddha S. Gokhale Dr. Aniruddha S. Gokhale is an Associate Professor in the Department of Electrical Engineering and Computer Science, and Senior Research Scientist at the Institute for Software Integrated Systems, both at Vanderbilt University, Nashville, TN, USA. His current research focuses on developing novel solutions to emerging challenges in edge-to-cloud computing, real-time stream processing, publish/subscribe systems, and cyber physical systems. He is also working on using cloud computing and software engineering technologies for STEM education. Dr. Gokhale obtained his B.E (Computer Engineering) from University of Pune, India, 1989; MS (Computer Science) from Arizona State University, 1992; and D.Sc (Computer Science) from Washington University in St. Louis, 1998. Prior to joining Vanderbilt, Dr. Gokhale was a member of technical staff at Lucent Bell Laboratories, NJ. Dr. Gokhale is a Senior member of both IEEE and ACM, and a member of ASEE. His research has been funded over the years by DARPA, DoD and NSF including a NSF CAREER award.