Softw Syst Model (2012) 11:11–28
DOI 10.1007/s10270-010-0163-y

THEME SECTION

# Supporting the internet-based evaluation of research software with cloud infrastructure

**Pieter Van Gorp · Paul Grefen**

Received: 28 October 2009 / Revised: 29 March 2010 / Accepted: 12 May 2010 / Published online: 30 May 2010


**Abstract** Due to license restrictions and installation issues, it is often not feasible to experiment with software without making substantial investments. Especially in the case of legacy tools, it turns out that even free software is often too costly (i.e., time-consuming) to be installed for evaluating the quality of a research contribution. After organizing a series of events related to software modeling, we have constructed (and started to use) SHARE, a system for sharing practically any type of software artifact to reviewers and to other participants who have very limited time available. The system relies on cloud-computing technologies to provide online access to interactive environments containing all the tools, documentation, input and output models to reproduce alleged research results. The system also enables one to clone such an environment and add additional models or tools in order to extend a contribution or pinpoint a problem. In retrospect, we observe that the approach is not limited to software modeling and SHARE is in fact gaining acceptance in other fields already.

**Keywords** Reproducible research · Model transformation · Tool contest · Peer review · Cloud computing

Communicated by Tony Clark and Jorn Bettin.

P. Van Gorp (✉) · P. Grefen
School of Industrial Engineering,
Eindhoven University of Technology,
Eindhoven, The Netherlands
e-mail: p.m.e.v.gorp@tue.nl

P. Grefen
e-mail: p.w.p.j.grefen@tue.nl

## 1 Evaluating software related research: a call to arms

The amount of research contributions that rely on software is increasing. Especially when the contribution itself consists of an algorithm or information system, the results should at least be available for peer review and ideally even for reproduction by the complete research community. Section 1.1 describes how reproducibility problems in the graph transformation community have triggered our work on SHARE. Section 1.2 outlines the different levels of reproducibility that can be observed in practice, independently of that research domain. The subsequent sections introduce our key solution to the underlying problems. More specifically, they describe why we are applying cloud-computing technologies, how we are integrating them, and how others can use our supportive information system—*SHARE* (*Sharing Hosted Autonomous Research Environments* [49]).

### 1.1 Background: graph transformation tools contest

The graph transformation community organizes a yearly contest to compare the expressiveness, the usability and the performance of graph transformation tools along with a number of selected case studies. Similar to other academic initiatives, the contest leads to peer-reviewed publications. These publications facilitate the comparison of transformation languages, since different languages are applied for tackling exactly the same case studies. A scientific publication is often however not the optimal medium to describe pragmatic tool-related features. Moreover, readers (including reviewers) cannot verify (without unrealistic effort) whether all aspects of the transformation definition are documented fairly in the article.

In a special issue on experimental software engineering, van den Brand observes that "the pressure to publish is much

*higher than the pressure to develop good software*" [45]. This confirms that also outside the graph transformation community, the limitations and risks of the mainstream scientific reviewing process are evident. Coming back to the transformation tool contest, these risks were exposed most in the first edition of the event [39]. In fact, the way in which a solution was made available was not formally regulated yet. Most participants however did provide a hyperlink to the download site of the proposed tool, along with a compressed archive containing the solution to the case study. At the time of writing this article (2 years after the first contest), the disadvantages of this approach are clear: it is already impossible to reproduce the submissions to a particular case study [55] within one man-week. Consequently, only very few experts are aware of the fact that some accepted articles to the first tool contest are based on solutions that have disadvantages and limitations that slipped the peer-reviewing process.

For the second edition of the contest, the organizers aimed to anticipate reproducibility problems by giving participants access to virtual machines (VMs [17]) running on a local server during the workshop. It turned out that without specialized management software, this leads to hardware resource problems and software license problems. Additionally, it turned out that too much effort was required from the organizers as well as from participants when applying this approach over the Internet, for example when editing a journal [52].

This article describes SHARE [49], a system that has been constructed for solving such organizational problems. We will demonstrate how it has been used successfully for the third edition of the contest (and other events). Moreover, we call the organizers of other software-related publication targets (workshops, conferences and journals) to action: we have experienced that with marginal extra effort from organizers, authors and reviewers, SHARE can drastically improve the outcome of the peer-reviewing process. Therefore, we advocate that this way of working should be adopted globally, as soon as possible.

### 1.2 Demonstrating software: levels of accessibility

Before discussing the proposed *cloud*-based solution to software demonstration in the upcoming section, we should clarify what makes the evaluation of research software so expensive today. Therefore, Table 1 briefly classifies the common ways in which the software related to research articles is currently made available. Each approach can be ranked according to the level of accessibility of the proposed contribution (a tool and the related documentation, inputs and expected outputs, analysis results, etc.) Obviously, the effort for the author who wishes to demonstrate his contribution also needs to be taken into account.

At level 1, a research article is not accompanied by the artifacts for reproducing the discussed results. To researchers in other disciplines of experimental science, it may come as a surprise to hear that in software engineering it is very common to publish articles whose results cannot be reproduced. Van den Brand confirms that, "in physics or chemistry papers about experiments contain a lot of technical details in order to facilitate other researchers to replay the experiments in order to validate the results described in these papers", and advocates that in software engineering, more attention should be paid to reviewing the software artifacts that support research contributions [45].

**Table 1** Levels of accessibility for software related contributions

| | |
|---|---|
| 1 | Not accessible |
| 2 | Accessible after request |
| 3 | Available Online, Manual installation |
| 4 | Available Online, Manual configuration |
| 5 | Available Online, Fully configured |

At level 2, readers are invited to contact the authors for getting access to the artifacts supporting a research article. Among other disadvantages, this prevents an anonymous peer-reviewing process.

From level 3 and up, the research artifacts are publicly available. Van den Brand indicates that "more and more computer scientists use the open source community to distribute their tools. In this way it is not necessary to reimplement tools, only to download and install them." [45]. However, one cannot expect that *all* software-related publications strictly rely on open source licenses. It should be possible to demonstrate closed source contributions too, for a limited amount of time, in an environment that can be trusted by license holders. In fact, often articles are published before the underlying software is ready for distribution under *any* license (see [6] for an example from the model transformation domain). SHARE should easily enable reviewers to verify the results from such articles, without requiring so much effort from authors that they decide to publish elsewhere.

Apart from these license issues, one should note that contributions at accessibility level 3 are often not investigated by readers when they involve more than one sign-up, more than one download and/or a fragmented installation procedure. In the graph transformation domain, *GReAT* [1] for example requires users to register for and install different parts of a tool chain manually. Even tools based on the Eclipse integration platform often require readers to install plugins from various sources (see [8] for an example from the model transformation domain). In this context, uncontrolled updates from third-party components often lead to inconsistencies some years after the related article was published (see [21] for an example from the graph-based software engineering domain).

At level 4, the installation of a modeling and transformation environment is fully automatic (or the tool runs from a browser, as presented in [14]). A final burden for reproducing case study results remains when the input or expected output files are missing (these artifacts can be at level 1 or 2).

At level 5, all tools and all case study specific artifacts (a small manual, a set of inputs, outputs, …) are available in an environment that is dedicated to the research article. SHARE is a platform for setting up such environments and sharing them in a controlled manner. Notice that legacy libraries and prototypical software can be installed in such an environment easily and permanently too, which is of special interest when the supporting researcher has no time to make the software available at level 4.

The remainder of this article is structured as follows: the following section illustrates how SHARE enables a more reliable peer-reviewing process. It aims to activate the research community towards higher levels of availability for evaluating research software. Section 2 describes the requirements for that system and the models that have lead to its implementation. Section 3 presents simple walk-throughs to enable interested readers to instantly adopt the proposed approach too. Section 4 then presents more detailed models of SHARE: a conceptual data model clarifies the underlying data structure whereas a simulation model provides a basis for statistically determining how much time one can save by adopting the proposed approach. Section 4.3 presents our lessons learned from using and maintaining the SHARE infrastructure. Finally, Sect. 5 discusses related work and the final section presents our conclusions.

## 2 An academic cloud for evaluating research software

"Cloud computing" is an emerging paradigm for the provision of computing infrastructure [54]. Buyya et al. point to various industrial sources to motivate the economical impact of the paradigm and derive the following running definition: "A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualised computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers." [10]. SHARE is a *public* cloud (cost aspect) that enables *academic researchers* (providers and consumers) to demonstrate and evaluate *research software* (the service). At the time of writing, SHARE provides 70 users dynamic access to more than 50[1] *virtual machine images* (unified computing resources), which are currently deployed across *three university networks* (distributed system). Existing images can be cloned, modified and published, in order to add new research software to the cloud. A demonstrator can *limit the time that an image can be instantiated* as a virtual machine (the service-level agreement).

In a nutshell, SHARE is an online workflow system to:

- request access to a group of virtual machine images,
- start a virtual machine for evaluation purposes,
- create a virtual machine image containing your research software and related documentation and publish it to a group.

Additionally, SHARE enables organizers to create new groups, manage user registrations and perform other administrative tasks. Finally, server administrators can monitor the usage of servers and replicate (or migrate) images across servers. This section describes the requirements and an architectural model that have lead to the implementation of this system.

### 2.1 Requirements models

Figure 1 displays a use case diagram that models the main functional requirements of the system. Use cases related to virtual machines are shown on the left of the diagram, use cases related to groups (i.e., image access rights) are shown in the middle, and use cases related to computational resources are shown on the right.

Quite intentionally, users who wish to access a research demonstration are modeled in the very center of that diagram: the *Evaluator* actor and the associated *"Evaluate Demo"* use case indicate that the system's core requirement is to facilitate the evaluation of demonstrations. Remark that the *Evaluator* actor is also connected to the *"Register for Group"* use case. This models the requirement that demonstrations should only be accessible to users who are members of a particular group. As indicated by the association to actor *Organizer*, the corresponding sign-up requests need to be approved by a privileged user. In practice, this user is often the organizer of a workshop or a conference. The actor *Organizer* is also connected to the use case *Advertise*: the system should make it easy for organizers to include a list of demonstrations in printed proceedings and/or on a website. Notice that the number of organizers should grow linearly with the number of groups to avoid organizational bottlenecks.

The use case *"Manage Hardware Load"* relates to the actor "*Host Admin*" only. This conforms with the definition of cloud computing as "the user-friendly version of grid computing", in which case grid computing is defined as "reduce computing costs and increase flexibility and reliability by using third-party operated hardware" [54]. Concretely, the "*Manage Hardware Load*" use case refers to the

[1] This number excludes research demos created by an author of this article.

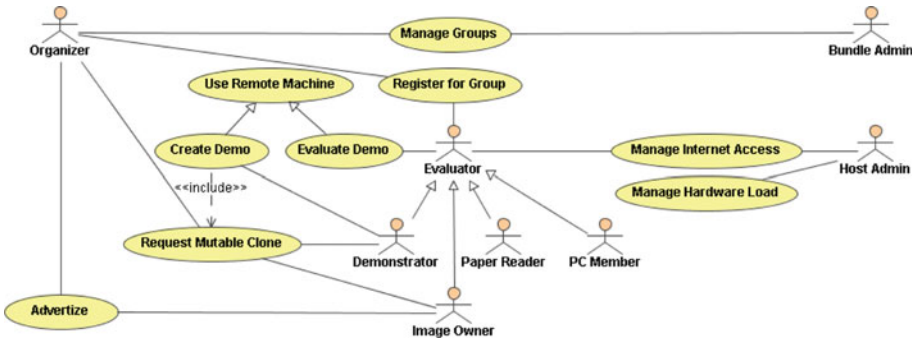



**Fig. 1** Use case diagram

browser-based inspection of server usage and the browser-based migration and/or replication of images. As soon as an image is replicated, the system should take care of load balancing automatically. Neither evaluators, nor demonstrators should care about the physical location of virtual machine images or the capacity of particular hardware.

Note once more that the demonstration of licensed software should be supported too. This has an impact on the use cases "*Request Mutable Clone*" (since it should be possible to indicate the maximal allowed session time), "*Use Remote Machine*" (since some sessions need to be terminated automatically), and "*Manage Internet Access*" (since the system should prevent that licensed artifacts leave the cloud).

The use case "Work Offline" refers to the optional support for creating virtual machine images locally (i.e., on the physical machine of a demonstrator). At the time of writing, only three out of all images have been created locally. Therefore, we acknowledge that this is a separate use case, but do not attach high priority to automating the related workflow.

In contrast, the use case "*Request Mutable Clone*" represents a feature that is of uttermost importance in practice. It involves the ability to request a clone of an existing virtual machine image in order to:

- create a new version of the original contribution,
- pinpoint a limitation of an allegedly complete solution, or
- create a completely new demo that happens to rely on some software that happens to be installed in the source image already.

Notice that each demonstrator should be able to protect the artifacts that he uploads to a virtual machine image. In some cases, he may object to the publication of variants on his submitted contribution. Therefore, before cloning an image, the owner of that image is asked for approval explicitly. Notice that after cloning, the image may be enriched with artifacts for which the new demonstrator owns the rights. Therefore, he should later be contacted for approving (or denying) requests to clone the enriched clone. In order to keep this approval process transparent, we decided to keep it simple. More specifically, we assume that each approval of a clone request also involves a transfer of ownership (and responsibility) of the cloned content. Consequently, when someone else later requests for a clone of the cloned image, the original image owner is not asked for approval anymore. So far, this restriction has not been an issue yet, but it may become an issue when using the system to control the sharing of licensed content on a global scale.

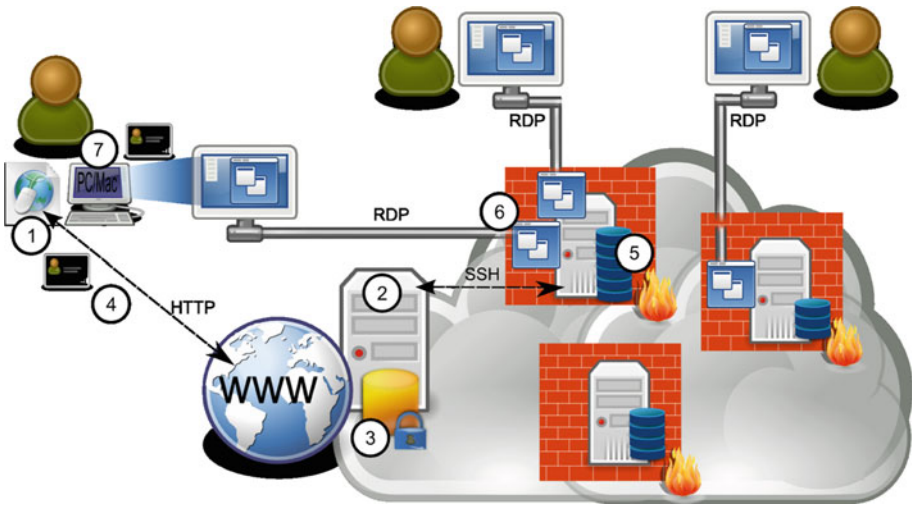
According to these requirements, even licensed software can already be shared safely (1) because the software license is securely contained in an internet connectionless virtual machine and (2) because no new images can be cloned from existing ones unless the image owner approves. However, license holders should be aware that they fully transfer the approval rights to another user as soon as they approve a clone request. The organizer should preserve his veto right though. In that setting, owners of images with licensed content should only approve a clone request if they have a good trust relationship with this organizer. In our ongoing work, we are investigating how this trust relationship can be supported by electronic contracts that are based on a new license type [4].

Besides these functional requirements, the system was implemented with the following non-functional requirements (NFRs [31]) in mind:

*$NFR_1$: Stress-Testable Environments* The system should enable the collection of performance metrics from a set of software solutions. This can either be supported by enabling group organizers to download all images for local stress-testing. Alternatively, the system can ensure that at the time of stress-testing an image, the underlying machine can be selected. That physical machine should then not be serving any other session.

*$NFR_2$: WAN-Deployable* As an academic research cloud, SHARE can rely on academic infrastructure, that is largely underused in most cases [13]. By asking such a small investment from new organizers over time, the system can scale up organically. From an implementation perspective, this approach does result in a system that runs across different university networks. This has

**Fig. 2** Architectural model




an impact on security-related as well as performance-related design decisions.

*$NFR_3$: Easily Installable* SHARE is used by computer science researchers who are not trained as system administrators and who often cannot wait for the assistance of one. Such researchers should be able to install the virtual machine server software on a mainstream university machine within one hour. Ideally, the installation does not require root privileges.

*$NFR_4$: Secure* As a specific concern, unknown university system administrators have root privileges on the machines running the virtual machine server software. Consequently, passwords should, for example, not be stored on the system's virtual machine servers. Additionally, SHARE should communicate via robust communication protocols, for which the university network administrators are willing to open up the required firewall ports.

*$NFR_5$: Free* Last but not least, SHARE can only rely on software infrastructure that can be used free of charge for academic purposes.

These NFRs are conflicting in various ways and balanced trade-offs need to be made. For example, an easy to install approach (satisfying $NFR_3$) for realizing image replication would be the use of certificates to grant two-way file synchronization between all virtual machine servers. Unfortunately, that is in conflict with the security requirement ($NFR_4$.)

## 2.2 Architectural model

Figure 2 shows an enterprise architectural model of SHARE. Following Winter and Fischer's terminology [59], the diagram aggregates elements from the *infrastructure* layer (machine and network elements) and the *software* layer. It focuses on the flow of information, the communication protocols and the security measures. The numbers in Fig. 2 represent a typical scenario through the system: first (see ① on the figure), a user authenticates to the SHARE website. The web server (labeled with ② in Fig. 2) uses information from the database (labeled with ③ on Fig. 2) to decide which demos a particular user can start. The arrow with label ④ represents an interaction between the user and the web server to initiate a new virtual machine session. Label ⑤ represents the resulting activity on a virtual machine server that (1) contains the selected image and that (2) has a sufficient amount of processing and memory power available at the selected time interval: the server boots a virtual machine with the selected image and makes it remotely available to the user (label ⑥). Notice that Fig. 2 correctly visualizes that (1) not all virtual machine servers contain all images and (2) on some virtual machine servers, there may be no active users. Notice that the latter property enables SHARE administrators to keep some servers shut down outside peak hours (or days), in order to save energy or perform maintenance tasks [28].

Section 2 defines SHARE as an academic *cloud*. Therefore, Fig. 2 symbolically displays all SHARE servers on top of a cloud-shaped graphic. Within that cloud shape, one can identify two types of servers: a web server and a set of virtual machine servers (the three blocks shown at the right). The web server hosts all dynamic web pages for starting, cloning and organizing virtual machines. Currently, the web server also hosts the database and there is no need (yet) to move it to a separate server.

The virtual machine servers are part of different university networks. Therefore, they are protected by corporate firewalls. The web server communicates with these machines via a secure protocol (SSH [5]). This ensures that when a new session is initiated across the internet, no other internet users can intercept information that could be used to login to a virtual machine on behalf of the initiating user. As additional security measures, (1) the port for connecting remotely to a virtual machine (see Fig. 2, between labels ⑥ and ⑦) is

assigned randomly for each session, and (2) users need to enter credentials at connection time (see Fig. 2, label ⑦). The underlying protocol (Remote Desktop Protocol, or *RDP*) supports modern encryption methods [44].

From a platform perspective, SHARE relies on Apache as a web container, MySQL as a database engine, and VirtualBox as a hypervisor [17]. The web server and the virtual machine servers run Linux natively. The website is implemented in PHP [29]. The communication between web and virtual machine servers is realized via Bash cron jobs [33]. On virtual machine servers, the hypervisor API is encapsulated by Bash wrapper scripts. Thanks to this indirection layer, it is conceptually possible to switch to another hypervisor without affecting any code on the web server. However, since VirtualBox currently satisfies almost[2] all requirements there is no plan to do so. Interestingly, VirtualBox can even use virtual machine images from other hypervisor tools such as VMware [56]. We are successfully using this feature for hosting a legacy demonstration that has been created offline.

The following section provides descriptions of the workflows that have been deployed so far. Next, Sect. 4 presents more models of the system.

## 3 Implementation: what?

This section walks the reader through some concrete SHARE usage scenarios. Section 3.1 shows how to evaluate someone else's demo whereas Sect. 3.2 shows how to create new demos. Sections 3.3 and 3.4 describe administrative features.

### 3.1 Evaluator walk-through

Figure 3 shows an outline of the concrete screens that a typical evaluator will be confronted with when using the system for the first time. Figure 3, ① shows the sign-up screen for a particular group (*GraBaTs09* in this case). The user can open the dropdown menu to select from all other groups known to SHARE at that point in time. In most cases, new users will have entered this screen by following a hyperlink on a workshop website and the dropdown menu will point to the intended group already.

Figure 3, ② shows SHARE's main screen. The user can access the menu in the top to start new sessions, request access to other groups, etc.[3] The screen also lists all details of sessions that are currently running for that user. Mind that the concrete server address and port can vary for different sessions for the same machine image, since images can be replicated across multiple servers.

Figure 3, ③ shows the built-in Microsoft Windows client for connecting to remote machines via the RDP protocol. As shown in the figure, the user is supported to copy/paste all relevant connection information from the SHARE website (Fig. 3, ②: username, password, server and port) to the corresponding fields of the RDP client. Notice that any operating system comes with an RDP client nowadays. Nevertheless, we have also considered the use of a Java applet that can be started directly from the screen shown in Fig. 3, ②.

Figure 3, ④ shows a screen shot after connecting to three remote virtual machines using the procedure described above. The three concrete machines shown in the screenshot relate to the 2008 edition of the graph-based tools workshop (*GraBaTs08*). Without installing any graph transformation tool locally and without fetching the specific inputs or executing the proper startup commands, users can thus compare the rather complex software environments.

Figure 3, ⑤ displays the screen that is shown after clicking "Evaluate" and "Request New Session" on the screen from Fig. 3, ②. The figure shows two variability points: first of all, users can choose between all images of the groups they are currently registered to. Secondly, users can indicate when the virtual machine should be running. By default, the interface is configured to start a machine immediately and keep it active for at least 4 h. In the case that all servers are already executing their maximal amount of virtual machines, a user may reserve a time slot in the future. Also notice that virtual machine images containing licensed software can be assigned a maximal execution time by their owner. By default, virtual machines will only be terminated when (1) the evaluator terminates his session explicitly, or (2) the agreed time has expired and a server slot needs to be freed. In contrast, when an image has been assigned a maximal execution time, all its virtual machines will be terminated directly after expiry of that time frame.

### 3.2 Demonstrator walk-through

By default, the state of a virtual machine is not persistent across sessions: for the evaluator walk-through discussed in the previous session, it would be quite undesirable that any changes to the virtual machines (as shown in Fig. 3, ④) would be persisted. All evaluators want to start from the image that was prepared by a demonstrator. Evaluators can safely investigate the effect of drastic changes to that environment since all changes are flushed after each session. Obviously, demonstrators do require stateful access.

Figure 4 visualizes how SHARE supports the creation of new demonstration images from existing ones. The following steps are based on a realistic scenario from the graph-based tools contest that was mentioned above and they correspond

[2] $NFR_3$ is only partly satisfied since installing VirtualBox requires root privileges.

[3] The figure also shows the menu entries "Organize" and "Administer", which are available only to particular users.

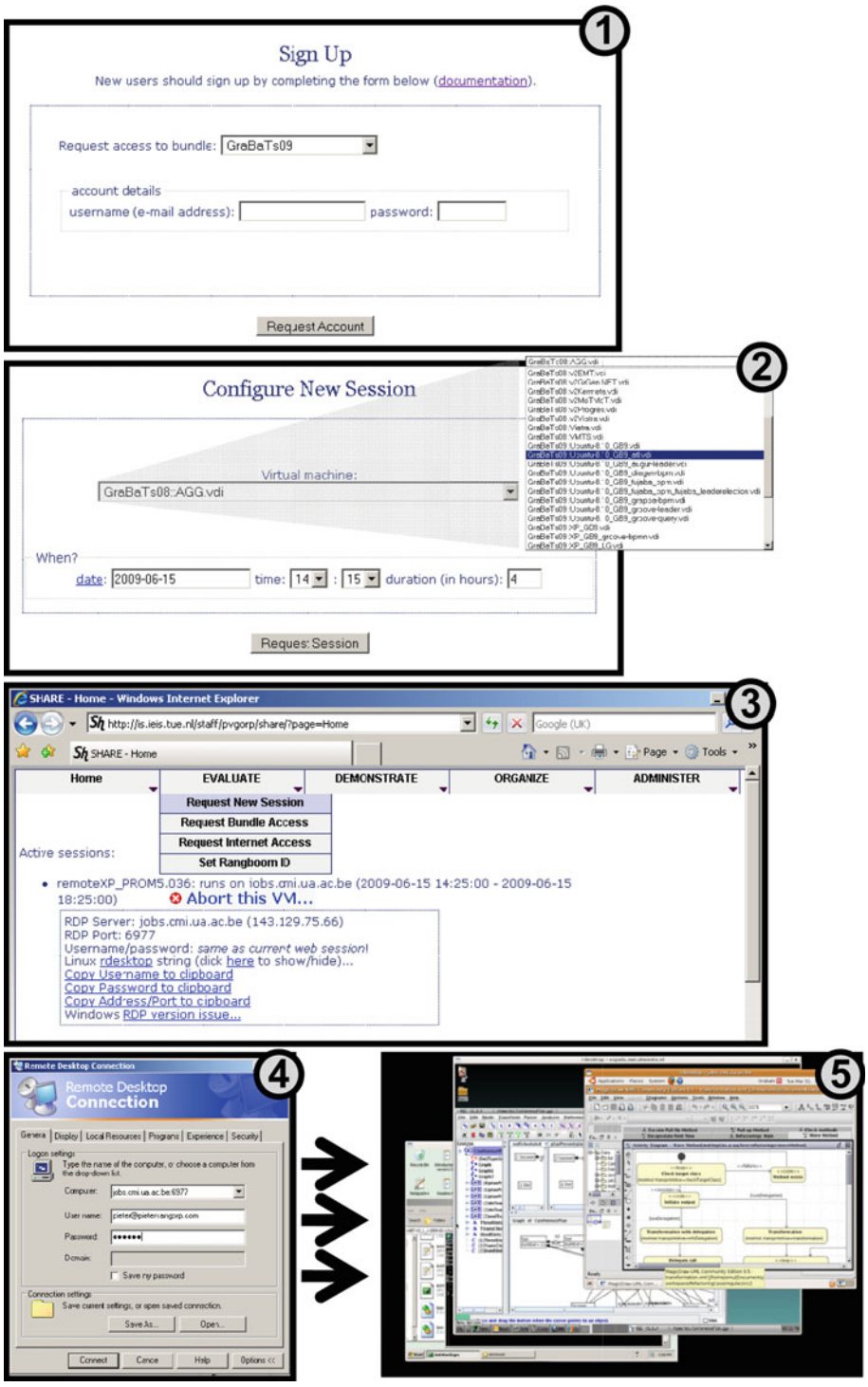



**Fig. 3** Typical walkthrough for evaluators

to the numbers from Fig. 4. It should be noted that the graph-based tools contest involved several case studies, some of which could be solved in advance (offline case studies) and one of which had to be solved during the workshop (the live contest) [52]. The workshop attracted two submissions based on the *GrGen* tool: one submission to an offline case study and one submission to the live contest. The authors of the offline submission completed a SHARE demo before the workshop. The following walk-through indicates how one of the authors of the live contest submission could leverage that for quickly completing a new demo dedicated to his own article.

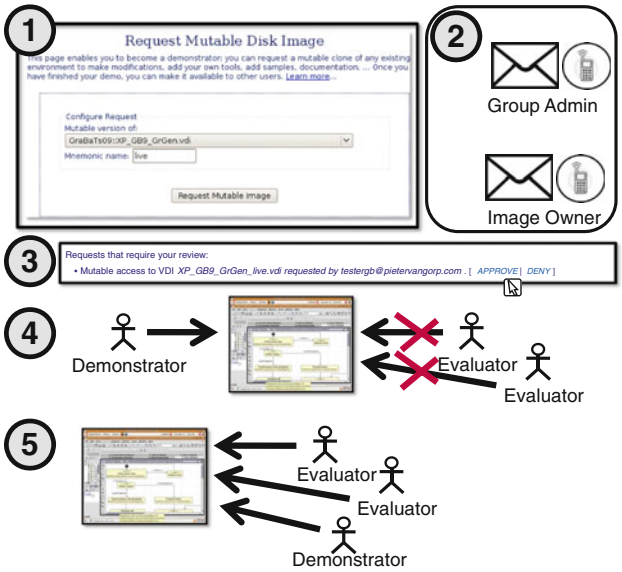



**Fig. 4** Typical walkthrough for demonstrators

1. Figure 4, ① shows how this author first indicates the image that he wishes to clone. If there would not yet be a *GrGen* demo in SHARE, the author would start from a rudimentary operating system image (e.g., *XP_GB.vdi*, an image from the GraBaTs09 group that only contained *Windows XP*). In fact, this is what the other *GrGen* authors had done before the workshop. These other authors added the "GrGen" suffix to the name of their cloned image. To save time, the new author decides to request a clone from this image (Fig. 4, ① shows how *XP_GB_GrGen.vdi* is selected) and adds the suffix "live" to indicate that the new image will relate to the live contest. This name will be visible later to other users who are browsing the list of available demonstration images in the *GraBaTs09* group.
2. After the request for a new image has been submitted, SHARE sends an automatic message to the owner of the original image as well as to the administrator of the group in which that image is contained. As indicated by Fig. 4, ②, such messages can be retrieved via multiple communication channels.
3. The administrator and the image owner can either approve or deny the requests for new images. Figure 4, ③, shows the related fragment from the user interface. In our running example, the owner of the image as well as the issuer of the request are members of the development team of the *GrGen* tool. The administrator of the corresponding SHARE group is organizing the tool contest. Therefore, both parties approve the request for the new image.
4. After these approvals, SHARE creates a clone of the original virtual machine image. As indicated by Fig. 4, ④, this image is initially visible to the demonstrator only. Using steps ② to ⑤ from the workflow related to Fig. 3, the demonstrator can connect to the image remotely and install additional software and documentation at will. The main difference to what has been discussed in the context of Fig. 3 is that demonstrators have stateful access to their image: changes are preserved across sessions. In our running example, the author of the live contest submission simply replaces the input models of the original demo by new ones.
5. Once the virtual machine contains the complete environment as intended by the demonstrator, this demonstrator can publish the image to the other group members. In the example given, the image then becomes visible to all the members of the *GraBaTs09* group. All uses of the virtual machine image will be stateless from this moment on, including sessions initiated by the demonstrator himself. This ensures that users can refer in a dependable manner to a SHARE demonstration. In fact, all SHARE demonstrations can be compared to library artifacts that can be referenced from research articles and other documents.

The scenario described above does not discuss the possibility to deny requests for new demonstration images. If either the owner of an image or the organizer of the group denies such request, the issuer of the request will be notified automatically. The scenario also does not elaborate as to what happens when a demonstrator starts creating an image and later decides he does not want to publish it to the group. In that case, SHARE enables such demonstrators to discard the stateful image. Finally, the scenario does not indicate explicitly how updates to demo images are supported. In fact, in order to make such images true library artifacts, SHARE intentionally only supports such updates through the creation of new clones, via the workflow described above.

### 3.3 Organizer walk-through

The aim of this article is to activate other event (workshop, conference, . . .) organizers to administer their own SHARE groups. The other sections of this article should have convinced such organizers that it is desirable to have electronic demonstrations available to the reviewers and target audience of an event. In turn, this section should convince organizers that the administrative overhead is manageable.

An organizer is responsible for the following tasks: (1) managing group access, (2) managing clone requests, and (3) advertising the images in the group. As indicated in Sect. 3, new users can sign up online. Sign-up requests need to be approved by the organizer. In order to protect such organizers from fake (spam) form submissions, SHARE only forwards requests from verified e-mail addresses. In practice, an organizer then checks whether the e-mail address relates to an academic individual and clicks a generated hyperlink

to either approve or deny the sign-up request. Additionally, SHARE provides organizers a tabular user interface to revoke group access later in time.

As indicated in the previous section, the second organizational task (i.e., managing clone requests) is supported by automatic e-mails with direct hyperlinks for approval or denial. Finally, the system facilitates the advertisement of the images in a group by providing the organizers with automatically generated HTML code for inclusion in call-for-paper announcements.

As a concrete example scenario, consider the following steps that "John", the organizer of the hypothetical workshop "OOPSLA101", should follow as a new SHARE group organizer:

1. John signs up to the group "Operating Systems"[48], using his e-mail address john.doe@uni.ac.eu and a password of his choice.
2. John receives an automatic e-mail from SHARE and clicks a hyperlink to verify his john.doe@uni.ac.eu address.
3. The organizer of the "Operating Systems" group receives an e-mail describing John's request and approves it by clicking a hyperlink.
4. John receives an e-mail announcing the approval of his request.
5. John clicks a hyperlink from that e-mail, enters his password and logs into SHARE.
6. John can now evaluate the operating systems that are available in SHARE by default, following the steps described in Sect. 3.1 or following the online documentation[50]. Since John's workshop is unrelated to existing groups, John does not investigate which software is already installed in images from other groups. In fact, John decides that in the context of the OOPSLA101 workshop, all participants should be able to install their software on machines running the Ubuntu 8 variant of Linux or the XP variant of Windows.
7. John uses the SHARE website to contact the SHARE support staff. He indicates that he wants to become organizer of a new group, called OOPSLA101 and would like to initialize that group with an Ubuntu and an XP image.
8. A "group administrator" creates that new group via the web interface: in summary, the two base images are cloned, the clones are deployed to at least one virtual machine server, and finally registered to the new group. Afterwards, John is notified by e-mail.
9. John finalizes the call-for-papers of his workshop by stating that for software-based contributions, a demo should be made available in group OOPSLA101.
10. Before the submission deadline, John will receive various registration requests for his SHARE group. After the deadline, he indicates to the reviewers that software attachments can be inspected in the group OOPSLA101.

The approval of clone requests has been discussed in the previous section already. The final responsibility of John is the advertisement of the images in the group OOPSLA101. On the one hand, John is already supported by the SHARE website, since that lists all available images for all public groups. On the other hand, John can invite the participants to his workshop to use the HTML and LaTeX code that SHARE generates for their demonstration images. Such code lists the metadata of each image (name and owner) and a hyperlink for directly initiating a session for a particular image.

Figure 5 displays a screenshot of a session from a user that owns three images (one from the *GraBaTs08* group and two from the *GraBaTs09* group). The first banner holds a direct link for starting a virtual machine based on the image under consideration. The second banner launches a popup to post such a link to a social networking site. The third banner integrates with a website for collaborative bookmarking and tagging [23]. The LaTeX/BibTeX code facilitates one to reference the demo image from a research article. With such features, we want to assist academics as much as possible in advertising their reproducible research.

### 3.4 Administrative features

SHARE defines two types of super users: (1) host administrators and (2) group administrators.

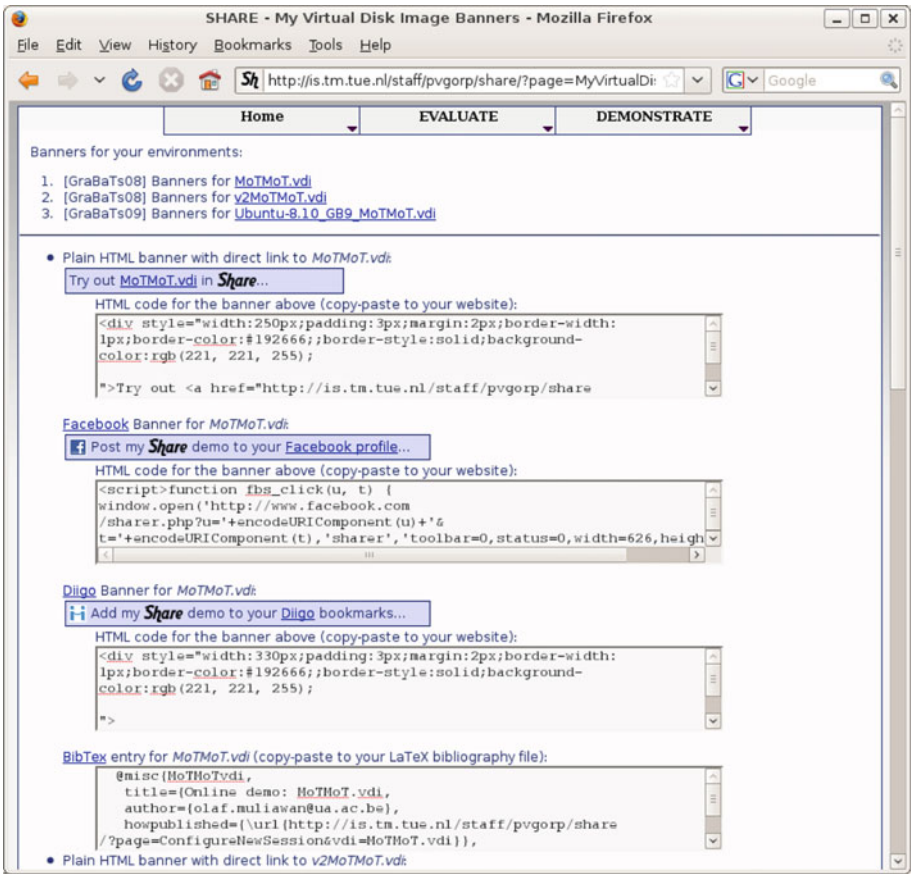
The first type of users is responsible for monitoring hardware (virtual machine server) resources and for managing internet access. SHARE is designed to make full internet access unnecessary in most cases. However, in order to minimize the overhead when it does become necessary (e.g., to perform an online operating system update), the system provides an automatic messaging system and a tabular interface for managing internet access on virtual machine servers. In some cases, it turns out that a particular virtual machine server is (or may become) a bottleneck. In such cases, a SHARE host administrator can easily replicate virtual machine images to additional virtual machine servers. More specifically, SHARE generates the supportive migration scripts automatically.

The second type of super users should facilitate the organizational scalability of the system: a *group administrator* can create new virtual machine groups and assign organizers to groups.

## 4 Implementation: how?

This section puts the example walk-throughs from the previous section in a broader perspective. By describing several

**Fig. 5** HTML and LaTeX code




models of SHARE, users should better understand the design rationale of the system, whereas others may use these models to improve the system's implementation.

### 4.1 Conceptual data model

Figure 6 displays a class diagram that models SHARE's data from a conceptual point of view: all classes in the diagram correspond to persistent entities [3,22]. The diagram contains several applications of stereotypes such as $\ll K \gg$ and $\ll NN \gg$. The former stereotype indicates that a particular attribute can be used as a *K*ey of its class whereas the latter indicates that a particular attribute should *N*ot have a *N*ull value.

The center of the diagram contains the *User* class. As indicated by association "*Registered for*", a user can be registered for multiple groups. The *organizerApproved* attribute of association class "*Registered for*" ensures that users only have access to a group's images after approval from the organizer. A result from an early design decision is that an image belongs to exactly one group (remark the UML composition between *Group* and *Image*). This decision was made to limit the amount of administrative messages sent by SHARE. More specifically, this design ensures that when negotiating access to an image (for registration or for cloning), the administrator of only one group needs to be contacted for approval. We do not claim this decision is optimal in all cases but it turned out to be valid so far.

The most notable attribute from the *Image* (from "Virtual Machine Image") class is *maxduration*. This attribute enables demonstrators to prohibit production-use of their research demo. With regards to the *minMemory* attribute, the system is configured to assign a fixed amount of main memory (i.e., 1 gigabyte) to each virtual machine instance. Some images however require more than that default.

With regards to the "VM usage" class, it may be worth mentioning that the system dynamically allocates ports on the virtual machine server. The *serverPort* attribute represents the number of the port that will be available at least for the time frame that has been reserved via the *start* and *durationHours* attributes. The allocation is based on instances of the "Reachable port" class (displayed at the top right of Fig. 6). These instances can be used to align SHARE with some university-specific firewall settings.

The "*VMU state*" enumeration (displayed at the top left of Fig. 6) is used to represent whether a machine can already (or still) be accessed remotely, whether it still needs to be started by the virtual machine server, or whether it has already been terminated. The "*Clone Of*" association is used to represent which image is cloned from which other one. Recall that

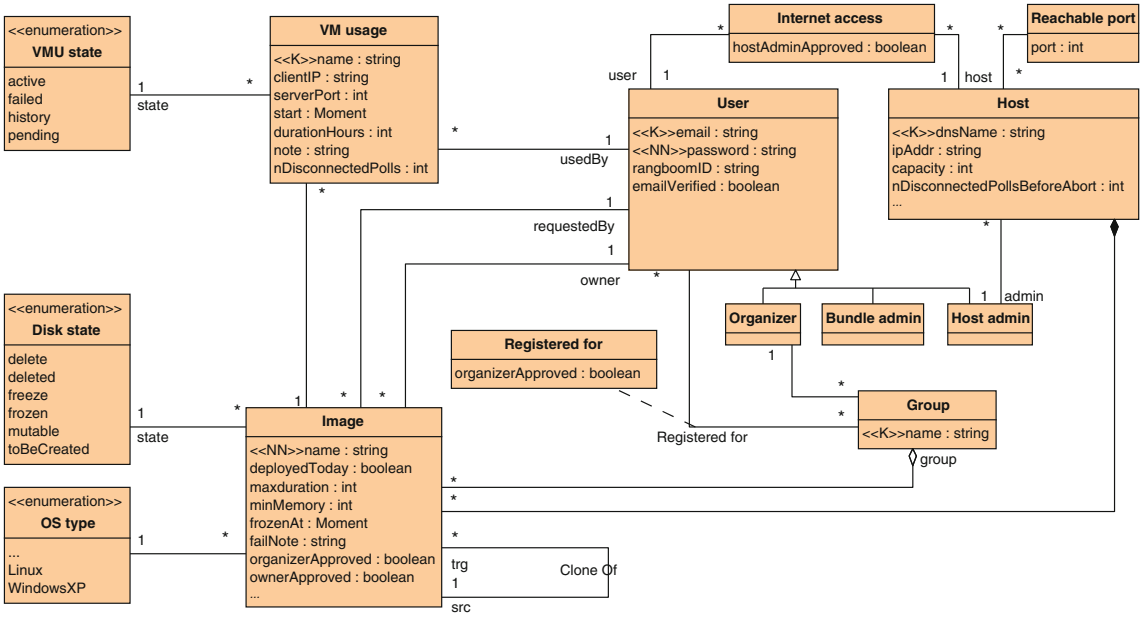



**Fig. 6** Conceptual data model

for the protection of intellectual property and licenses, each cloning operation needs to be approved by the owner of the source image (see association end "*owner*") and the organizer of the corresponding SHARE group. After cloning, the user who requested the clone (see association end "*requested by*") has exclusive and stateful access to the image. This user can then install the new software that needs to be demonstrated. Once complete, the user can "freeze" (see enumeration "*Disk state*") the image and make it visible to all other users of the corresponding group.

### 4.2 Process simulation model

This section presents an executable model of the primary features. The model is written in the high-level colored petri-net formalism [24], using CPNtools [25]. The model serves the following goals:

- the model enables one to step through some concrete scenarios in order to better understand the system behavior under particular circumstances,
- the model can be used for simulation purposes. More specifically, the model consists of specific submodels (one submodel for each double-bordered box in Fig. 7) and contains timing information for each primitive event. This enables one to estimate the average time required to perform a sequence of activities under particular environment conditions. For example, one can use the simulation model to assess the effect of adding a new virtual machine server on the waiting times for end-users,
- the model can serve as a blueprint for re-implementing SHARE on top of a process execution engine.

Figure 7 shows the entry point of the hierarchical colored petri-net. An oval is called a "place" in the petri-net formalism whereas a box is called a "transition". By modeling convention, a black place represents a page where an end-user can reside. Most transitions shown in Fig. 7 represent a click on a menu item (and most likely some follow-up activities such as entering parameters). Transitions can also encapsulate autonomous system behavior. Transition "startVM" for example models the behavior of a cron job on the web server (as discussed in Sect. 2.2). The detailed timing behavior is modeled on the next level of abstraction (in a so-called *subpage* of the hierarchical net).

Again by modeling convention, gray places represent persistent data. Notice that the associated data structure is derived from the conceptual data model discussed in Sect. 4.1. In another research project, we are developing model transformations for maintaining the consistency between these models and the underlying database code.

Each place can contain so-called "tokens". For black places, tokens represents users that reside on the corresponding webpages. The "green" rectangles show the token values after the execution of more than 1000 events. The green rectangles indicate that at that point, there is still one user on the system's start page (token "Pieter" in place "View Startpage" while two other users reside on the main page (tokens "Hans" and "Rik" in place "Reside on Main Page").

For gray places, tokens represent information in the system database. The place "Auth DB" for example holds all

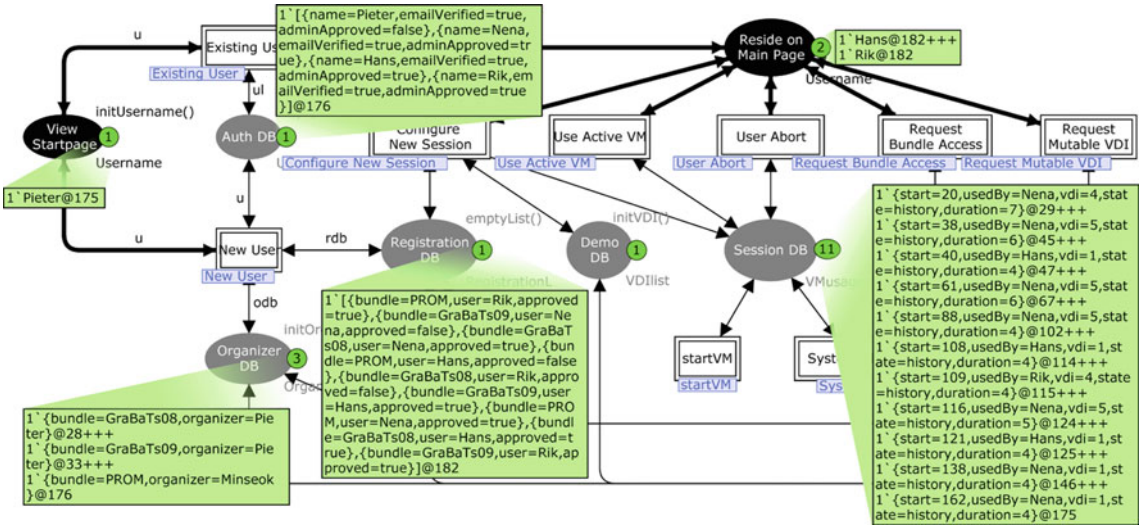



**Fig. 7** Simulation model

authentication information. The connected green rectangle indicates that four users are known to the system: "Pieter", "Hans", "Rik" and "Nena". All accounts, except that of "Pieter" are approved by a group organizer (which is why "Pieter" was unable to login).

The place "Registration DB" holds a token that represents the group registration information. Notice that this information is generated randomly by the simulation engine, based on statistical annotations in the simulation model. In the state displayed in Fig. 7, all but three registration requests were approved by the organizer of the corresponding group.

Finally, place "Session DB" holds a token that represents the history of virtual machine sessions. In the state displayed in Fig. 7, there are no active sessions. Instead, all sessions have state "history". By firing transition "Configure New Session", new sessions can be initiated dynamically. Obviously, we are convinced that readers can better understand this executable model by further exploring it using the right tool. Therefore, users are invited to connect to a SHARE demo that automatically starts the right modeling tool and loads the simulation model for evaluation purposes [46].

### 4.3 Usage, stability and maintenance of the platform

At the time of writing, the primary SHARE installation runs on three virtual machine servers. Each server has multiple cores (2, 4 and 8, respectively) and an amount of main memory that is typical for entry-level university servers (4, 8 and 16 gigabytes, respectively). So far, the scarcest resource is hard drive space and we are working with an academic partner to address that [35].

The most powerful server is running at the University of Antwerp (UA, Belgium) and this server has served the SHARE cloud for more than 1 year at the time of writing. The second-most powerful server is running at the University of Twente (UTwente, The Netherlands) and entered the cloud just 1 month before the time of writing. This server has replaced another SHARE machine at the same university that has served the cloud for about 1 year. The least powerful server is running at the Industrial Engineering department of Eindhoven University of Technology (TU/e, The Netherlands) and is serving the cloud already for about a half-year.

Temporarily, the cloud has also been served by a cluster of machines running at the Computer Science department of the same university (TU/e). This was done to anticipate a heavy load during the 2009 edition of the aforementioned graph-based tools workshop. During that 2-day workshop, about 130 virtual machines were started but this high number is partly due to users *playing* with the system (which involves quickly *starting* and *stopping* virtual machines). Even during that peak in actual system usage, neither CPU nor main memory turned out to be a bottleneck. Instead, the hard drives of individual cluster nodes were too small to hold *all* virtual machine images that could be of interest to the workshop. Therefore, the cluster nodes were configured with distinct (yet overlapping) sets of images. After the workshop, the cluster nodes were "returned" to their owner simply by configuring them as "offline" in SHARE.

From the perspective of software robustness, the quality of the underlying hypervisor (VirtualBox) is sufficient. Apart from a minor bug (see [47]) the software performed remarkably well even for users on lower bandwidth wireless networks. We have updated the VirtualBox executables on the aforementioned servers at various times as part of the general server maintenance routines. Just recently, the VirtualBox command-line API has changed slightly. Therefore, we had to update SHARE's source code. Due to the design that we described in Sect. 2.2, all changes could be localized

to the Bash scripting layer. That design even supports phased upgrades: we upgrade the hypervisor on a testing machine first and upgrade operational servers incrementally.

In summary, we have shown that the SHARE system can effectively deal with highly varying usage loads. We have also explained how the architecture easily supports hardware scalability and controlled software upgrades. We invite other research institutes to make available computational resources as the popularity of the system increases. Moreover, we will apply for SHARE-specific funding to sustain its deployment, maintenance and further development.

## 5 Related work

In this section, we discuss related work from three perspectives. The first set of related articles motivates reproducibility from the viewpoint of science consumers as well as from an author point of view. Secondly, the SHARE method is related to existing methods in the reproducible research domain. Finally, this section relates the SHARE platform to other platforms in the cloud-computing domain.

### 5.1 Reproducible research: why?

Vandewalle et al. observe that although Descartes already stressed the importance of reproducibility in 1637 [16], fraudulent articles still slip the peer-reviewing process of established journals in mature fields [53,58]. On a brighter side, Vandewalle et al. also describe promising results from a study on the correlation between research reproducibility and research impact. The study is based on a survey on the availability of the algorithm, code, and test data related to articles in *IEEE Transactions on Image Processing* in 2004. The study is based on 134 articles and 90 reviewers. As an obvious observation, reproducibility does not *guarantee* citation impact. More interestingly, the study confirms that articles with a high citation impact have code and data online in most cases. Similarly, Piwowar et al. [38] have examined the citation history of 85 biomedical publications. On the one hand, that study is narrower since it only considers research data (not code or configuration files). On the other hand, the conclusions can be generalized more easily: a linear regression analysis points out that independently of journal impact factor, date of publication, and author country of origin, reproducibility has a positive influence on citation impact.

### 5.2 Reproducible research: how?

Probably the most influential contribution to computational research reproducibility was developed in the geophysics domain, by the team of Claerbout between 1990 and 1995 [12]. The team already applied automatic build tools to produce CD-ROM images that contained: (1) a research article, (2) the TEX source of that article, (3) all Unix code and data related to that article, (4) Unix scripts to rebuild figures from the article automatically, perhaps after changing some parameter values or making some code changes, and (5) a special purpose TEX viewer to trigger these scripts while reading the research article.

Unfortunately, the CD-ROM images could not be used by researchers using platforms other than Unix. This may have inspired Donoho et al. to develop the WaveLab library [9,18]. Wavelab (and similar successors such as Madagascar [19]) is an online repository of research code and data, organized by research article. By using MATLAB as a computation platform, the approach was no longer limited to one operating system. Unfortunately, this operating system flexibility came at the cost of the reader's experience. More specifically, the hyperlinks from article figures to supportive data and code were again considered as future work:

> One way to do this would be if journals were fully electronic, and if we adopted hypermedia techniques. . . . If one were interested in a figure, one would click on it with a mouse, and a new window would instantly appear, containing the code that the author of the article used to create the figure. [9]

Today, there are no technical obstacles anymore for realizing this vision. In the concrete example of WaveLab, one can create a SHARE image with (1) MATLAB, (2) the WaveLab module of a particular article, (3) a startup script that initializes MATLAB with the code of a particular figure. By using citations such as [46] (cfr., discussion of Fig. 5), authors can enable readers to jump directly to the optimal environment. Those that are concerned with hard disk limitations of their virtual machine servers can adopt the practice of having one SHARE demo per article and on the desktop of their virtual machine one tool startup script per figure (or other obvious candidate for reader verification). Without SHARE, authors would face the concerns described by Vandewalle et al.:

*Platform Standardization* "Can we suppose that programs such as MATLAB, Mathematica, or even Windows are available, or should we constrain ourselves to the use of open-source programs such as R, Octave, or SciLab in a Linux environment?" [53]

*Balancing Author and Reader Effort* "There exists a large variety of computer platforms, and making even some simple code work on most platforms requires a lot of work. We need to balance the ease of use for a reader with the additional work to be done after development of the algorithms and initial code. . . . " [53]

*Durability* "In this tradeoff, it is also important to consider the long-term availability of a platform. Due to changes in data formats, compilers, and research platforms, research is often only reproducible in a limited time window." [53]

Regarding *standardization*, we very much welcome the development and use of interoperability standards in industry. In fact, we are also working on a UML-based technique to model transformation interoperability [51]. We also welcome the other approaches to model-based interoperability, as presented in articles of this Special Issue. However, from the organization of the transformation tool contest (see Sect. 1.1) we conclude that also in the area of model-driven development one should never *constrain* researchers to the use of a particular file format or tool bus. Moreover, even *if* a community adopts a standard platform (such as MATLAB in the signal-processing domain [18]), we advocate that demonstrations should be made available via a platform such as SHARE, because (1) results should be reproducible with minimal effort by readers from outside that community too, and (2) because of the *durability* concern.

Regarding the *effort* concern, consider for example the amount of work that authors need to perform for adopting heavyweight integration approaches such as the *Electronic Tool Integration Platform* (jETI [30,32,42]). The idea of approaches such as jETI is that authors make their code callable remotely using web service technology. Other researchers can then treat that service as an online test oracle. Again, in the context of the transformation tool contest, we have invited all participants to investigate: (1) how to *call* the relevant part of their tool with some standard inputs, (2) how to get the transformed results (and some statistics), and (3) how to pass that to a visualizer, which could be a service itself [52]. Participants preferred a virtual machine-based approach, because (1) that did not require them to extend their tool and (2) the reader would get a realistic experience of their tool's mature user interface (instead of a web-based interface that would be constructed in an ad-hoc manner). SHARE is designed and maintained based on private feedback from this and other kinds of users. At the time of writing, we have not yet performed a large scale evaluation survey. However, we plan to do so, we will make available the results and we will use these results to prioritize further research and development.

Regarding *durability*, we support Claerbout's observation that every platform eventually becomes obsolete [20]. Since SHARE relies on virtualization, it does not matter that a platform on which a research demo is installed, becomes obsolete. A hypervisor will continue to run the self-contained image on modern hardware while users will perceive the complete environment in exactly the same way as when it was first created. Librarians do not have to worry about the dependencies of individual contributions anymore. They only need to ensure that (1) all images are digitally preserved and that (2) there is a version of the hypervisor that can boot the format of the images and that runs on a physical machine. Notice that SHARE administrators can use a standard format for virtual machine images without losing generality. Focusing on the conversion between virtual machine image formats, or developing techniques to even run hypervisors virtually is much more productive than focusing on domain-specific data formats, or software migrations.

Notice that the "*A*" from the *SHARE* acronym (standing for "*Autonomous*") represents a key characteristic related to long-term reproducibility (durability): images should be self-contained because the use of artifacts from outside the SHARE cloud (e.g., artifacts from the public internet) may lead to version inconsistencies. Therefore, even demos of distributed software should (and usually can) be installed on the *localhost* of one SHARE virtual machine. Technically, we can extend the platform to also support virtual machines that communicate with other virtual machines in the durable cloud. At the time of writing, the virtual cluster functionality is however not a priority. Whenever implementing such advanced features, one should keep the workflows and user interface simple enough for users who do not require/understand the advanced functionality.

A special instance of distributed software that is quite popular nowadays is software that relies on public web services. Again, for long-term reproducibility, one should provide a *stub* for each of such services within the durable cloud. Fortunately, this should not be an additional burden to the researcher since the development of such stubs is increasingly considered an integral part of development [11,60]. Nevertheless, SHARE does not provide a solution for those service-oriented software contributions for which no offline test stubs are available.

Apart from the technical aspects mentioned above, Stodden analyzes different license forms (for media as well as software) with computational science reproducibility in mind [43]. Stodden concludes that there is no license that fits all purposes of different researchers. The article also indicates that data cannot be copyrighted. On the other hand, Stodden indicates that the effort of preparing metadata and filtering data could legally be protected by means of an attribution license. In our experience however, scientists usually do not have the resources to check whether their code or data are used in accordance with license restrictions. In fact, even choosing an optimal license may take more time than the available window of opportunity for submitting the related article. Perhaps this explains the very limited success of research data repositories? For example, in the software-modeling domain, the *Atlantic ZOO* is the largest repository of public research data. The repository has already been moved due to intellectual property issues with the *Eclipse* initiative [7]. Moreover, it turns out that no companies agree to make industrial models available for download there. With SHARE, one can keep industrial models in safe connection-less environments on virtual machine servers that are administered directly by the companies that own the models.

**Table 2** Classification of work on reproducible research

| | Focus | Bottleneck |
|---|---|---|
| Open source [36] | Program code availability | Author and reader effort |
| | | Platform durability |
| | | Scope: commercial contributions? |
| ETI, dETI, jETI [30,32,42] | Online test oracle | Author effort |
| | | Scope: no text/data integration |
| Wavelab, Madagascar [9,19,20] | Figure/table reproducibility | Platform standardization |
| | | Platform durability |
| | | Scope: interactive software? |
| SHARE | Holistic reproducibility | Hypervisor durability |
| | | Legal durability |
| New license types [43] | Legal durability | Lack of precedents |

Table 2 summarizes the discussion from this section in a schematic manner. The table can be used to guide future work on reproducible research. For example, in our ongoing work we are anticipating the legal aspects related to SHARE. As stated in Sect. 2.1 already, we are investigating how electronic contracts can formalize approvals from external parties that own artifacts that are included in a demo image. At the time of writing, each demonstrator takes responsibility for the rights on *all* the material that he uploads to the SHARE cloud. To ensure demo *durability* also from a *legal* perspective, one should involve vendors of operating systems (or more specialized packages of commercial software) in the workflows discussed in Sect. 3.2. The SHARE data model (see Sect. 4.1) should then also support the representation of approvals and all digital approvals should be *non-repudiable*. Interestingly, SHARE group organizers can already restrict image access to academic users (who can acquire evaluation licenses for almost any commercial software package anyhow) and thereby anticipate most *practical* legal concerns.

### 5.3 Cloud computing: other platforms

Nimbus is a cloud-computing platform based on the XEN hypervisor. It is aimed at exchanging and controlling virtual machine images and associated metadata across the internet. Keahey et al. use Nimbus to create virtual clusters dynamically. As a running example, the authors dynamically aggregate computational resources from three different universities to satisfy the service level agreements (SLAs) for heavyweight biomedical computations [26]. Executing long-running jobs according to SLAs is a typical scenario in high-performance computing in general and in e-Science in particular [15].

Nimbus and SHARE are similar in that both platforms deal with the metadata of virtual machine images. Both projects have a different background though: the Nimbus platform has an *e-Science* background and has thus been designed for executing computational jobs on a pool of long-running virtual machines. Typically, multiple machines are started automatically (behind the scenes) to support scientific workflows. The user (a researcher) typically does not have direct access to the underlying virtual machine sessions since these sessions tend to be stateful and tend to serve other users too.

In contrast, the SHARE platform has a *Reproducible Research* background and has thus been designed for (1) direct, and properly isolated, access to short-running virtual machine sessions that have been started explicitly by the user and (2) for image sharing. As a result, Nimbus has a more sophisticated API for *monitoring* runtime VM performance (e.g., to check computational SLAs) whereas SHARE has a more specialized user interface for image sharing (*cloning* and *advertising*).

Eucalyptus [34] and the Grid Virtualization Engine (GVE [57]) are open source cloud-computing platforms that are similar to Nimbus but that emphasize the use of *standard* web service technologies such as WSDL and BPEL more. Amazon provides a commercial cloud computing platform, called *EC2* [2]. The EC2 virtual machines are called *elastic*, since Amazon customers can *dynamically* change the amount of physical server resources that are allocated to their running virtual machines.

Interestingly, Eucalyptus machines can be controlled via the EC2 management API too. Over time, it may become worthwhile to investigate how SHARE images (in VirtualBox or VMware format) could be deployed (1) to external university servers running Eucalyptus, or (2) to Amazon infrastructure. In the latter scenario, an end-user would enter his Amazon credentials in SHARE and his Amazon computational credits would then be used for running SHARE demos. Table 3 summarizes the discussion from this section in a schematic manner.

## 6 Conclusions

This article introduces a new approach to making software-related research reproducible. SHARE, the supportive platform, has emerged to solve concrete reproducibility problems in the domain of model transformation. It turns out that about 15 years before, a conceptually similar approach was already proposed by Claerbout et al., in the domain of geophysics. SHARE solves several problems of such previous platforms by applying recent virtualization and web technologies. The system has been applied successfully by several workshops.

At the time of writing, some SHARE groups are already administered by non-experts. In summary, authors create a

**Table 3** Classification of cloud computing platforms

| | Focus | Limitation |
|---|---|---|
| Nimbus [26], Eucalyptus [34], GVE [34] | High performance computing | Sharing demo images |
| Amazon EC2 [2] | Commercial elastic computing | Sharing demo images |
| SHARE | Sharing (evaluating/cloning) | SLAs |

new demo as follows: (1) they register for a particular group, (2) they request a clone of a virtual machine image, (3) they install additional software, data and documentation, and (4) they publish their image to the group again. Apart from step (3), authors spend less than a half-hour on SHARE. Therefore, we advocate that the editors of journals and other publication types should invite authors to adopt, or at least evaluate, the proposed approach as soon as possible. This should result in the emergence of publications that are much easier to verify and that are *durable*, without asking unreasonable *effort* or *standardization* from authors.

In our ongoing work, we are investigating how the legal durability of SHARE demos can be guaranteed by means of electronic contracts between demonstrators, organizers, host administrators and commercial software vendors. Additionally, we are building a set of best practices for creating SHARE demos. This involves guidance on the use of desktop organization software (such as Fences [41]), screen capture software (such as Wink [27]) and other packages that are complementary to SHARE. A possible extension of SHARE may later support the storage and retrieval of screen recordings (as well as more structured user logs) as feedback to demonstrators. Finally, we are monitoring the evolution of general purpose cloud-computing platforms (such as Eucalyptus and EC2) and hope for the emergence of a standard format for virtual machine images as well as a standard API for exchanging images and accounting resource usage to end-users.

**Acknowledgments** The authors wish to thank Stefan Blom and Axel Belinfante for their contributions to the SHARE source code. Additionally, we wish to thank Leon Osinski for organizing an excellent workshop related to reproducible research [37]. Finally, we wish to thank Marcel Hartgerink (from *Wibu Systems*) and Arnoud Engelfriet (from *ICTRecht*) for fruitful discussions related to the technical and legal aspects of software licenses.



## References

1. Agrawal, A.: Graph rewriting and transformation (GReAT): a solution for the model integrated computing (MIC) bottleneck. ASE **0**, 364 (2003)
2. Amazon.: Amazon Elastic Compute Cloud (Amazon EC2). Amazon. http://aws.amazon.com/ec2/#pricing (2009)
3. Ambler, S.W.: A UML profile for data modeling. http://www.agiledata.org/essays/umlDataModelingProfile.html (2009)
4. Angelov, S., Grefen, P.: The 4W framework for B2B e-contracting. Int. J. Netw. Virtual Organ **2**(1), 78–97 (2003)
5. Barrett, D.J., Silverman, R.E., Byrnes, R.G.: SSH: The Secure Shell: The Definitive Guide, 2nd edn. O'Reilly and Associates (2005)
6. Bézivin, J., Dupé, G., Jouault, F., Pitette, G., Rougui, J.E.: First experiments with the ATL model transformation language: transforming XSLT into XQuery. In: OOPSLA—Generative Techniques in the Context of Model Driven Architecture. Anaheim, California (2003)
7. Bézivin, J., Jouault, F., Brunelière, H., Garces, K., Combemale, B., Sottet, J.-S., Kleiner, M., Doux, G., Tisi, M.: Zoos. http://www.emn.fr/z-info/atlanmod/index.php/Zoos. Accessed Mar 2010
8. Biermann, E., Ehrig, K., Köhler, C., Kuhns, G., Taentzer, G., Weiss, E.: Emf model refactoring based on graph transformation concepts. ECEASST—Electronic Communications of the EASST, vol. 3 (2006)
9. Buckheit, J., Donoho, D.L.: Wavelets and Statistics, vol. 103. In: Wavelab and Reproducible Research, pp. 55–81. Springer-Verlag, New York (1995)
10. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. **25**(6), 599–616 (2009)
11. Chen, J.-Y., Lin, C.-H.: Scenario-based service specification and testing. J. Softw. Eng. Stud. **2**, 69–80 (2007)
12. Claerbout, J.: Electronic documents give reproducible research a new meaning. In: Proc. Ann. Int. Mtg Soc. Expl. Geophys., pp. 601–604 (1992)
13. Cuban, L.: Oversold and Underused: Computers in the Classroom. Harvard University Press (2003)
14. Decker, G., Overdick, H., Weske, M.: Oryx—sharing conceptual models on the web. In: ER'08: Proceedings of the 27th International Conference on Conceptual Modeling, pp. 536–537. Springer-Verlag, Berlin (2008)
15. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-science: an overview of workflow system features and capabilities. Future Gener. Comput. Syst. **25**(5), 528–540 (2009)
16. Descartes, R.: Discourse on Method. Jan Maire, Leiden (1637/1991). Available in The Philosophical Writings of Descartes (trans: Cottingham, J., Stoothoff, R., Murdoch, D., Kenny, A.). Cambridge University Press
17. Dittner, R., Rule, D.: The Best Damn Server Virtualization Book Period. Elsevier, Amsterdam (2007)
18. Donoho, D., Maleki, A., Rahman, I., Shahram, M., Stodden, V.: Reproducible research in computational harmonic analysis. Comput. Sci. Eng. **11**(1), 8–18 (2009)
19. Fomel, S.: Madagascar—reproducible documents. http://www.reproducibility.org/wiki/Reproducible_Documents. Accessed Mar 2010
20. Fomel, S., Claerbout, J.: Guest editors' introduction: reproducible research. Comput. Sci. Eng. **11**(1), 5–7 (2009)
21. Geiger, L., Zündorf, A.: eDOBS—graphical debugging for eclipse. In: 3rd International Workshop on Graph-Based Tools (GraBaTs) ICGT Workshop, Natal, Brasil (Sept 2006)

22. Halpin, T., Bloesch, A.: Data modeling in UML and ORM: a comparison. J. Database Manag. **10**(4), 4–13 (1999)
23. Hong, L., Chi, E.H., Budiu, R., Pirolli, P., Nelson, L.: Spartag.us: a low cost tagging system for foraging of web content. In: AVI '08: Proceedings of the Working Conference on Advanced Visual Interfaces, pp. 65–72, New York, NY, USA. ACM (2008)
24. Jensen, K., Kristensen, L.: Coloured Petri Nets. Springer-Verlag, Berlin (2009)
25. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. Int. J. Softw. Tools Technol. Transf. **9**(3), 213–254 (2007)
26. Keahey, K., Tsugawa, M., Matsunaga, A., Fortes, J.: Sky computing. IEEE Internet Comput. **13**(5), 43–51 (2009)
27. Kumar, S.: Wink. http://www.debugmode.com/wink/. Accessed Mar 2010
28. Kurp, P.: Green computing. Commun. ACM **51**(10), 11–13 (2008)
29. Lerdorf, R.J., Tatroe, K., Kaehms, B., McGredy, R.: Programming PHP. O'Reilly and Associates (2002)
30. Margaria, T.: Web services-based tool-integration in the ETI platform. Softw. Syst. Model. **4**(2), 141–156 (2005)
31. Mylopoulos, J., Chung, L., Yu, E.: From object-oriented to goal-oriented requirements analysis. Commun. ACM **42**(1), 31–37 (1999)
32. Nagel, R.: How to setup a jETI service provider (jETI server). http://jabc.cs.tu-dortmund.de/manual/index.php/JETI#Additional_Documentation. Accessed June 2006
33. Newham, C., Rosenblatt, B.: Learning the Bash Shell, 2nd edn. O'Reilly and Associates (1998)
34. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The eucalyptus open-source cloud-computing system. In: Proceedings of 9th IEEE International Symposium on Cluster Computing and the Grid (2009)
35. D.U. of Technology, E. U. of Technology, and U. of Twente.: 3TU. Data Centre. http://datacentrum.3tu.nl/. Accessed Mar 2010
36. Open Source Initiative.: Open source licenses by category. http://www.opensource.org/licenses/category. Accessed Mar 2010
37. Osinski, L.: Research data! Who cares? http://w3.tue.nl/nl/diensten/bib/over/minisymposium/. Accessed Sept 2009
38. Piwowar, H.A., Day, R.S., Fridsma, D.B.: Sharing detailed research data is associated with increased citation rate. PLoS ONE **2**(3), e308+ (2007)
39. Rensink, A., Taentzer, G.: AGTiVE 2007 graph transformation tool contest. In: Schürr, et al. (eds.) Lecture Notes in Computer Science, vol. 5088. Springer, New York (2008)
40. Schürr, A., Nagl, M., Zündorf, A. (eds.): Applications of Graph Transformations with Industrial Relevance, Third International Symposium, AGTiVE 2007, Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 5088. Springer, New York (2008)
41. Stardock Corporation. Fences. http://en.wikipedia.org/wiki/Fences_(software). Accessed Mar 2010
42. Steffen, B., Margaria, T., Nagel, R., Jörges, S., Kubczak, C.: Model-driven development with the jABC. In: Bin, E., Ziv, A., Ur, S. (eds.) Haifa Verification Conference. Lecture Notes in Computer Science, vol. 4383, pp. 92–108. Springer, New York (2006)
43. Stodden, V.: The legal framework for reproducible scientific research: licensing and copyright. Comput. Sci. Eng. **11**(1), 35–40 (2009)
44. R. D. S. T. S. Team: Top 10 RDP Protocol Misconceptions—part 2. http://blogs.msdn.com/rds/archive/2009/03/12/top-10-rdp-protocol-misconceptions-part-2.aspx. Accessed Mar 2009
45. van den Brand, M.: Guest editor's introduction: experimental software and toolkits (EST). Sci. Comput. Program. **69**(1–3):1–2. Special issue on Exp. Softw. Toolkits (2007)
46. Van Gorp, P.: SHARE image with CPNtools and behavioral models of SHARE. http://is.ieis.tue.nl/staff/pvgorp/share/?page=ConfigureNewSession&vdi=ModelsOfSHARE_v1.vdi. Accessed Feb 2009
47. Van Gorp, P.: VRDP connection count bug? (wrong value of mcVRDPC lients). http://www.virtualbox.org/ticket/4655. Accessed July 2009
48. Van Gorp, P.: SHARE group with basic operating system images http://is.ieis.tue.nl/staff/pvgorp/share/?page=Signup&bundlename=Operating%20Systems. Accessed Mar 2010
49. Van Gorp, P., Blom, S., Belinfante, A.: SHARE–Sharing Hosted Autonomous Research Environments. http://is.ieis.tue.nl/staff/pvgorp/share/ (2009)
50. Van Gorp, P., Blom, S., Belinfante, A.: SHARE documentation. http://fmt.cs.utwente.nl/redmine/wiki/5/SHARE. Accessed Mar 2010
51. Van Gorp, P., Keller, A., Janssens, D. : Transformation language integration based on profiles and higher order transformations. In: Gasevic, D, Lämmel, R., Wyk, E.V. (eds.) SLE. Lecture Notes in Computer Science, vol. 5452, pp. 208–226. Springer-Verlag, New York (2008)
52. Van Gorp, P., Rensink, A.: Call for papers: STTT special section on Graph-Based tool comparison. http://www.fots.ua.ac.be/events/grabats2008/sttt-section-cfp.pdf. Accessed Nov 2008
53. Vandewalle, P., Kovacevic, J., Vetterli, M.: Reproducible research in signal processing—what, why, and how. IEEE Signal Process. Mag. **26**(3), 37–47 (2009)
54. Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commun. Rev. **39**(1), 50–55 (2009)
55. Varro, D., Asztalos, M., Bisztray, D., Boronat, A., Geiss, R., Gogolla, M., Gorp, P.V., Kniemeyer, O., Narayanan, A., Rencis, E., Weinell, E.: Graph transformation tools contest on the transformation of UML models to CSP. In: Schürr, et al. (eds.) Lecture Notes in Computer Science, vol. 5088. Springer, New York (2008)
56. VMware. VMware ESXi. http://www.vmware.com/. Accessed Mar 2010
57. Wang, L., von Laszewski, G., Tao, J., Kunze, M.: Grid virtualization engine: design, implementation and evaluation. IEEE Syst. J. **3**(4), 477–488 (2009)
58. Wikipedia. Hwang woo-suk-wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Hwang_Woo-Suk (2006)
59. Winter, R., Fischer, R.: Essential layers, artifacts, and dependencies of enterprise architecture. In: EDOCW '06: Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops, p. 30, Washington, DC, USA. IEEE Computer Society (2006)
60. Zhu, H.: A framework for service-oriented testing of web services. Comput. Softw. Appl. Conf. Annu. Int. **2**, 145–150 (2006)

## Author Biographies

**Pieter Van Gorp** is investigating the applicability of graph transformation to standard compliant model-driven engineering since 2002. His research interests include model transformation, business process modeling and reproducible research. To support this research, he has contributed to the MoTMoT tool and more recently to SHARE. He also teaches courses on modeling, transformation and simulation, supported by tools such as AToM$^3$, GrGen.NET and CPN-tools. Van Gorp has participated in the organization of national and international workshops such as OCL 2008, various editions of the Transformation Tools Contest (GraBaTs and TTC), and MHPW 2010. He is a reviewer for various international conferences and journals. Since 2008, Van Gorp is an assistant professor in the School of Industrial Engineering at Eindhoven University of Technology. Previously he held a postdoc position at the University of Antwerp, where he also obtained his Ph.D. degree.

**Paul Grefen** is a full professor in the School of Industrial Engineering at Eindhoven University of Technology since 2003, where he chairs the Information Systems subdepartment. He received his Ph.D. in 1992 from the University of Twente. From 1992 until early 2003, he held assistant and associate professor positions in the Computer Science Department at the University of Twente. He was a visiting researcher at Stanford University in 1994. He has been involved in various European research projects as well as various projects within the Netherlands. He is a member of the editorial boards of the International Journal of Cooperative Information Systems and the International Journal of Service Oriented Computing and Applications. He is editor of the books on the WIDE and CrossWork projects, and has published books on workflow management and e-business. He is a member of the Executive Board of the European Supply Chain Forum. His current research interests include inter-organizational workflow management, architectural design of business information systems, and high-level transaction and contract management in electronic business.