An Exercise Assistant for Practical Networking Courses

Jens Haag[1,2], Christian Witte[1], Stefan Karsch[1], Harald Vranken[2] and Marko van Eekelen[2,3]
[1]*Cologne University of Applied Sciences, Steinmüllerallee 1, Gummersbach, Germany*
[2]*Open Universiteit, Heerlen, The Netherlands*
[3]*Radboud University Nijmegen, Nijmegen, The Netherlands*

Keywords: Virtual Lab, e-Learning, Exercise Assistant, Networking Exercises, Description Logic.

Abstract: Supporting students with feedback and guidance while they work on networking exercises can be provided in on-campus universities by human course advisors. A shortcoming however is that these advisors are not continuously available for the students, especially when students are working on exercises independently from the university, e.g. at home using a virtual environment. In order to improve this learning situation we present our concept of an exercise assistant, which is able to provide feedback and guidance to the student while they are working on exercises. This exercise assistant is also able to verify solutions based on expert knowledge modelled using description logic.

1 INTRODUCTION

Computer science curricula for students at universities nowadays include courses on networking and information technology security. Teaching theory on networking and IT security is usually done by means of textbooks and classes (either face-to-face classes or virtual classes, which are popular at universities for distance education). To anchor and deepen the acquired theoretical knowledge, a commonly used teaching method is to hand out practical exercises. The exercises can be worked out in a computer lab, which can be either a traditional on-campus lab or a virtual lab.

Recent evaluation shows that students of a traditional on-campus networking course deem it crucial for their learning success to be able to get support from a course advisor (Haag et al., 2013). While an on-campus university will be able to provide course advisors which can support students in so-called guided learning hours, this support is no longer feasible if students work e.g. at home in the evening hours using a virtual lab.

In this paper we introduce an exercise assistant for networking courses which is able to support students while they work on networking exercises. Equipped with a formal model of an exercise, the exercise assistant can be run on a student's computer whenever and wherever support is needed. The effort to author such an exercise has to be done once while instances of the exercise assistant equipped with this exercise will then be able to support any number of students.

The paper is organized as follows: First we introduce our current learning environment in chapter 2 and an example exercise in chapter 3. In chapter 4 we explain our formal model of an exercise. This formal model can be processed by our exercise assistant, whose software architecture we introduce in chapter 5. After giving a guiding example in chapter 6 we conclude our work in chapter 7.

2 VIRTUAL LAB

The virtual computer security lab (VCSL) is a stand-alone environment that each student can install on his or her local computer (Vranken and Koppelmann, 2009). It is composed of two virtualization layers, as shown in Figure 1. The host machine is the student's computer, which runs an arbitrary operating system, i.e. the host operating system. The first virtualization layer creates the virtual host machine. It consists of virtualization software such as VMware Player or Oracle VirtualBox, which runs on the host machine just like an ordinary application. Virtualization software in general introduces an additional software layer with corresponding interface, which creates a logical

Haag J., Witte C., Karsch S., Vranken H. and van Eekelen M..
An Exercise Assistant for Practical Networking Courses.
DOI: 10.5220/0004843700970104
In *Proceedings of the 6th International Conference on Computer Supported Education* (CSEDU-2014), pages 97-104
ISBN: 978-989-758-020-8

abstraction from the underlying system software and hardware (Smith and Nair, 2005). Versions of this software are available for free for a large range of platforms and therefore run on nearly all student computers, regardless of the hardware and the host operating system.

The virtual host machine runs the guest operating system. For the VCSL, Linux was selected, since it is open source and can also be distributed to students without licensing costs.

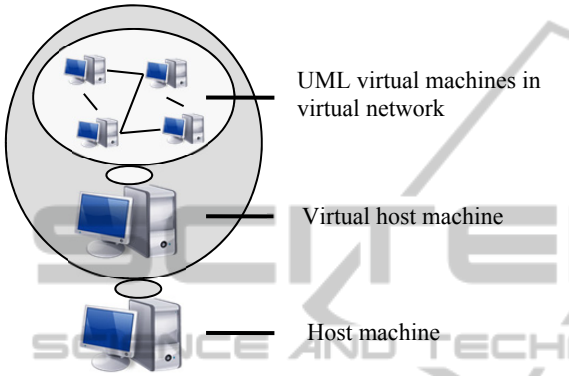

Figure 1: Architecture of the VCSL.

The second virtualization layer is a Linux application, called Netkit (Pizzonia and Rimondini, 2008), which runs inside the virtual host machine. This layer allows to instantiate multiple virtual machines that all run Linux. Netkit applies virtualization based upon User Mode Linux (UML). A UML virtual machine is created by running a Linux kernel as a user process in the virtual host machine (Dike, 2006). Multiple UML virtual machines can easily be run simultaneously, while using minimal resources. The file system is shared by all UML virtual machines using the copy-on-write (COW) mechanism. Hence, the file system is shared read-only by all UML virtual machines. Each UML virtual machine has a second, separate file system in which only the local changes to the shared file system are stored. This saves both disk space and memory, and simplifies management of multiple UML virtual machines. Restoring an initial clean system means to simply remove the second file system.

The VCSL was further developed (Vranken et al., 2011), (Haag et al., 2011) into a distributed VCSL (DVCSL). This DVCSL enables students to work together in a virtual lab by connecting their labs, even if they are physically distant from each other by using an interface to the Netkit environment. This interface consists of a Ghost Host and a Remote Bridge. While the Ghost Host was developed to extract and inject network packets when connected to an existing Netkit virtual network, the Remote Bridge is able to send and receive this packets using an intermediate connection network, e.g. the internet. Using this interface, local Netkit networks can be connected in a transparent and secure manner although they reside on different, distant students' computers.

This decentralized approach is suited to accommodate any number of students and offers students freedom to run the lab whenever and wherever they want, while preserving the properties of a conventional computer lab (e.g. the isolated network). Therefore, this approach is not limited to distance teaching but could also be useful for universities using a conventional computer lab.

3 EXAMPLE EXERCISE

An example assignment of a practical networking course to be solved using the VCSL environment is:

"Setup and configure a scenario with at least three hosts (client, router, server). Client and server should be located within different subnets. The client should be able to intercommunicate with the server by using the intermediate router. The routing should be based on static routing tables."

The minimal requirement for this setup is shown in Figure 2, consisting of at least three hosts. The client and the server have one network interface card (NIC); the router is equipped with two NICs; one for the client network named *n1* and one for the server network *n2*. Each NIC of each host has to be configured with a valid network configuration.

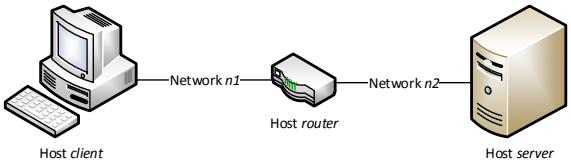

Figure 2: Valid concept draw for the example assignment.

In this example exercise, students will have to set up hosts and interconnect them accordingly within two different networks. They will then have to assign appropriate IP addresses to these hosts and ultimately configure the routing by altering the routing tables on the hosts. Once the setup is configured properly, students can demonstrate the validity of their solution, e.g. by sending network packets between client and server.

A valid and straightforward solution for this example networking assignment solved in Netkit is stated in Table 1.

Table 1: Valid solution using Netkit.

```
# Create the hosts and networks in Netkit
vstart client --eth0=n1
vstart router --eth0=n1 --eth1=n2
vstart server --eth0=n2
# Assign IP address on the client
ifconfig eth0 10.0.0.1 up
# Assign IP address on the router
ifconfig eth0 10.0.0.2 up
ifconfig eth1 11.0.0.2 up
# Assign IP address on the server
ifconfig eth0 11.0.0.1 up
# Set default gateway on the client
route add default gw 10.0.0.2
# Set default gateway on the server
route add default gw 11.0.0.2
# Connection test on client to the server
ping 11.0.0.1
```

4 EXERCISE MODELLING

In the following chapter we show how the exercises can be transferred into a formal representation, in order to be processed by a computer program. First we will show the partition of our example exercise into activities that will then be organized in a graph structure. This graph will then be extended with conditions that will make the activities verifiable. We also show a way to add feedback attributes to the graph in order to model a certain feedback strategy. Finally we introduce probing, a mechanism to improve the verifiability of activities.

4.1 Activities

Typically, exercises will start with an empty lab. Students have to perform activities that result in a working network environment, configured according to the requirements of the given exercise. While Table 1 shows the commands needed to solve the exercise in Netkit, the minimal conceptual activities needed for solving this exercise are listed in Table 2.

While A10 is the final activity, the order of the activities A1 through A9 shows only one possible sequence. The order can vary because some activities are independent from each other (e.g. A1 and A2), while some other activities have interdependencies (e.g. A1 is a precondition for A3).

These activities and their interdependencies can be modelled as an acyclic, directed graph with exactly one sink (node N with outdegree(N) = 0) and at least one source (node N with indegree(N) = 0).

Table 2: Activities needed to solve the example exercise.

Activity	ID
The client network has to be created.	A1
The server network has to be created.	A2
The client has to be connected to the client network and an appropriate IP address has to be assigned.	A3
The server has to be connected to the server network and an appropriate IP address has to be assigned.	A4
One NIC of the router has to be connected to the client network and an IP address from the client network has to be assigned.	A5
One NIC of the router has to be connected to the server network and an IP address from the server network has to be assigned.	A6
The client has to be configured to use the router's NIC in the client network as default gateway.	A7
The server has to be configured to use the router's NIC in the server network as default gateway.	A8
Routing has to be enabled on the router.	A9
Client and server must intercommunicate via the intermediate router using the IP protocol.	A10

Activities are represented by nodes. A precondition is modelled as a directed edge from the predecessor to the successor, seamlessly indicating the order of the activities. The final activity will be represented by a sink. Activities without a precondition will be represented by sources.

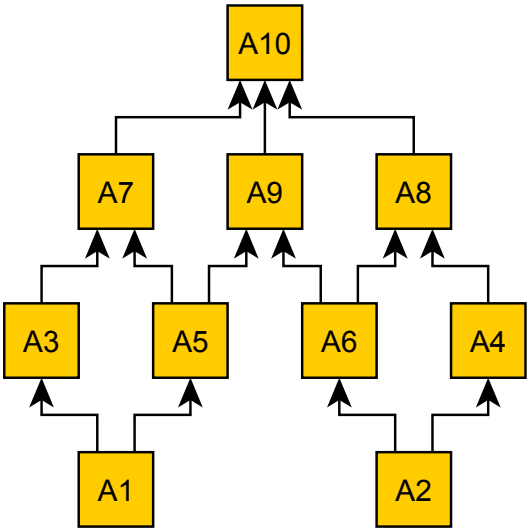

Figure 3: Example graph.

A valid graph for our example exercise is shown in Figure 3. This graph is based on the activities stated in Table 2. The interdependencies and thus possible sequences of activities show a valid example. These can of course vary, depending on the exercise and the author's intent, too.

4.2 Conditions

In order to process the graph, the activities have to be verifiable. That means that a condition is needed to detect or to decide, whether an activity is deemed passed, i.e. whether the student has successfully solved a part of the exercise.

In (Haag, Karsch, Vranken, and Van Eekelen, 2012) we showed, that network packets, obtained from the student's Netkit lab, can be used to detect and verify network properties and behaviour of an Ethernet based network. By modelling network specific expert knowledge as predicates and verifying these predicates using the captured network packets, it is possible to detect e.g. the presence of certain hosts and also routing behaviour. While the prototype in (Haag, Karsch, Vranken, and Van Eekelen, 2012) demonstrated the technical feasibility of that approach by using SQL queries to model predicates, we improved on it by using description logics (Baader, Calvanese, McGuinness, Nardi, and Patel-Schneider, 2003).

For the terminological box (TBox) we created a network ontology for Ethernet based networks, representing the network layers 2 and above (Tanenbaum, 1985), including but not limited to the header and payload fields of the most commonly used protocols, e.g. Ethernet (RFC1042), ARP (RFC826), IP (RFC791), TCP (RFC793) and UDP (RFC768). In addition, we added a unique identifier for each packet and the network origin. An excerpt of our ontology for Ethernet networks is shown in Figure 4.

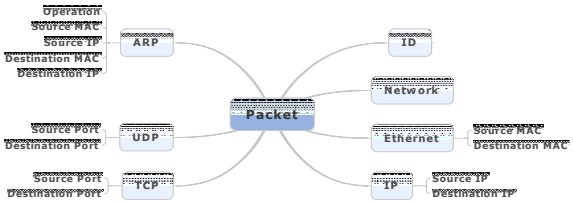

Figure 4: Ontology excerpt for Ethernet networks.

Using this ontology it is possible to model expert knowledge as predicates using a logic programming language, e.g. Prolog (Colmerauer and Roussel, 1993). For example, the expert knowledge to describe the network behaviour "routing" according to (Haag, Karsch, Vranken, and Van Eekelen, 2012) is:

"*Routing occurs if an OSI layer 3 IP transmission of a network packet between two hosts is based on more than one OSI layer 2 transmissions*".

The technical background is shown in Figure 5. The client wants to communicate with the server using the IP protocol, but the server is located in a different network segment. Direct inter-communication between client and server is not possible because the underlying Ethernet protocol does not support communication over network borders. The client has to use a known router located in the same network as itself, and thus reachable by Ethernet. The client now sends an IP packet addressed to the IP address of the server, but the underlying Ethernet packet will be addressed to the router. When the router does receive such a packet, it will forward it to the server. While the two packets that the client and the router send do not differ on the IP layer (both are sent from the client, and addressed to the server), both differ on the Ethernet layer, with different source and destination MAC addresses.

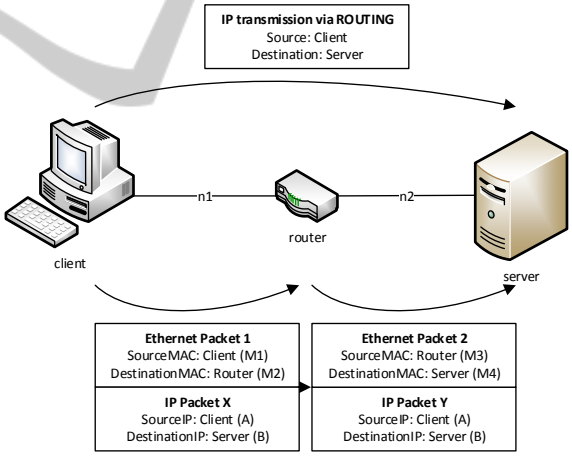

Figure 5: Routing packet flow example.

Based on the Ethernet network ontology, this behaviour can be expressed as the following Prolog predicate:

```
routing :-
   ip_packet(X,A,B),
   ip_packet(Y,A,B),
   ethernet_packet(X,M1,M2),
   ethernet_packet(Y,M3,M4),
   M1 \= M3, M2 \= M4.
```

This predicate can be read as "routing occurs, when there are two IP layer packets X and Y, both sent from IP address A to IP address B, for which

the source and destination addresses differ on the Ethernet layer."

Predicates can be used as conditions to detect activities. E.g. the predicate 'routing' can be used to verify the activity A10. We extended the graph, so that every activity can be associated with a condition to verify that activity.

Routing is only one example. We successfully created predicates describing e.g. the presence of hosts and networks, the network behaviour NAT or routing and also higher level usage. E.g. ARP spoofing behaviour can be detected if two hosts within the same subnet having different MAC addresses pretend to own the same IP address using the ARP protocol. However, this behaviour can also be caused by a misconfiguration of the hosts. For that reason this condition requires preconditions to verify a valid and error-free setup.

We also found a trade-off between the shape of an assignment and the capabilities to design predicates. If the assignment is more tightly controlled (e.g. predefined network names and IP addresses), more precise predicates can be designed to detect activities. If the assignment is more generic, the predicates also have to be designed in a more generalized manner.

4.3 Feedback

There are various types of feedback strategies which can be used to support students working on the exercise, e.g. suggestions, complete guiding or an exam mode. The specific shape will be either customized to match the author's aims or customized to the learning style of the learner or a combination. Usually recent progress the student has made in the exercise graph should trigger interaction with the student according to the feedback strategy.

Therefor we extended the graph with feedback attributes. The graph as a whole can be associated with an attribute containing the exercise description; all activities can be associated with different attributes for feedback control, i.e. text messages that give hints about what the next activity might involve (pre messages), or text messages that give feedback about detected activities (post messages). An example for activity A1 from our example exercise look like this:

```
pre_message = "You will need at least one
host connected to network 'n1'."
post_message = "Network 'n1' detected."
```

While our message mechanism provides the technical means for the implementation of various feedback strategies, the evaluation and choice of an appropriate strategy resides with the exercise author.

4.4 Probing

While the verification of activities based on passively observed network packets works for many activities, there still are limitations. One such limitation occurs, when an activity needs to be verified that does not have immediate results in the form of network packets.

An example for that would be A9 from our example exercise: the routing functionality has to be activated on the router. Students can do that by setting the appropriate kernel flag on the router if this flag is not enabled by default. This however will not result in the occurrence of observable network packets, until packets are sent to the router for being routed. A possible solution would be to ask the student to send appropriate network packets himself. We followed a different approach. For detecting certain activities we inject special predefined network packets into the Netkit environment to provoke a certain predictable behaviour. This behaviour can also be expressed as a predicate. In the routing example we inject an Ethernet packet addressed to the router into the client network that is addressed to a host in the server network (which does not have to exist) on the IP level. If routing is enabled in the router, the router will try to reach that host in the server network using ARP requests. These packets can be used to verify that routing is indeed enabled on the router.

Such a "probing" packet can be assembled by strictly following the network stack, starting with an Ethernet frame. The destination MAC address must be the router's NIC connected to network *n1*. In Netkit, the MAC address of a network interface is bound to the name of the client, resulting in a predictable MAC address for the router's first NIC eth0: 0a:ab:64:91:09:80. The source MAC address can be virtual, e.g. ee:ba:7b:99:bc:a5, followed by an IPv4 ethertype identifier (0x0800). The encapsulated IP packet starts with the version identifier (0x4), followed by mandatory header fields, e.g. length and checksum. The source IP address can be virtual but should be located within the IP range of network *n1*. The destination IP address can also be virtual but must be part of subnet *n2*. The IP packet encapsulates an ICMP echo request just to get a complete and valid network packet. This customized packet layout can be represented by a hexadecimal character array, e.g. `0aab64910980eeba7b99bca508004500001c12344000ff01549c0a0000010b0000100800f7fd00010001`.

We extended the graph, so that every activity can be associated with a custom network "probing" packet to be sent once before verifying its condition. While that actively alters the environment, it enables the verification of additional activities.

5 EXERCISE ASSISTANT

In order to support a student while working on an exercise, we developed an exercise assistant, which can be used in the VCSL. As shown in Figure 6, the exercise assistant is composed of three components: reasoning engine, feedback engine, and an interface to the student's working environment called Netkit interface.

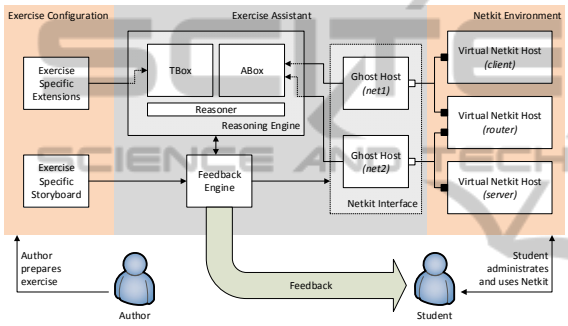

Figure 6: Architecture of the Exercise Assistant.

The reasoning engine itself is composed of a reasoner and a knowledge base, which contains a TBox ("terminology box") and an ABox ("assertion box"). The TBox contains knowledge about the domain, i.e. our ontology, in the form of predefined predicates that can be extended by the author with exercise specific extensions, while the ABox contains the concrete instantiations.

The data in the ABox is obtained through an interface to the "real world", in our case the Netkit interface. The Netkit interface consists of one or more Ghost Hosts (Vranken, Haag, Horsmann, and Karsch, 2011) that record network packets from their respective Netkit network, extract the information in them and store that information in the ABox. The Ghost Hosts can also be used to inject special network packets into the environment.

The feedback engine is the part where the activity graph will be processed. Our exercise assistant is able to read an exercise graph stored in the GraphML (Brandes, Eiglsperger, Herman, Himsolt, and Marshall, 2002) format. Once read, the activities are continuously processed according to their interdependencies, starting at the source nodes which represent activities without preconditions. Processing the activities in this case means verifying their conditions and giving the student feedback according to the feedback attributes of that activity. Once the activity is completed it will be removed from the graph and thus as a precondition for its successors. The feedback engine can also use the Netkit interface, respectively the Ghost Hosts, to insert custom network packets into the environment in order to provoke certain network behaviour to verify an activity's condition using the reasoning engine.

The Exercise Assistant is a software program written in the programming language C using SWI-Prolog (Wielemaker, 2009) as the reasoning engine.

6 EXAMPLE

Using the VCSL, the window layout of the desktop presented to the students looks like Figure 7. The exercise assistant shell is a window where the student can keep track of the feedback generated by the feedback engine. The linux shell is a window where the student is able to administrate and use Netkit in order to e.g. create hosts and networks. Once a host is started, it will open a respective shell enabling the student to administrate the host itself. Further hosts, e.g. the router and the server, will open respective shells, too.

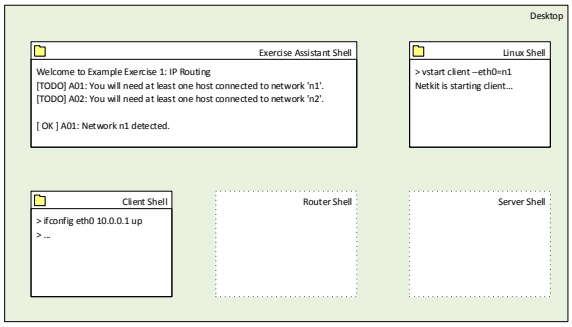

Figure 7: Desktop draft.

The following figures are screenshots taken from the exercise assistant shell guiding the example exercise. We authored the activities of table 2 according to the exercise graph of figure 3 and added verbose feedback. The introduced routing predicate is used to verify the final activity (A10). The intermediate activities too have been modelled using our ontology, partially by utilizing probing packets.

Once started, the exercise assistant introduces the exercise by displaying the exercise description.

Starting with the activities without precondition (A1 and A2), the exercise assistant will prompt the student using the respective `pre_messages`.

```
Example Exercise 1: IP Routing

Setup and configure at least three hosts (client, router, server). Client and se
rver should be located in different networks. The client should be able to inter
communicate with the server by using the intermediate router. Verify your routin
g environment by sending routed network packets between client and server.

Please name the network(s) and the host(s) according to the diagram below.

Diagram:
        +------+               +------+               +------+
        |client| <-----n1-----> |router| <-----n2-----> |server|
        +------+               +------+               +------+

IP addresses:

         client                  router
        10.0.0.1 <-----n1-----> 10.0.0.2
                                   |                    server
                                11.0.0.2 <-----n2-----> 11.0.0.1


Good luck!

[TODO] A01: You will need at least one host connected to network 'n1'.
[TODO] A02: You will need at least one host connected to network 'n2'.
```

The student can start solving the exercise according to Table 1. After the first command `vstart client --eth0=n1` is entered using the linux shell, the exercise assistant is able to confirm this valid activity.

```
[ OK ] A01: Network n1 detected.

[TODO] A02: You will need at least one host connected to network 'n2'.
[TODO] A03: Please configure the NIC of the client.
[TODO] A05: Please configure the router's NIC connected to 'n1'.
```

While A1 is being marked as verified, using the respective `post_message` of A1, the remaining independent activities without preconditions will be displayed again, superseding the preceding messages. According to the exercise graph, the student is now able to choose A2, A3 or A5 as the next activity. Starting the router connected to network *n1* and *n2* results in a verified presence of *n2*.

```
[ OK ] A02: Network n2 detected.

[TODO] A03: Please configure the NIC of the client.
[TODO] A04: Please configure the NIC of the server.
[TODO] A05: Please configure the router's NIC connected to 'n1'.
[TODO] A06: Please configure the router's NIC connected to 'n2'.
```

While the presence of the two networks is verified now, the exercise assistant is not able to detect whether the student has started the server, unless its network interface card gets assigned an IP address. Therefore the `pre_messages` are authored to prompt the student properly.

Choosing to assign the client's IP address as next activity, using the command `ifconfig eth0 10.0.0.1 up` in the client shell, will result in a verified activity A3.

```
[ OK ] A03: Client host with IP address 10.0.0.1 detected.

[TODO] A04: Please configure the NIC of the server.
[TODO] A05: Please configure the router's NIC connected to 'n1'.
[TODO] A06: Please configure the router's NIC connected to 'n2'.
```

Still missing IP addresses of router's and server's NICs, the student can proceed to configure the router's NICs.

```
[ OK ] A05: Router's NIC with IP 10.0.0.2 detected.

[TODO] A04: Please configure the NIC of the server.
[TODO] A06: Please configure the router's NIC connected to 'n2'.
[TODO] A07: Please adjust client's routing table to use the router.
```

```
[ OK ] A06: Router's NIC with IP 11.0.0.2 detected.

[TODO] A04: Please configure the NIC of the server.
[TODO] A07: Please adjust client's routing table to use the router.
[TODO] A09: Ensure, that the router is able to route packets.
[ OK ] A09: Router acts as a router between n1 and n2.

[TODO] A04: Please configure the NIC of the server.
[TODO] A07: Please adjust client's routing table to use the router.
```

Having verified that the two NICs of the router are present, the exercise assistant is able to verify A9 using a probe packet. For the simple reason that routing is enabled per default for hosts in the Netkit environment, the condition of A9 can be verified immediately.

```
[ OK ] A04: Server's NIC with IP 11.0.0.1 detected.

[TODO] A07: Please adjust client's routing table to use the router.
[TODO] A08: Please adjust server's routing table to use the router.
```

After assigning an IP address to the remaining NIC of the server, the student has to alter the routing table on the client and on the server. The exercise assistant is also able to verify these activities by using probing packets.

```
[ OK ] A07: Client uses router as gateway to 'n2'.

[TODO] A08: Please adjust server's routing table to use the router.
[ OK ] A08: Client uses router as gateway to 'n1'.

[TODO] A10: Finally, show me that client and server can intercommunicate.
```

Finally, the student is asked to demonstrate the routing functionality by sending packets between the client and the server using the intermediate router. One valid solution is to use the command *ping*.

```
[ OK ] A10: Setup verified, exercise completed.

[DEBUG] Solved in 7 minutes and 42 seconds.
[ OK ] Finished! Well done!

vmware@ubuntu-vm:~/Entwicklung/svn/current/jens/eca-pl-dev/eca-pl-dev$
```

Once the final activity is verified, the exercise assistant congratulates the student and then quits.

7 CONCLUSIONS

We presented an exercise assistant which improves the learning situation of students solving practical exercises in a networking course. Even when human

course advisors are not available, our exercise assistant can recognize learning progress and provide appropriate feedback and support. This significantly improves the learning situation for students working remotely in a virtual environment, which is common at universities for distance education. Besides this automatic support, the exercise assistant can verify intermediate and complete solutions of an exercise.

We also presented an approach to formally model exercises in a manner processable by the exercise assistant. For that purpose the exercise author can define possible activities and sequences using a graph structure. Description logic is used to define conditions for the verification of these activities. The exercise author is also able to define a feedback strategy by adding feedback attributes to the graph.

Especially for courses with many participants, our experience shows that teaching staff can benefit from utilizing the exercise assistant. While the teaching method of tutors personally and individually supporting students is certainly one of the most effective for knowledge transfer, it is not feasible for courses of sufficient size. In such scenarios, the exercise assistant can e.g. be used to offer all students a basic guided tutoring support not only wherever and whenever they want, but also at the speed that best suits their own learning style and their own abilities.

REFERENCES

Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., and Patel-Schneider, P.F. 2003. *The description logic handbook: theory, implementation, and applications*, Cambridge University Press New York, NY, USA.

Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., and Marshall, M. S. 2002, GraphML Progress Report: Structural Layer Proposal, *Proceedings of the 9th Intl. Symp. Graph Drawing (GD '01)*, LNCS 2265, pp. 501-512, Springer-Verlag.

Colmerauer, A. and Roussel, P. 1993, The birth of Prolog, *Proceedings of HOPL-II The second ACM SIGPLAN conference on History of programming languages*, ACM New York, NY, USA, pp. 37-52.

Dike, J. 2006, *User Mode Linux*, Prentice Hall, Upper Saddle River, NJ, USA.

Haag, J., Horsmann, T., Karsch, S., and Vranken, H. 2011, A distributed virtual computer security lab with central authority, *Proceedings of the CSERC '11 Computer Science Education Research Conference (Heerlen, The Netherlands, April 7 - 8, 2011),* Open Universiteit, Heerlen, pp. 89-95.

Haag, J., Karsch, S., Vranken, H., and Van Eekelen, M. 2012, A Virtual Computer Security Lab As Learning Environment For Networking and Security Courses, *Proceedings of the 3rd Annual International Conference on Computer Science Education: Innovation and Technology. CSEIT 2012, Singapore, November 19 - 20, 2012*, Global Science & Technology Forum, pp. 61-68.

Haag, J., Witte, C., Karsch, S., Vranken, H., and Van Eekelen, M. 2013, Evaluation Of Students' Learning behaviour And Success In A Practical Computer Networking Course, *Proceedings of the Second ICEEE2013 International Conference on E-Learning and E-Technologies in Education, (Lodz, Poland, Sept. 23-25, 2013)*.

Pizzonia, M. and Rimondini, M. 2008, Netkit: easy emulation of complex networks on inexpensive hardware, *Proceedings of the ICST Int. Conf. on Testbeds and Research Infrastructures for the Development of Networks & Communities*, pp. 1-10.

Smith, J. and Nair, R. 2005, *Virtual machines: versatile platforms for systems and processes*, Morgan Kaufmann, Amsterdam.

Tanenbaum, A. S. 1985, *Computer Networks*, Prentice Hall PTR Upper Saddle River, NJ, USA.

Vranken, H., Haag, J., Horsmann, T., and Karsch, S. 2011, A distributed virtual computer security lab, *Proceedings of the CSEDU '11 3rd International Conference on Computer Supported Education, Vol. 1 (Noordwijkerhout, May 6 - 8, 2011)*, SciTePress, The Netherlands, pp. 110-119.

Vranken, H. and Koppelman, H. 2009, A virtual computer security lab for distance education, *Proceedings of the EuroIMSA '09 5th IASTED European Conference on Internet and Multimedia Systems and Applications*, (Cambridge, UK, July 13-15, 2009), Acta Press, Calgary, Canada, pp. 21-27.

Wielemaker, J. 2009, *Logic programming for knowledge-intensive interactive applications*, PhD Thesis, University of Amsterdam, Netherlands.