A Cloud-based Immersive Learning Environment for Distributed Systems Algorithms

Yogesh D. Barve, Prithviraj Patil and Aniruddha Gokhale
Dept of EECS, Vanderbilt University, Nashville, TN 37212, USA
Email:{yogesh.d.barve, prithviraj.p.patil, a.gokhale}@vanderbilt.edu

***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 system algorithms is needed to complement existing teaching modalities. Such an environment must be flexible to support learning of a variety of algorithms. Moreover, since many of these algorithms share several common traits with each other while differing only in some aspects, the environment should support extensibility and reuse. 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 (SPLs) and model-driven engineering and adopt the cloud platform to design an immersive learning environment called the Playground of Algorithms for Distributed Systems (PADS). The research contributions in PADS include the underlying feature model, the design of a domain-specific modeling language that supports the feature model, and the generative capabilities that maximally automate the synthesis of experiments on cloud platforms. A prototype implementation of PADS is described to showcase a distributed systems algorithm illustrating a peer to peer file transfer algorithm based on BitTorrent, which shows the benefits of rapid deployment of the distributed systems algorithm.**

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

I. Introduction

Complexities in Distributed Systems and Algorithms: As the world gets more connected owing to many advances in hardware, software and networking technologies, many new services of significant societal relevance (e.g. healthcare, transportation, avionics, weather prediction, search and rescue, education etc.) are likely to emerge. With the advent of low-cost embedded devices, sensors and other ubiquitous computing devices, such services will be inherently networked and distributed. Although these services will be designed to be easy to use, their underlying design and implementations will be substantially complex. In large-scale networked and distributed systems, many complex issues need to be addressed, such as time synchronization, fault management, replication and replica synchronization, consensus among peers, leader-election among nodes, deadlock avoidance etc. There is also a large degree of heterogeneity in the 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 types (client-server, peer-to-peer, publish-subscribe etc.), network types (Ethernet, WiFi, Satellite). These design considerations and heterogeneity make the algorithms for distributed systems very complex.

Difficulties in Teaching and Learning Distributed Algorithms: From our experience with distributed algorithms both as a student taking a course on Distributed Systems and as an instructor teaching such a course, we felt that learning and teaching distributed systems and their algorithms is a difficult task. Existing teaching modalities, tools and techniques for understanding the algorithms for distributed systems often rely on traditional approaches such as didactic lecturing, simple proof sketches on the whiteboard, and basic simulations or toy assignments. It is general practice in universities to teach distributed systems algorithms theoretically and then having students implement them in a programming language like (Java, Python or C++) or simulate them using basic simulation tools [1] . This approach incurs several difficulties for students including (1) programming them in languages they are not experts in, (2) analyzing these algorithms in simulators/emulators they are unfamiliar with, and (3) needing to deal with accidental infrastructure 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) hence do not obtain a holistic view of distributed systems and how different algorithms work together in a real world distributed system.

Solution Approach and Organization of paper: To address these challenges, we use Software Product Lines (SPLs) [2] in the context of cloud platforms to 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 while differing only in some aspects. 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.

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 the strengths of model driven engineering (MDE) [3] with generative capabilities, feature modeling and teaching/learning tools & technologies. In this context, this paper makes two contributions:

1) We present the underlying feature model that captures the commonality and variability across a collection of distributed systems algorithms, and show how this feature model is realized in a domain-specific modeling language (DSML), as well as generative capabilities that maximally automate the synthesis of product variants (i.e., experiments involving one or more distributed algorithms) that can be deployed and tested at large-scale using cloud platforms.
2) We present a prototype implementation of PADS to showcase a distributed systems algorithm illustrating a peer to peer file transfer algorithm based on BitTorrent, which shows the benefits of rapid deployment of the distributed systems algorithm. Using this example, we provide some qualitative evaluation of PADS showcasing the effort saved on the part of a student.

By no means do we claim to have solved all the challenges in this realm, however, our research is continuing to further develop the capabilities of PADS. The rest of the paper is organized as follows: Section II surveys related research and compares and contrasts them with PADS; Section III presents a motivating example and key challenges that are resolved by PADS currently; Section IV delves into the details of PADS; Section V provides a validation of PADS; and finally Section VI provides concluding remarks alluding to lessons learned and future work.

II. Related Work

In this section we compare PADS to related works along three dimensions: tools for network experimentation, current work in the teaching specific software for distributed systems algorithms, and use of model driven engineering in the design of large scale software systems in the context of educational learning systems.

1. Tools for network experimentation: The authors in [4] have provided an extensive list of experiment management tools for carrying out research in distributed systems. The strengths and weaknesses of various tools are provided. It highlights the need for good experimentation tools to ensure replicability and reproducibility of an experiment. It also points out the need for further development of these tools due to numerous challenges in fully exploiting the capacity of certain experimental testbeds. Efforts have been made to address the problem in repeatable research environments. For example, the Apt (the Adaptable Profile-driven Testbed) approach has been presented in [5]. It builds an experiment profile which describes the dependencies needed to conduct networking experiments. Dependencies include both hardware and software requirements of the experiment. Researchers can build their own testbeds and share these profiles with others, who can then repeat and reproduce the experimental environment.

The cOntrol and Management Framework (OMF) [6] provides a modular architecture for managing heterogeneous resources in networking testbeds. It manages resources such as remote bootstrapping, and saving and loading disk snapshots for conducting experiments. It also features an experiment description language which can be used to specify resource requirements and configuration and experiment orchestration. OMF-F [7] is based on OMF [6]. It addresses the shortcomings of OMF which was targeted for a single testbed deployment only. OMF-F allows management of resources for network experiments across federated networking testbeds. It provides a DSML supporting special event-based experimental scenario. It also supports management of resources using a resource model which features publish-subscribe messaging pattern for communication and control of resources. Further, it supports scalable deployment of experiments across federated testbeds.

The above frameworks are oriented more towards the research community to conduct experimental research. Unlike these tools, PADS is designed as a learning aid in teaching distributed systems algorithms. Our approach also facilitates use of both the simulator and real world testbed in a single development environment for testing different distributed systems algorithms. We also use the SPL approach in the design and development of PADS.

2. Learning systems for distributed algorithms teaching: Authors in [8] 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 [9] is a framework for designing, simulating and visualizing distributed algorithms. It is developed using JAVA 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 [10] 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. It lacks integration with simulators that would facilitate quick testing and debugging of algorithms.

LYDIAN [11] is an animation environment for visualizing the behavior of distributed algorithms. It supports writing custom protocols which can be executed on the LYDIAN's simulator and the output animation can be viewed on the TCL/TK based graphical user interface. It also supports playback of algorithm execution events using a trace file for quick demonstration of algorithm behavior on the animation windows. VADE [12] is another framework that provides visualization of distributed algorithms. The VADE framework is designed such that computation and implementation of algorithm is done on the server side while users can see the algorithm visualization via JAVA applets using the web browser on their client machines. The algorithm is implemented using JAVA programming language.

Another teaching and learning framework called FADA (Framework Animations of Distributed Algorithms) is presented in [13]. In FADA, the simulations are written using JAVA programming language using the visualization APIs provided by the framework. It also provides a set of preassembled 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.

3. MDE in learning systems: Previous work has shown MDE and DSML being effective tools [14] in developing teaching software systems. Students have also seen the benefits of rapid code generation based on MDE techniques. In [15], 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 [16]. It utilizes 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 [17] 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.

III. Dimensions of Variability in the Learning Process

Decentralization is a fundamental aspect of distributed algorithms. Distributed systems algorithms deal with a large number of issues that arise in distributed systems, e.g., challenges pertaining to successful coordination of various entities in the system, fault-tolerance, communication heterogeneity, and the distributed time synchronization. Given these challenges, distributed algorithms are difficult to comprehend and its implementations are often non-trivial. Moreover, these algorithms target different classes of problems such as consensus, synchronization, discovery, fault-tolerance, performance and correctness. To observe the behavior of a distributed algorithm in action one could use a simulation environment or real time observation on a set of actual distributed systems. Network simulators can be used to implement and test existing or new algorithms in a controlled environment at a lower expense both in terms of time and money.

This section elicits the key challenges in the learning process of distributed systems. It presents a motivating scenario based upon which a set of challenges are documented that subsequently drive our research on PADS.

A. *Motivating Scenario*

Figure 1 shows a motivating scenario illustrating a workflow that captures a typical approach to hands-on learning of distributed systems algorithms. A number of challenges manifested in this workflow and described below prompted us to investigate solutions to address these challenges.

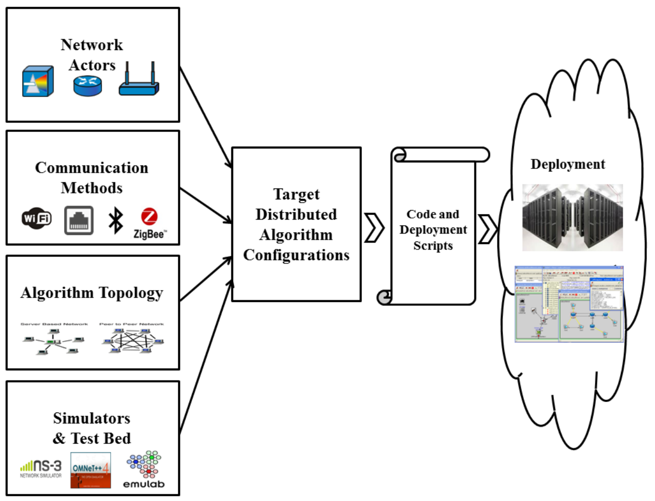


Fig. 1: Commonality and Variability in Algorithm Learning Workflow

As shown in Figure 1, teaching and understanding of an algorithm is typically accomplished by simulating the algorithm in simulators that can be parametrized in a variety of ways or by running the algorithm in testbed environments that can be configured in multiple different ways. This apparently simple teaching and learning workflow manifests a significant amount of variability both within and across the stages of the workflow. For example, many different network simulators exist such as $OMNET{++}$ and ns-3. Similarly many different network testbeds exist such as Emulab, PlanetLab and GENI. Furthermore, the algorithm demonstration involves setting up of different deployment configurations and network topology setups. The algorithm under study may involve a variety of network actors like mobile hosts, clients, servers, peers, elected leader actor, and network coordinator. The underlying network communication may involve different types of network interfaces either standalone or a combination of Bluetooth, WiFi, ZigBee, Ethernet, USB, and serial interfaces. Moreover, the target algorithm involves defining the network topology for demonstrating the experiment and understanding the algorithm details.

As an example, the process of understanding a peer to peer file transfer algorithm such as BitTorrent will involve a set of peers which communicate with each other. In this scenario, there would be different network actors like Peers and Trackers. These actors, which are specified by the algorithm, may communicate via an Ethernet interface in a LAN environment or over a wireless link. The network interface could support variable transmission speeds. For demonstrating the target algorithm, one also needs to represent the desired network topology accounting for all the associated network actors in the system thereby ensuring a faithful operation of the algorithm. Based on all the configurations chosen in the different stages of the workflow, a student must then proceed to deploy the experiment in the deployment environment, which can be either a simulator or a testbed.

B. Challenges and Requirements

As evident from the above scenario, any tool used in teaching and learning distributed systems algorithms must manage a large amount of variability across the different stages as well as support the key commonalities. The commonalities include the communication model, model of computation, and the problem being solved, e.g., consensus, fault tolerance, consistency, lookup, etc. We surmise that by capturing the variability in the demonstration of the distributed algorithm as a software product line would enable instructors to rapidly provision a desired target algorithm for teaching, which in turn will make it easier for students to learn it thereby saving time in the experiment configuration and setup. To be an effective teaching aid, however, the educational software product line must address a number of challenges that are summarized based on the above discussion of the variabilities:

Requirement 1$\rightarrow$ Extensibility and Tool Reuse: The tool should enable new algorithms to be added to the existing collection. At the same time, any associated tool such as a simulator or a testbed must be reusable in the context of newly introduced algorithms. At the same time as new simulators, experimental testbeds, and real-world scenarios emerge, the framework should be extensible to include these new back ends also.

Requirement 2$\rightarrow$ Programming and Deployment heterogeneity: The framework may be tied to one programming language but the student may not always be familiar with that programming language. One should be able to program the distributed systems algorithms in the language one is familiar with and the SPL framework should support learning of distributed algorithms by being implementation language-agnostic. The deployment of algorithms can be made on simulators, emulated testbeds or in a real world network deployment. The SPL framework must allow users to define a network topology of the target distributed algorithm and run it on the desired deployment environment thus enabling the possibility to seamlessly transition the code artifacts written for a simulator to a real-world deployment or vice versa. There should be clear separation of concerns between the definition and deployment of network topology.

Requirement 3$\rightarrow$ Integrated tool to rapidly create topology and deploy experiment: It is a challenging task to use existing tools to rapidly create a new network topology to showcase the desired distributed systems algorithm and deploy on single or multiple deployment environments all using a single toolchain. It is hard to find such tools that can be used to run experiments targeting multiple deployment frameworks in an integrated tool suite. Such a requirement must be met.

IV. Design and Implementation of PADS

We now discuss 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. We also show how PADS addresses the challenges and requirements introduced in Section III.

A. 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 [18]. Feature models provide proven techniques for improving reusability by specifying the reuse rules.

The feature model for PADS must capture the commonalities and different dimensions of variabilities that we highlighted in Section III. To that end we have defined a conceptual feature model for PADS shown in Figure 2. The `PADS` framework is represented as the root feature in the figure. The `Deployment` and the `Distributed-Algorithms` are the required features. Both these features are required because we must have an algorithm that we want to test, and it must have one of the many potential choices available for deployment so that it can be tested.

These features are used to capture from the user the types of target distributed systems algorithm to be evaluated and where to perform the deployment of such algorithm. The `Deployment` can be made on `Simulator`, `RealDeployment` or an `Emulator`. `OMNET++` and `NS-2` are types of `Simulator` feature. `Emulator` can be `Mininet` or `Qemu` which are type of emulator tools available for experimentation. `RealDeployment` can be done on cloud based virtual machines(`Cloud-VM`) or on

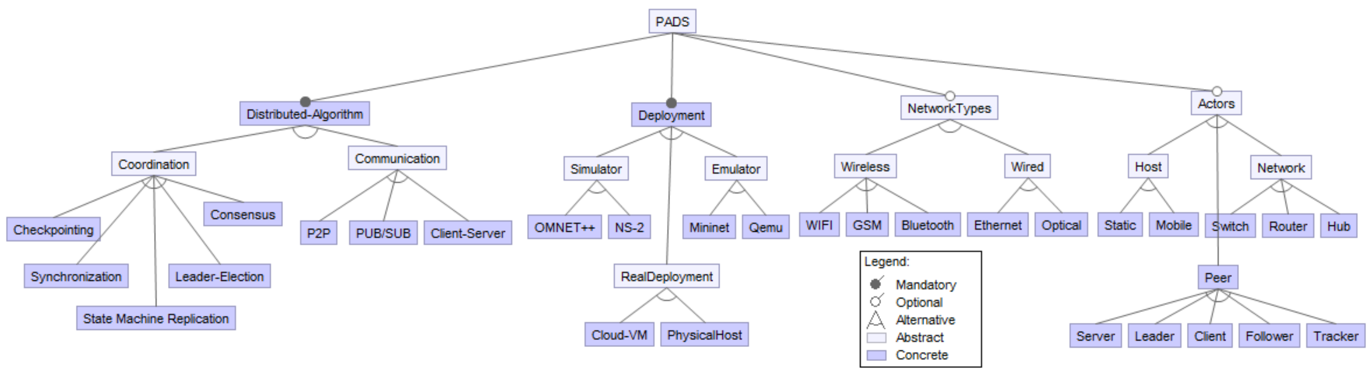


Fig. 2: Feature Model Diagram of Playground of Algorithms for Distributed Systems (PADS) Framework

actual physical host machine(`PhysicalHost`) to conduct experiments. The `Distributed-Algorithms` consists of an extensible set of distributed algorithms, which in turn can be categorized under different classes, such as `Coordination` and `Communication`. `Coordination` category of algorithms include `Checkpointing`, `Synchronization`, `State Machine Replication`, `Consensus` and `Leader-Election`. Peer to peer (`P2P`), publish-subscribe (`PUB/SUB`) and client-server (`Client-Server`) communication models are a part of `Communication` models category. The `Distributed-Algorithms` comprises one or more kinds of `Actors`. `Actors` may be an abstract representation of peer systems: like `Server`, `Client`, `Leader`, `Tracker`.

Other actors could represent an abstract notion of host system, which could either be `Mobile` or `Static` host. Some of these actors could represent network devices based on the functionality it performs like `Router`, `Hub`, `Switch`. Each of these `Actor` feature exhibits a set of network and communications features. These features are represented by the `NetworkTypes` feature. `NetworkTypes` could be `Wireless` or `Wired` communication interface. `WIFI`, `GSM`, `Bluetooth` are a type of `Wireless` communication interface. `Ethernet` and `Optical` are type of `Wired` communication interfaces.

B. 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 [19]. 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 and use generative technologies, which are key artifacts of MDE, to automate the synthesis of product variants of our PADS product line.

Our PADS framework is hosted on virtual machines deployed on Openstack cloud, which is an open source cloud computing infrastructure [20]. 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 using remote access client which could be either a web browser client or a desktop client.

Figure 3 shows the overall process of creating a network topology, selection of the distributed algorithm, intermediate topology-specific code generation, and deployment of the algorithm using our PADS framework. A student develops a model of the experiment using our DSML. The user first selects the algorithm. The algorithm selection can be done from one of the pre-assembled algorithm modules provided as part of the framework. The user then creates the network topology for the desired test algorithm. The framework has built-in constraint checker module which verifies if the test topology is supported, if any violations are present it will notify the user about the constraint invalidation. Next, to conduct the experiment the user selects the deployment environment. Once the experimental setup for a given algorithm and deployment environment are modeled, a set of reusable model interpreters are executed to automatically generate the deployment engine specific glue-code and the execution artifacts.

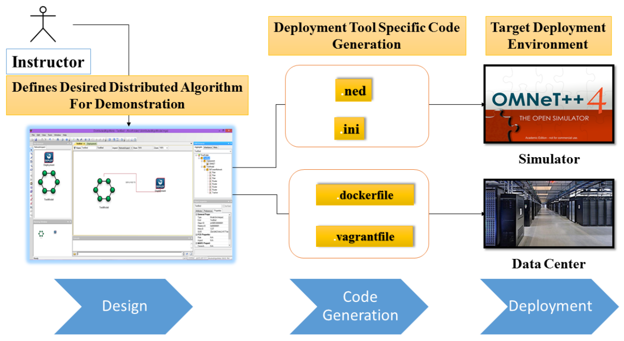

Instructor
Defines Desired Distributed Algorithm For Demonstration
Deployment Tool Specific Code Generation
Target Deployment Environment
ned
ini
dockerfile
vagrantfile
OMNeT++ 4
THE OPEN SIMULATOR
Simulator
Data Center
Design
Code Generation
Deployment

Fig. 3: Model-based Process for Distributed Algorithm Demonstration and Deployment

The Generic Modeling Environment (GME) [21] is used to develop the DSML and generative capabilities for provisioning and deployment management of the experiment. GME 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 same GME 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.

Figures 4, 5 and 6 illustrate the metamodels used in the framework. In these metamodels we have mapped the features that we described in the feature model into concrete metamodeling artifacts. The metamodel primarily comprises first class entities, such as DistributedAlgorithm, Deployment, and Main.

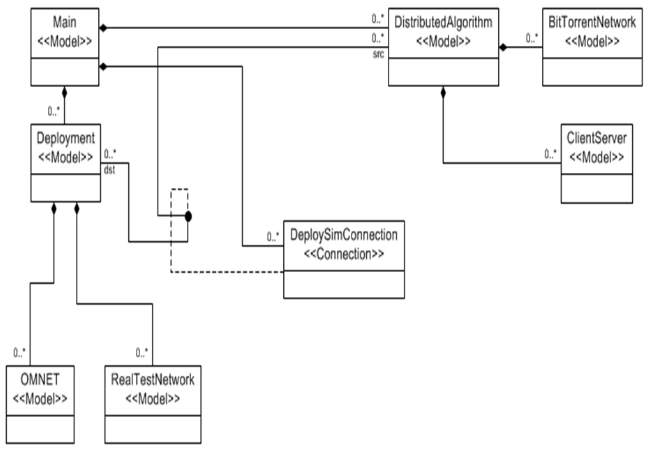


Fig. 4: Meta-Model of Playground of Algorithms for Distributed Systems (PADS) Framework

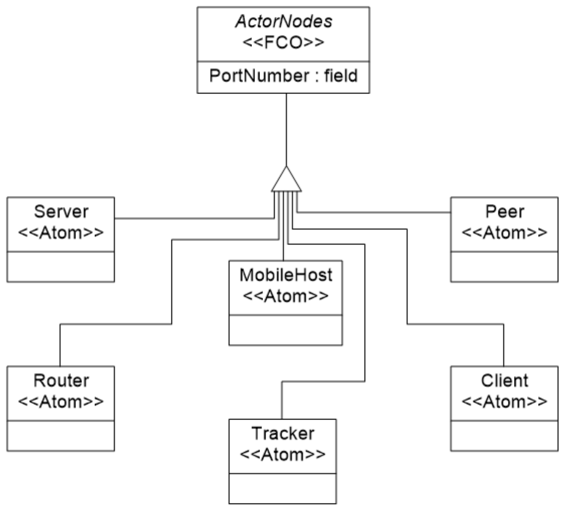


Fig. 5: Meta-Model of Network Actors

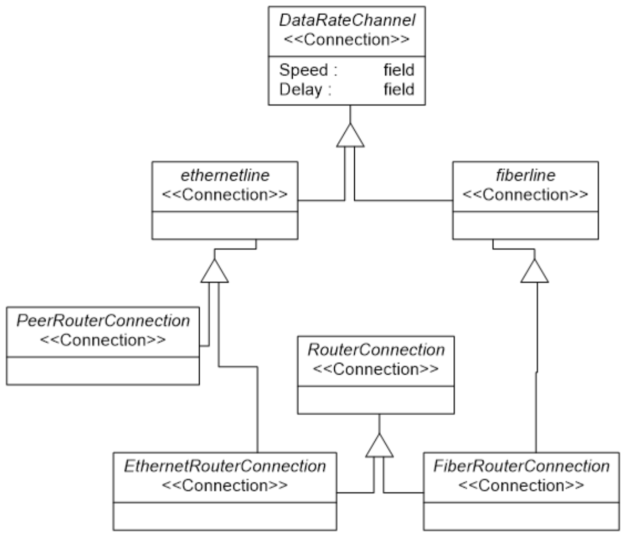


Fig. 6: Meta-Model of DataRateChannel representing the communication characteristics

Next, we describe the different metamodel components in the DSML and their responsibilities:

- *Main:* represents the main meta-model block of PADS framework. This component aims to provide information for all the framework components like: *Deployment* and *DistributedAlgorithm*, which can be configured by users depending on their experimental setup needs.
- *DistributedAlgorithm:* defines the distributed algorithms supported by the framework. Users can then select distributed algorithms contained within this model. As can be seen in the Figure 4 the *BitTorrentNetwork* and *ClientServer* algorithms are currently available with *DistributedAlgorithm*.
- *Deployment:* specifies the type of deployment environments available for testing the algorithms from *DistributedAlgorithm*. A user can create the network topology and deploy the algorithm on the target environments supported by *Deployment*. *OMNET* which is a simulation environment and *RealTestNetwork* which is a real world testbed are two such deployment environments.
- *ActorNodes:* these represent the type of network participants/nodes that will be used in the algorithm. As seen in Figure 5, these have an attribute field of *PortNumber* which represents the network port of the node. *Server*, *Router*, *Peer*, *MobileHost*, *Tracker*, *Router* are such *ActorNodes* that can be utilized in distributed systems algorithm as network actors.
- *DataRateChannel:* is used to define the network communication characteristics. Communication properties like communication rate (*Speed*) and communication delay

(*delay*) can be specified for the network. *Ethernetline* and *Fiberline* are two such communication media represented in the Figure 6.

- *DeploySimConnection:* acts as a bridge between the test algorithm and the deployment environment. Therefore the *DistributedAlgorithm* defined by the user are connected to the *Deployment* component via *DeploySimConnection.*

Our framework also leverages the GME's OCL constraint checker facility to detect design time configuration errors. If there is any violation the constraint checker reports this to the user.

In Figure 7, an example model of execution of a distributed systems algorithm using the DSML is illustrated. In the figure, DSML components such as `TestModel` (*DistributedAlgorithm*) and `Deployment` (*Deployment*) are defined. In this example model, we have selected *OMNET++* as a target deployment environment. Selection of desired distributed algorithms is done as shown in the model in Figure 8. Here we have selected a BitTorrent algorithm as an example.

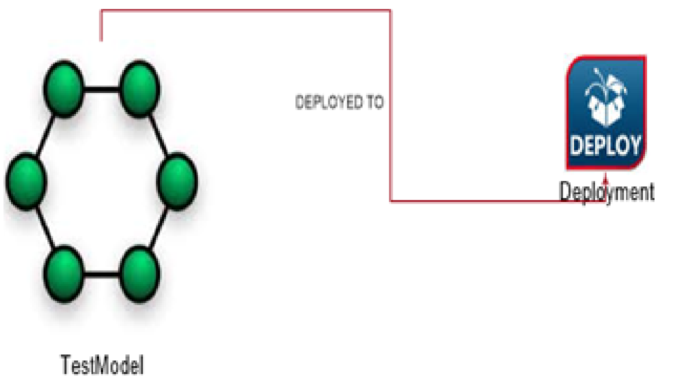


Fig. 7: Model example

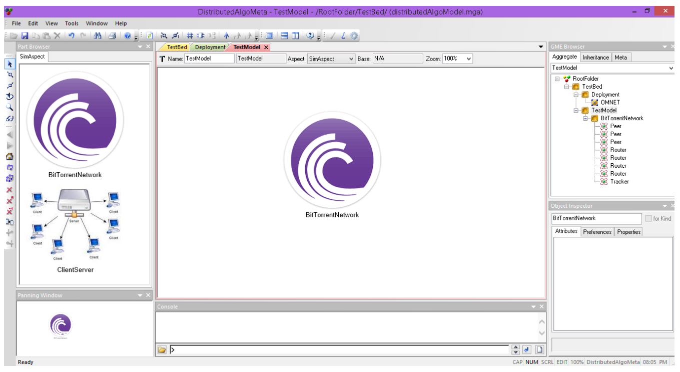


Fig. 8: Model of target algorithm selection

C. Meeting the Requirements

1) **Extensibility and Tool Reuse:** The framework supports working with new distributed systems algorithms and topologies. Using the MDE concept of metamodels, we can specify the metamodels for new distributed systems scheme. This also applies to new deployment environments which the framework may need to support in future. As shown in Figure 4, we can add new distributed systems algorithms and deployment schemes by attaching the new meta-models to *DistributedAlgorithm* and *Deployment* metamodels respectively. We also need to specify the model interpreter and how to generate the new configuration code for the new metamodels. Thus this framework can be easily extended to support new type of distributed systems and deployment environments on top of the existing software tools.
2) **Overcoming Programming and Deployment heterogeneity:** We leverage MDE technique where in we created a DSML which allows one to describe the type of distributed systems one would like to experiment with. Using the DSML the user can create a network topology of the distributed systems and deploy it on the target environment to which he is familiar with. For example, if the user is familiar with the programming construct of OMNET++ simulator, the framework will autogenerate configuration files which are compatible with OMNET++ environment. If an experiment needs to be conducted on the cloud, the tool will generate the files required to configure and deploy the experiments on virtual machines in the cloud.
3) **Integrated tool to rapidly create topology and deploy experiment:** This framework is designed to facilitate users to use a single toolchain that can create and deploy large distributed systems on different deployment environments. It also lets a user reuse the same network system topology and allows to target simultaneously to multiple deployment environments. Also the autogeneration code facility helps one to quickly configure new network topology without manually writing the network configuration code which can be both tedious and error prone.

V. Framework Validation

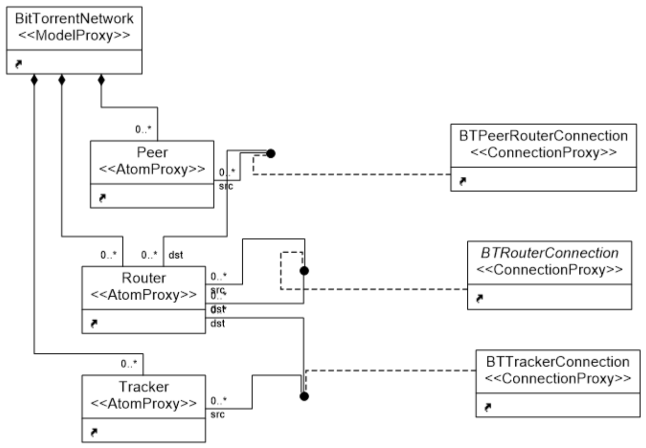
In this section we evaluate PADS along three dimensions. First, we show how PADS can be extended to include a new algorithm. Second, we show PADS' effectiveness in terms of effort saved on the part of the instructor and learner in using the framework. Note that we have not yet conducted any user studies since the framework was developed after our experience with the course. We expect to use the framework in future offerings of the course and report on the learning outcomes in a future publication. Third, based on the case study used in the first two evaluations, we elicit how the three key requirements from Section III are met by PADS.

A. Extensibility of PADS

To showcase the extensibility of our framework we have created an application based on a peer to peer (P2P) distributed system that uses the BitTorrent algorithm [22]. The BitTorrent system is a P2P distributed system that facilitates downloading of files. This is achieved by splitting a file into large number of chunks, which may be spread across a number of peers. A peer interested in downloading a specific file can download the file by downloading different chunks simultaneously from peers who have those chunks. The BitTorrent algorithm has an elaborate scheme involving a Tracker node which keeps track of peers in the system and what chunks they hold.

Thus, in the BitTorrent system, there are two kinds of network actors: a *tracker* and *peer*. As mentioned above, the *Tracker* is a centralized entity responsible for keeping track of the location of file copies in the P2P network. It also keeps track of available file chunks with the participatory peers in the network. Typically *peers* that are interested in downloading a certain file query the *tracker* for the availability of the file. Once the peer has the information required to download the file chunks from the tracker, it downloads the file chunks directly from the respective peers without the tracker acting as a broker. Moreover, peers that are interested in sharing the file register themselves with the tracker so that other peers interested in downloading that file can find them.

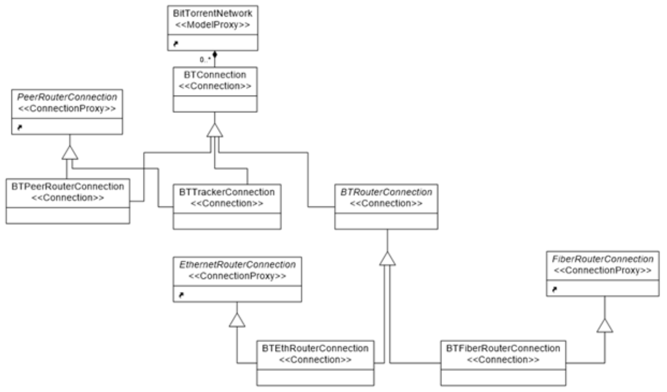
To enable the BitTorrent systems logic in our framework the instructor can first specify a meta-model as shown in Figure 9. It consists of a *peer*, *router* and *tracker* network actors in this *BitTorrentNetwork* model. The proxy elements in the figure basically refer to the references to their target models. We also see different kinds of *DataRateChannel* connections depending on the type of link shared among the network actors.



BitTorrentNetwork <<ModelProxy>>
Peer <<AtomProxy>>
Router <<AtomProxy>>
Tracker <<AtomProxy>>
BTPeerRouterConnection <<ConnectionProxy>>
BTRouterConnection <<ConnectionProxy>>
BTTrackerConnection <<ConnectionProxy>>
0..*
src
dst

Fig. 9: Meta-Model of BitTorrent Algorithm

Figure 10 provides more details about how the specific connections such as *BTPeerRouterConnection*, *BTRouterConnection* and *BTTrackerConnection* are derived from the base *DataRateChannel* model. The instructor can specify the OCL constraints for the metamodel which ensures that there is at least one *peer* and only one *tracker* in the deployment experiment as shown in the Listing 1. Once the metamodel is specified for the BitTorrent system, the instructor can make the the BitTorrent network available to students for modeling and experimentation.



BitTorrentNetwork <<ModelProxy>>
0..*
BTConnection <<Connection>>
PeerRouterConnection <<ConnectionProxy>>
BTPeerRouterConnection <<Connection>>
BTTrackerConnection <<Connection>>
BTRouterConnection <<Connection>>
EthernetRouterConnection <<ConnectionProxy>>
FiberRouterConnection <<ConnectionProxy>>
BTEthRouterConnection <<Connection>>
BTFiberRouterConnection <<Connection>>

Fig. 10: Metamodel of BitTorrent Network Connection

Listing 1: Using OCL to Detect Conflicts in BitTorrent System Modelling

```
-- onError: "Network Topology needs
    atleast one Peer and Tracker"
self.atoms("Peer") -> size>1
self.atoms("Tracker") -> size=1
```

Next we show the steps that the student will perform after the instructor has newly introduced the BitTorrent System for experimentation.

- Using the graphical modeling feature the student can choose to first draw the connection between *DistributedAlgorithm* and *Deployment* as shown earlier in Figure 7.
- From the *DistributedAlgorithm* the student can select the *BitTorrentNetwork* model as shown in Figure 8.
- Next the student can configure the experimental topology using the *Tracker*, *Peer* and *Router* from the selection window. One such possible topology is shown in Figure 11.

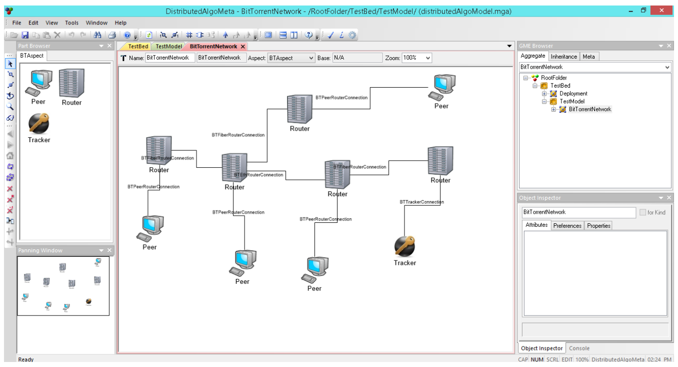
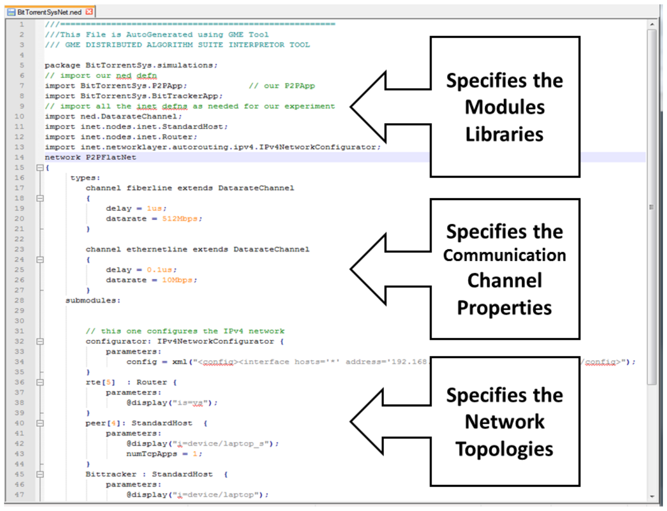


Fig. 11: Model of BitTorrent Algorithm

- The student must also specify where he/she would like to deploy the experiment. Using the *Deployment* model the student selects the target deployment.

- The student can run the OCL constraint checker to validate if there are any constraint violation in their model selection.
- Once the experimental model is specified, the GME interpreter is invoked which generates the intermediate code for the network topology which is specific to the deployment environment. Figure12 shows the screenshot of the source code section auto generated.


BitTorrentSysNet.ned
Specifies the Modules Libraries
Specifies the Communication Channel Properties
Specifies the Network Topologies

```
///=========================================================
///This File is AutoGenerated using GME Tool
/// GME DISTRIBUTED ALGORITHM SUITE INTERPRETOR TOOL

package BitTorrentSys.simulations;
// import our ned defn
import BitTorrentSys.P2PApp;             // our P2PApp
import BitTorrentSys.BitTrackerApp;
// import all the inet defns as needed for our experiment
import ned.DatarateChannel;
import inet.nodes.inet.StandardHost;
import inet.nodes.inet.Router;
import inet.networklayer.autorouting.ipv4.IPv4NetworkConfigurator;
network P2PFlatNet
{
     types:
        channel fiberline extends DatarateChannel
        {
            delay = 1us;
            datarate = 512Mbps;
        }

        channel ethernetline extends DatarateChannel
        {
            delay = 0.1us;
            datarate = 10Mbps;
        }
    submodules:


        // this one configures the IPv4 network
        configurator: IPv4NetworkConfigurator {
            parameters:
                config = xml("<config><interface hosts='*' address='192.168.                     /config>");
        }
        rte[5]  : Router {
            parameters:
                @display("is=vs");
        }
        peer[4]: StandardHost  {
            parameters:
                @display("i=device/laptop_s");
                numTcpApps = 1;
        }
        Bittracker : StandardHost  {
            parameters:
                @display("i=device/laptop");
```

Fig. 12: Screenshot of the section of code generated by GME interpreter

- Further, the GME interpreter calls the deployment runtime engine and passes the generated code and executes the experimental topology in the target environment. The student can then analyze the results and performance of the distributed systems using the tools provided by the deployment environment.

B. *Effectiveness of PADS*

In this section we describe our evaluation of Distributed Systems Playground. We focus on how the framework alleviates the error prone and tedious effort of manually writing the network topology and deployment configuration code. We then summarize how our approach addresses the challenges discussed in the section III

To evaluate how the modeling approach helps in simplifying manual configuration gluecode of the network actors used in our Distributed Systems Playground, a topology of a distributed systems similar to Figure 11 is constructed. It consists of varying number of routers and peers. For each such configuration we record the number of lines autogenerated by the GME interpreter. As can be seen from the Table I, the number of generated lines increases with an increase in the number of network actors. One can observe the effectiveness of such autogeneration feature specifically in the context of large distributed systems topology, without which one would need to configure all the connection links and endpoints manually. Manual configuration of such large system can be very tedious and error prone. The Distributed Systems Playground automatically generates all the configuration gluecode which is deployment environment specific, thereby simplifying modeling significantly

TABLE I: Increase in the number of generated code lines with increase in number of components

Router	Peers	Total Lines
5	4	85
10	103	190
20	203	299

C. *Meeting the Requirements*

Based on our BitTorrent case study from above, we now qualitatively describe how PADS meets the challenges and requirements discussed in Section III.

1) **Extensibility and Tool Reuse:** Section V-A demonstrated how an instructor can use the existing PADS framework and introduce a new algorithm. It also explains how a student can then model the system and use all existing back ends provided by PADS.
2) **Overcoming Programming and Deployment heterogeneity:** Sections V-A and V-B show how a student can model an algorithm and with a click of a button, most artifacts needed to experiment with the algorithm are generated thereby relieving the student from having to deal with any programming and deployment heterogeneity.
3) **Integrated tool to rapidly create topology and deploy experiment:** Sections V-A and V-B also illustrate how easy it is to extend the framework and rapidly create the experimental scenarios and test them in a variety of deployment scenarios.

VI. Concluding Remarks

This paper motivated the need for an integrated teaching framework used for demonstrating distributed systems algorithms. The genesis of this work stemmed from our experience as an instructor and student of a Distributed Systems course where a number of different algorithms were studied. We were interested in developing a learning aid to overcome the challenges we faced in the course. Our software engineering background helped us develop an intuition behind our solution approach. We realized that the problem space can be viewed as a software product line because of the commonalities and variabilities demonstrated by the problem space and how individual algorithms and their testing environment can be viewed as individual variants or members of a product line.

To realize these ideas, we decided to utilize a model-driven engineering approach comprising metamodeling and generative techniques. Modeling based on visual artifacts provides intuitive means to model a system using artifacts that are closer to the domain in which the system is being modeled while generative mechanisms automate many of the mundane and repetitive manual tasks. To that end, the initial prototyping of the ideas were accomplished as a class project for an MDE course. Our cloud hosted solution called the *Playground of Algorithms for Distributed Systems (PADS)* is a continuation of these original efforts.

PADS provides intuitive, domain-specific modeling abstractions to capture various distributed systems algorithms' components and requirements. The playground resolves the potential conflicts faced by student learners/researchers, such as programming these algorithms in simulators they may not be familiar with, or implementing them in specific programming languages and deploying them on real testbeds. The playground provides fundamental distributed systems building blocks that a student can use to model their system, and automate the tasks of generating simulation or real-world code, deploy these on the platforms, visualize the resulting behavior and provide feedback to the user so they can continue to iterate through this learning cycle. The playground has an extensible interface and as such has lot of capabilities for adding and supporting both various distributed algorithms and deployment plans.

We evaluated the capabilities of PADS using a representative case study of BitTorrent, which is a peer to peer file sharing distributed system. Our evaluation indicates that it 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.

Our work on PADS is by no means complete; rather it provides some evidence of developing intuitive and effective learning aids for distributed systems. Our future work will involve the following dimensions of work:

- **Extensibility:** In its current form, the suite of algorithms we currently have in our PADS framework is rather limited (e.g., BitTorrent, Chord, Paxos). We aim to improve the collection by adding several new algorithms. Moreover, currently our communication model is restricted to TCP/IP level communication. Many higher level protocols and systems hide these low-level details and instead offer other means to represent the end points. We plan to identify this variability in the problem space.
- **Reuse:** We do not aim to reinvent the wheel. Many other frameworks exist that tend to provide specialized capabilities. For example, frameworks such as Ptolemy provide effective mechanisms to model different models of computation. We will seek solutions to integrate such frameworks within PADS.
- **User studies:** Utilizing the framework in an actual course and evaluating its effectiveness in impacting the learning outcomes is a significantly important activity. We plan to undertake this activity when we offer this course again in Spring 2016 and beyond. By opening up the framework for downloads, we also hope that others would utilize its capabilities and report on its effectiveness.

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] H. Nagataki, T. Fujii, Y. Yamauchi, H. Kakugawa, and T. Masuzawa, "A kinesthetic-based collaborative learning system for distributed algorithms," in *Education Technology and Computer (ICETC), 2010 2nd International Conference on*, vol. 2. IEEE, 2010, pp. V2–97.

[2] P. Clements and L. Northrop, "Software product lines: practices and patterns," 2002.

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

[4] T. Buchert, C. Ruiz, L. Nussbaum, and O. Richard, "A survey of general-purpose experiment management tools for distributed systems," *Future Generation Computer Systems*, vol. 45, pp. 1–12, 2015.

[5] R. Ricci, G. Wong, L. Stoller, K. Webb, J. Duerig, K. Downie, and M. Hibler, "Apt: A platform for repeatable research in computer science," *SIGOPS Oper. Syst. Rev.*, vol. 49, no. 1, pp. 100–107, Jan. 2015. [Online]. Available: http://doi.acm.org/10.1145/2723872.2723885

[6] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar, "Omf: a control and management framework for networking testbeds," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 4, pp. 54–59, 2010.

[7] T. Rakotoarivelo, G. Jourjon, and M. Ott, "Designing and orchestrating reproducible experiments on federated networking testbeds," *Computer Networks*, vol. 63, pp. 173–187, 2014.

[8] F. J. De Hoyos Clavijo, "Learning about distributed systems," 2010.

[9] 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.

[10] 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.

[11] B. Koldehofe, M. Papatriantafilou, and P. Tsigas, "Lydian: An extensible educational animation environment for distributed algorithms," *J. Educ. Resour. Comput.*, vol. 6, no. 2, Jun. 2006. [Online]. Available: http://doi.acm.org/10.1145/1236201.1236202

[12] Y. Moses, Z. Polunsky, A. Tal, and L. Ulitsky, "Algorithm visualization for distributed environments," in *Information Visualization, 1998. Proceedings. IEEE Symposium on*, Oct 1998, pp. 71–78, 154.

[13] F. O'Donnell, "Simulation frameworks for the teaching and learning of distributed algorithms," 2006.

[14] S. Mosser, P. Collet, and M. Blay-Fornarino, "Exploiting the internet of things to teach domain-specific languages and modeling."

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The bittorrent p2p file-sharing system: Measurements and analysis," in *Peer-to-Peer Systems IV*. Springer, 2005, pp. 205–216.