RealPeer – A Framework for Simulation-based Development of Peer-to-Peer Systems

Dieter Hildebrandt, Ludger Bischofs
OFFIS Institute for Information Technology
Business Information Management
Escherweg 2, 26121 Oldenburg, Germany
{dieter.hildebrandt, ludger.bischofs}@offis.de

Wilhelm Hasselbring
University of Oldenburg
Software Engineering Group
26111 Oldenburg, Germany
hasselbring@informatik.uni-oldenburg.de

Abstract

In the process of developing P2P systems simulation has proved to be an essential tool for the evaluation of existing and conceived P2P systems. So far, in practice there has been a clear separation between a simulation model of a P2P system and a real P2P system that operates on a real physical network. This separation hinders the transition of models to real systems and the evaluation of already deployed systems by means of simulation.

This work tries to bridge this gap by proposing a methodology and a framework for the simulation-based development of P2P systems. In our approach, an initial model of a P2P system is iteratively transformed into the intended real P2P system. The presented framework effectively supports a developer in modelling, simulating and ultimately developing P2P systems. We demonstrate the validity of our approach and the framework by constructing an example P2P application. This application is simulated in a series of experiments as well as deployed in a large-scale internet based P2P system.

1. Introduction

The concept of direct communication between equal entities is not new but currently relives a renaissance under the notion of Peer-to-Peer (P2P). P2P systems are distributed systems in which units of equal rights and capabilities communicate directly or indirectly with each other. When system designers employ the P2P model in lieu of the client-server model, the expected benefits are amongst others improved scalability and adaptability, decreased costs of operation and the aggregation of idle resources [16].

Evaluation and testing is a vital element in the development process of any system and P2P systems in particular. A developer performs these activities amongst others to compare alternatives within the design space, to detect deficient properties of a system before the expensive step of system deployment and to measure properties of a system once it is deployed. However, evaluating P2P systems in a realistic environment is often not feasible, especially if they are required to scale to a large number (thousands or even millions) of peers. P2P systems are complex systems. The assumption that "most real-world systems are too complex to allow realistic models to be evaluated analytically" [13] applies to P2P systems. Thus, simulation established itself as a preferred method for the evaluation of P2P systems.

So far, in practice there has been a clear separation between the simulation model of a P2P system and a real P2P system that operates on a real physical network. Simulation models are simplified abstractions of real systems that are formalised with the language and modelling concepts that a particular simulation paradigm and environment offers. In contrast, the functionality of real P2P systems is usually provided by P2P applications that are formalised by means of application programming and that include every detail of the intended system.

This separation hinders the transition of a model of a P2P system to a real P2P system and vice versa. Typically, once a system designer is done evaluating a system's properties using a simulation model, he has to build a P2P application from scratch that implements the functionality of the system. Additionally, once a P2P application is deployed, it is difficult to assess the properties of the P2P system as a whole. In order to evaluate the system by means of simulation, a model has to be created which captures the characteristics of the system.

In this paper, we propose a methodology and framework for the *simulation-based development of P2P systems*. The intention is to ease the development and evaluation of P2P systems that are based on new architectural styles [6]. In our approach, within an iterative, evolutionary development process an initial model of a P2P system is transformed step-

wise into the intended real P2P system. Each iteration of the model can be evaluated and tested by simulation. We consider a *P2P application* as the central element of both a model of a P2P system and a real P2P system. We propose a method for implementing P2P applications which enables them to be used as a model and as a real application at the same time. Thus, there is no separation in representation between a model and a real system in the central elements.

In order to support a developer in the process of the simulation-based development of P2P systems, we built a reusable tool: the REALPEER framework [10]. This open source, platform independent, object-oriented software framework was designed to meet the following basic requirements:

1. The framework must support the *modelling and simulation* as well as the *development* of P2P systems.
2. It should be as *generic* as feasible with regard to the class of P2P systems that can be modelled, simulated and developed using the framework.
3. The framework must offer one single *representation* for a P2P application that permits that application to be *reused* both as a simulation model and as part of a real P2P system.
4. The framework's architecture must be highly *modular* and *extensible* with a clear separation of concerns. This enables a developer to *combine* and freely exchange elements of the framework and the model and thus provides a mechanism to reuse elements.
5. In order to support the simulation of large-scale P2P systems, the framework should be as *scalable* and lightweight as possible.
6. The framework must enable a developer to conduct *controlled simulation experiments* with complete *internal validity* in order to obtain accurate and reproducible simulation results.

The benefits of utilizing the proposed methodology and framework are in the first instance: *reusability* and *comparability*. A model of a P2P system can be reused in a real P2P system and vice versa. Individual elements of the framework and the model can be reused and combined to gain valuable opportunities to evaluate P2P systems (see Section 3.3). The framework and the integrated simulator are reusable. Different P2P systems, design alternatives of the same system and simulation results are easier to compare if they are based on the same framework.

The remainder of the paper is organised as follows. Section 2 briefly reviews related work. Section 3 presents our approach to simulation-based development of P2P systems. Section 4 gives an overview of the REALPEER framework. Section 5 presents evaluation results. Finally, Section 6 concludes the paper and outlines future work.

2. Related Work

To the best of our knowledge, there is no dedicated methodology or tool that supports the simulation-based development of P2P systems as outlined in the introduction. Hence, we examined work that comes closest to our effort:

Network Simulators and P2P Simulators We have reviewed several network simulators (e. g. NS-2 [3] and SSFNet [2]) and more than 30 P2P simulators [11], including PeerSim [17], Omnix/Simix [12] and Darlagiannis et al. [9]. We assessed the simulators against the aforementioned requirements. All simulators but Omnix/Simix fail to allow the reuse of a model as part of a real P2P system. Moreover, Omnix/Simix fails in conducting controlled experiments.

Development tools allowing to reuse the same code in simulations and on real networks Neko [21] is a Java platform that allows the same distributed algorithm to execute both within a simulation and on a real network. Since Neko is tailored towards distributed algorithms in general, it does not adequately support some peculiarities of P2P systems. For example, it is conceived for a rather small amount of nodes and does not consider peer dynamics and discovery. MACEDON [19] is an infrastructure to ease the design, development and evaluation of overlay networks. Because distributed algorithms are specified in a domain-specific language rather than by extending an object-oriented framework, it follows a completely different approach. Moreover, MACEDON is limited to distributed hash tables and application-level multicast. WiDS [14] is an integrated toolkit for the development of distributed systems. REALPEER differs from this effort by focusing on a highly extensible framework that allows a developer to combine and freely exchange any element of the model. Furthermore, it presents a domain model for P2P systems that goes beyond the modelling of P2P protocols. Because WiDS was implemented in C++, it is expected to be platform dependent to some extent. It is not publicly available. Moreover, all presented efforts operate on the message-level. They do not allow to specify the binary layout of protocol messages (limiting interoperability) and do not allow the transfer of large amounts of data between peers (e. g. file transfers).

Reference Architectures for P2P Systems We examined suggestions for reference architectures for P2P systems to determine if one is suitable for representing a P2P system within the REALPEER framework: Aberer et al. [5], Dabek et al. [8], JXTA [1] and Melville et al. [15]. Unfortunately, none of them is at the same time commonly accepted, concrete enough to be of practical use and generic with regard to the class of supported P2P systems. Consequently, we

employed none of the suggestions.

3. Simulation-Based Development of Peer-to-Peer Systems

We consider simulation as an essential, integral element in the development of P2P systems. In our approach to simulation-based development of P2P systems, a developer iteratively transforms an initial simulation model into the intended real system. A tool that supports a developer in this process has to offer facilities to model, represent and execute both a model of a P2P system and parts of a real P2P system.

In Section 3.1, we compare the concepts of a model and a real system. Our goal is to determine what basic properties a combined representation for these concepts has to exhibit in order to meet the requirements. In Section 3.2, we propose a *domain model for P2P systems* that serves three purposes. First, it defines how a model of a P2P system and parts of a real P2P system are represented within the framework in a combined manner. Furthermore, it constrains the notion of "generic" as it applies to the framework. Finally, it constitutes the interface of the framework to a developer that extends the domain model to build concrete, executable P2P systems. In Section 3.3, we elaborate on the methodology and the methodical application of our domain model for P2P systems in the process of simulation-based development of P2P systems.

3.1. Model vs. Real System

In order to be able to compare the concepts of a model of a P2P system and a real P2P system, we first have to clarify how each of these concepts is represented separately. A model of a P2P system is appropriately represented by a *discrete-event simulation model* [13]. This class of models is discrete, dynamic and stochastic. It is to be executed by a simulator and can be represented by code. Practically every P2P simulator we reviewed applied discrete-event simulation. In contrast, we consider a *P2P application* as the central element of a real P2P system that provides the functionality of a P2P system.

We identified four relevant, distinctive characteristics in which models of P2P systems and real P2P systems differ:

Degree of Abstraction A model of a system is commonly defined as a system that is an abstracted and simplified representation of the original (real) system. Since a model is defined as a system, in principle a real system and the model of that real system can have the same kind of formal representation. This applies when both are represented as code. In this case, we assume that models of a real system can be ordered according to the degree of abstraction. They can be transformed into the real system by a sequence of transformations.

Formal Language Discrete-event models are formalised with modelling concepts like entities, processes, events, queues and so forth [13]. In contrast, P2P applications are formalised by means of application programming utilising a common programming language.

System Time We define system time as the time continuum that a system is based on and in which it unfolds its dynamics. Models and real systems differ in the way they map system time to real-time. In real systems, this mapping is the identical mapping. In simulation models, this mapping is complex. The execution of the model can take less (or more) time than the execution of the original system.

Control of System Time and Execution In discrete-event simulation, system time and the execution of the model are under strict control of a *scheduler* component. The scheduler advances the system time to the next occurrence of an event and executes the event logic that may change the model's state. The execution of the model is strictly sequential with only one control flow. Real P2P systems and P2P applications exhibit a lot of parallelism. System time and execution are under no central control.

We now reason about desirable properties of a combined representation for artefacts that are to be used both as a model of a P2P system and a real P2P system.

- As we have pointed out there is no conflict in the degree of abstraction.
- Regarding the formal language, artefacts must be represented by code that is formalised by means of application programming utilising a common programming language. This is derived from the goal to ultimately develop P2P applications as the central elements of real P2P systems.
- Artefacts must be executed in both simulated time and real-time. The capability to execute a model in simulated time (e. g. faster than the original system) is of considerable practical use. Of course, real systems must execute in real-time.
- System time and the execution of the code must be under central control of a scheduler. This is necessary in order to ensure that the artefacts provide complete internal validity when they are executed in the role of a simulation model. Thus, every change in the simulation results can be attributed to explicit changes in the model and input variables to the experiment.

In summary, an artefact should be developed like a P2P application and at the same time simulated like a discrete-event simulation model.

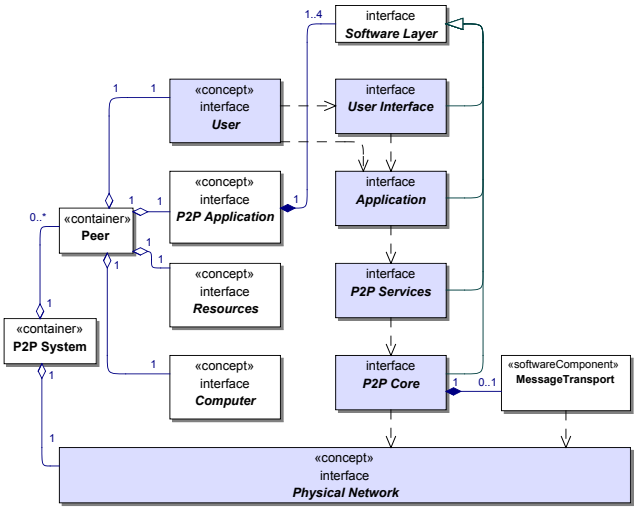

Figure 1. Domain Model for P2P Systems

3.2. Domain Model for Peer-to-Peer Systems

As we have already indicated in Section 2, no suitable reference architecture for P2P systems exists that can be employed for representing a P2P system within our framework. Hence, we created our own domain model for P2P systems that is based on the related work we have reviewed. The proposed model is intentionally basic and coarse. It is not our goal to define a detailed reference architecture for P2P systems. In addition, because no requirements exist besides those stated in the introduction, further refinements seem arbitrarily. In consequence, the domain model captures what appears to be common for a wide range of P2P systems and is open for extensions by developers.

Figure 1 gives an overview of the static structure of the proposed domain model. A *P2P system* is an aggregate that consists of a set of peers and a *Physical Network*. A *Peer* consists of a *User*, a *P2P Application*, *Resources* and a *Computer*. Classes labelled with the stereotype ≪*concept*≫ provide an abstraction layer between the concept they stand for and clients of that concept. Concrete instances of these concepts depend on whether the represented P2P system is to be used in the role of a model or a real system. In the role of a model, these classes encapsulate models of their concepts. In the role of a real system, they encapsulate real occurrences of their concepts. The class *P2P Application* is an exception: Instances of this class can be used in both roles. In the following, we briefly characterise the main elements of the domain model.

P2P Application A P2P application is divided into four layers: *P2P Core* (essential functionality like maintenance of the overlay network, routing, resource location), *P2P Services* (common services like resource management), *Application* (functionality that is specific to the application domain) and *User Interface* (providing an interface to a human user). The layers are not further divided into functional components. The class *MessageTransport* is an exception: This component handles the common task of sending and receiving protocol messages over the physical network.

User This class represents the human user of a P2P application. In the role of a model, it encapsulates the behaviour of a user, i. e. how a user is acting on the other constituents of a peer (P2P application, resources and computer) over time. In the role of a real system, this class encapsulates a real human using the user interface of the P2P application.

Resources This class models the resources that a peer offers to the P2P system.

Computer The class *Computer* models the computer that runs the P2P application.

Physical Network This class models the physical network that peers use to communicate. It provides a common network access interface that is suitable to encapsulate a wide range of network types (TCP/IP, Bluetooth etc.). The interface is based on the socket abstraction and was designed to resemble the socket interface of the Java class library as closely as possible. The concrete network type is determined by the implementation of this interface. In the role of a model, this class models a physical network with the desired level of detail. In the role of a real system, it acts as an adapter to a network interface that is provided by the operating system, virtual machine or third parties.

3.3. Transition from Model to Real System

In our approach to simulation-based development of P2P systems within an iterative, evolutionary development process, a developer stepwise refines an initial model of a P2P system into the intended real P2P system. A fundamental aspect of this approach is the coupling of the degree of abstraction of the artefact under development with the development time. This method of developing P2P systems based on simulation models has to be embedded in a process model. In the area of software engineering, process models for the iterative and evolutionary development have been known for a long time [20]. In the area of modelling and simulation, process models exist that recommend the iterative development of models [13]. In this paper, we focus on the basic method and leave the elaboration of a combined process model to be addressed in future work.

Figure 2 depicts the ideal-typical use of the domain model for the simulation-based development of P2P systems. Initially, for each layer of the domain model a developer creates a simple model of the corresponding concept (left). Over time, these models are refined until they correspond to the intended real P2P system at the end of the de-

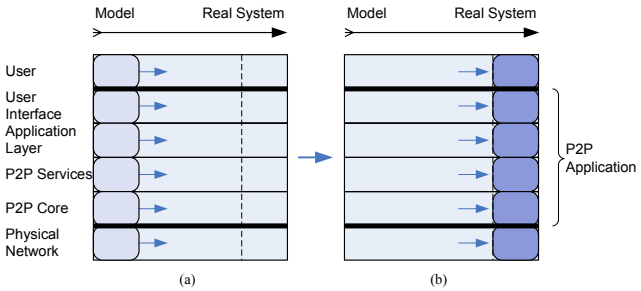

Figure 2. Ideal-typical Use

velopment process (right). The last iteration (separated by a dashed line) is a special case: In this discrete step, each element of the model that has no combined representation is replaced by its real occurrence. More precisely, models of the user and the physical network as well as models of resources and the computer (not displayed) are replaced.

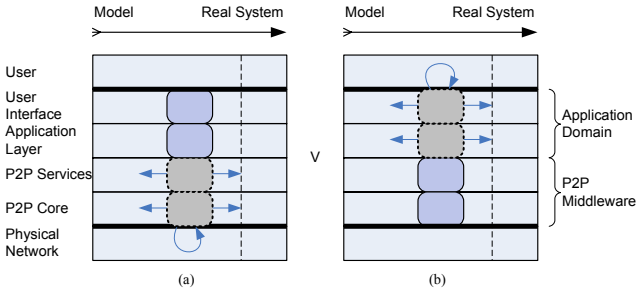

Figure 3. Freely Combining Model Elements

Figure 3 illustrates a different use of the domain model. A P2P application can be partitioned into a part that is specific to the application domain (layers User Interface and Application) and the P2P middleware (layers P2P Services and P2P Core). Provided that a developer establishes a fixed interface for the P2P middleware, these parts can be developed and exchanged independently. For example, a file-sharing application and an instant messenger application could both be combined with a P2P middleware that is based on a structured or unstructured P2P system respectively. The domain model (and REALPEER framework) supports a developer in freely combining elements of the model.

Figure 4 summarises the different options to execute a P2P application in the role of a model. We distinguish four cases:

Case (a): Each layer is represented by a model of its corresponding concept (the respective areas of the layers are indicated by a darker shade). The model constitutes a complete P2P system with a specified number of peers. It is executed on a single computer in simulated time.

Case (b): Several P2P applications are executed in simulated time or real-time on a single computer. The applications are controlled by one or more human users and operate on a model of a physical network. As a practical example, for every peer of a small overlay network a graphical user interface (implemented by the User Interface layer) could be presented. A human user utilises these interfaces to monitor and control several peers at the same time.

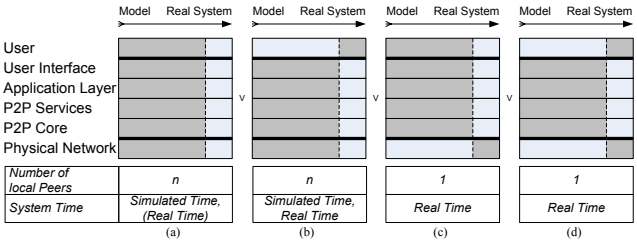

Figure 4. P2P App. in the Role of a Model

Case (c): In this case, a single P2P application is executed on a computer in real-time and connects itself via a real physical network with an existing overlay network. A user model is controlling the P2P application. Thus, the control of the application is automated by a user model. A peer of this kind could connect to an existing overlay network in order to measure properties of its own performance and of the real P2P system.

Case (d): A human user controls the model of a P2P application that operates on a real physical network in real-time. Thus, it is possible to execute and evaluate P2P applications that are not yet fully developed and still are labelled as "model".

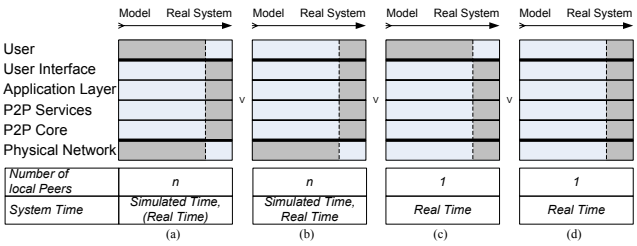

Figure 5. P2P App. in the Role of Real System

Figure 5 summarises the different options to execute a P2P application in the role of a real system. The only difference to the four cases just presented is that a fully developed P2P application is executed. This illustrates the possibility to evaluate and test P2P applications and P2P systems after their development has officially ended.

In the usage scenarios presented in Figure 4 and 5, the borders between a model and a real system are relaxed. All but two cases combine models of concepts with real occurrences of concepts to gain valuable opportunities to evaluate a P2P system.

4. The RealPeer Framework

The REALPEER framework is a reusable tool we built in order to support a developer in the simulation-based development of P2P systems as described in the preceding section. It was designed to meet the set of requirements stated in the introduction. The framework uses Java as the implementation and simulation language.

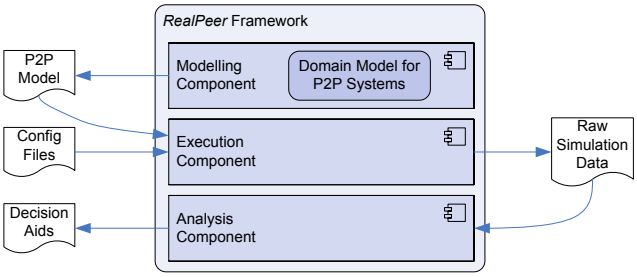

Figure 6. Components of the Framework

Figure 6 depicts the functional components of the framework on the highest, conceptual level. A developer employs the *Modelling Component* to develop a simulation model of a P2P system. In the last iteration of the development process, the P2P application constitutes the part of the intended real P2P system that can be represented within the framework. The domain specific part of the framework is concentrated in the *Domain Model for P2P systems*. In the shape of a clearly separated unit, it is embedded in the Modelling Component. A developer extends the domain model to build concrete, executable P2P systems. The *Execution Component* is used to execute both the model of a P2P system and a real P2P system. The *Analysis Component* aids a developer in statistically analysing and graphically presenting data that is acquired and collected while executing a P2P system within the framework. At present, the Analysis Component relies heavily on the reuse of external tools (e. g. SPSS and Microsoft Excel).

Fundamental Design Decisions Designing the tool as an *object-oriented software framework* seemed an obvious choice since ultimately the tool must support a developer in implementing a P2P application represented by code. All elements of the domain model for P2P systems and the most important elements of the framework core (e. g. the scheduler) are represented by hot-spots. A developer extends the framework by developing *plug-ins* for the given hot-spots. In this context, the use of the plug-in design pattern enables the framework to be used as a black-box framework (reuse through composition instead of inheritance), encourages freely combining and thus reusing elements of the framework and allows for data-driven configuration (the selection and configuration of the plug-ins is controlled by configuration files). For the execution of the model we designed a *control model* that allowed a central scheduler to control the system time and execution of the model and at the same time offers techniques used in application programming. The result is a synthesis of event-driven programming (from the area of application programming) and the event-scheduling modelling style [13] (from the area of discrete-event simulation). Applying this control model a developer cannot use multithreading when developing models (the control flow must be sequential and controlled by the framework's scheduler). Hence, a developer has to imitate the functionality of threads by the use of periodic self-activation through the scheduler and callbacks.

Architecture Overview Figure 7 gives an overview of the static structure of the framework's architecture. The package *core* constitutes the domain independent core of the framework. A simple command line interface is contained in the package *ui*. The task of the package *execution* is the execution of a model by a scheduler in simulated or real time. The package *model* provides abstract modelling concepts (e. g. *entity* and *event*). *observer* provides a mechanism to collect simulation data while executing a model. The framework defines a set of *ObserverEvents* that encapsulate different types of simulation data. They are produced by a model and consumed by registered *Observer* plug-ins that export the data to external analysis tools. Thus, data sources and consumers are decoupled and existent *Observers* can be reused for different models. The package *domain* contains the presented domain model for P2P systems that has been divided into the parts "P2P" and "physical network" for improved reusability. The package *plugins* contains reusable, ready to use plug-ins that are delivered with the framework. Currently, this package contains two *Schedulers* (simulated and real time), two *MessageTransports* (optimized without message serialisation and fully fea-

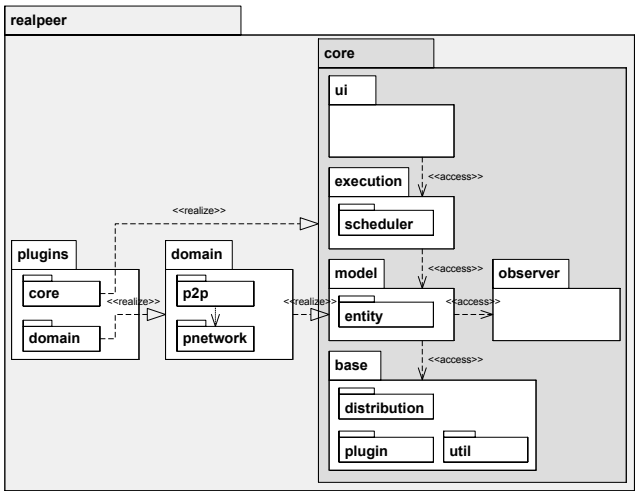

Figure 7. Package Overview

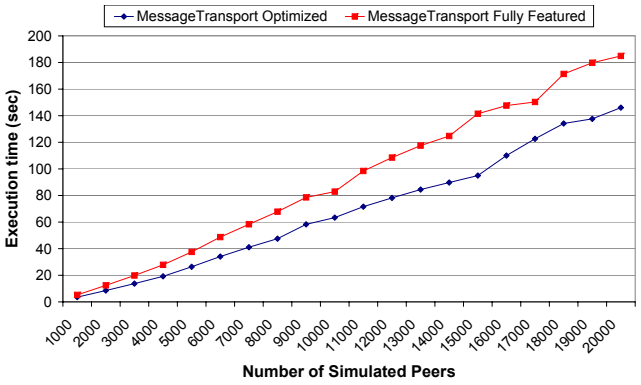

Figure 8. Execution Time

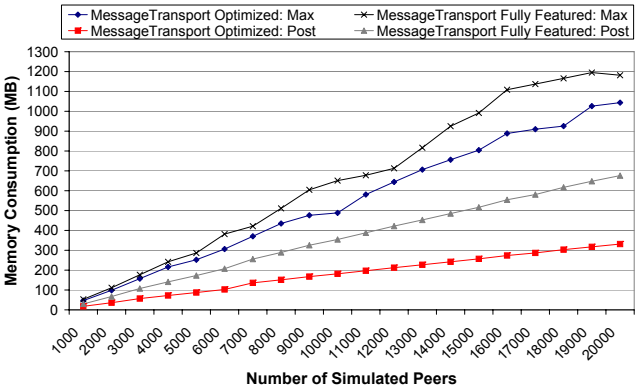

Figure 9. Memory Consumption

tured) and two *PhysicalNetworks* (minimal TCP/IP Internet model and adapter to real TCP/IP Internet).

5. Evaluation

We implemented a model of a P2P system for the purpose of evaluation using the proposed methodology and the REALPEER framework: Gnutella [4] (protocol version 0.4). In order to validate the Gnutella model and the simulator, in a series of simulation experiments we successfully reproduced simulation results presented in the literature [18].

Then we aimed at gaining evidences of how much resources are consumed by the model and the simulator when executing the model and of how scalable the simulator can be regarding the number of simulated peers. For these purposes, we implemented a simple experiment set-up (1.000 peers, 1.500 connections, 2.500 resources replicated 25 times, 300 search queries). In a series of 20 simulation runs, we scaled this set-up with factors from 1 to 20. The experiments where performed on a PC with a Pentium4 at 2.6 GHz and 1.5 GB memory. The employed JVM was JDK 5.0.

Figures 8 and 9 provide graphs that demonstrate the consumption of processing time and memory. In the experiments, the performance of two different MessageTransport plug-ins was measured (optimized and fully featured). Memory consumption was measured in two ways: peak consumption without garbage collection ("max") and consumption with garbage collection at the end of the run ("post"). Evidently, both the processing time and memory consumption increase linearly with respect to the size of the model. Profiling revealed that the performance depends significantly on the model's properties. The framework's footprint is very small. An overlay network consisting of 20.000 peers was simulated in approximately three minutes. Memory consumption seems to be the major bottleneck when performing experiments: The simulation of a network with 20.000 peers consumed up to 1.2 GB of memory. We expect that as models evolve to detailed real systems the number of peers that can be simulated on a single machine is decreasing fast.

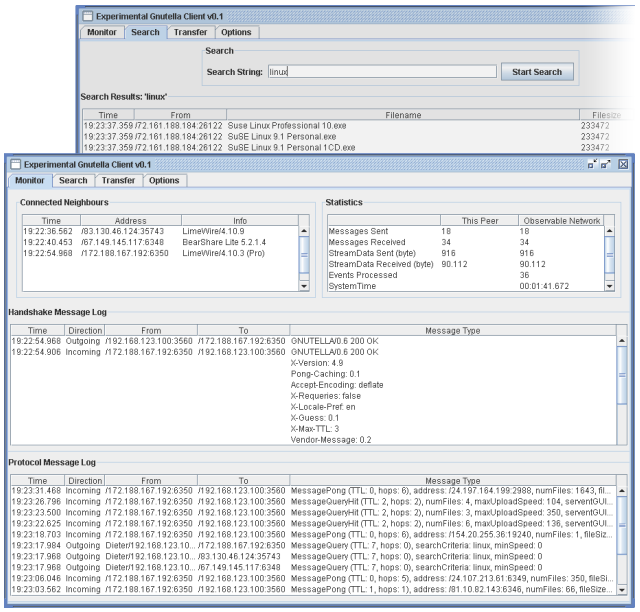

Figure 10. Screenshots of the Gnutella Application connected to a real Gnutella Network

In another experiment, the experiment set-up was modified. The user model (class *User* of the domain model) and the model of the physical network (class *PhysicalNetwork)* were replaced by real occurrences: A real human controls a P2P application that operates on the real Internet. Additionally, a GUI was activated (class *UserInterface*). Figure 10 depicts screenshots of the Gnutella P2P application connected to a real Gnutella network on the Internet executing within the framework. This effectively demonstrates the reuse of a model of a P2P system as part of a real P2P system.

Finally, in the last experiment again we used the set-up of the first experiment as a base. The user model was removed. The GUI was activated and set up to execute exactly one time. The effect was that an overlay network consisting of 1.000 peers was simulated on a single computer. A human user was enabled to control a single simulated peer with the provided GUI. Depending on the employed scheduler, the P2P system could be simulated in simulated or real time. This effectively demonstrates the valuable opportunities to evaluate P2P systems the framework offers by enabling a developer to freely combine different elements of the model.

6. Summary and Future Work

In this paper, we presented a methodology and the REALPEER framework for the simulation-based development of P2P systems.

In order to be able to iteratively transform an initial model of a P2P system into a real P2P system, we had to derive a combined representation for both concepts. Then we presented a domain model for P2P systems that defines how a model of a P2P system and parts of a real P2P system are represented within the framework. A set of four usage scenarios were presented that illustrated the methodical application of the domain model in the process of simulation-based development. Furthermore, they demonstrated the novel and valuable opportunities to evaluate P2P systems the framework offers. Fundamental design decisions and the overall architecture of the REALPEER framework were presented. Finally, we demonstrated the validity of our approach and the framework by a series of experiments. We showed that a model we constructed could be used as a simulation model and as part of a real system at the same time.

Our future work targets the construction of different models, the utilisation of parallel and distributed simulation to overcome the scalability limits and to fully develop a process model for the simulation-based development of P2P systems. Currently, the REALPEER framework is used to implement and evaluate an organisation-oriented P2P system [7] that optimizes the efficiency of P2P search methods by using two P2P overlay networks. The simulation performance already has been improved by deploying Grid middleware.

References

[1] JXTA. http://www.jxta.org/.

[2] Scalable Simulation Framework. http://www.ssfnet.org/.

[3] The Network Simulator – ns-2. http://www.isi.edu/nsnam/ns/.

[4] The Annotated Gnutella Protocol Specification v0.4. http://rfc-gnutella.sourceforge.net/developer/stable/, 2003.

[5] K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth. The essence of P2P: A reference architecture for overlay networks. In *Fifth IEEE Intl. Conference on P2P Computing*, Germany, Sept. 2005.

[6] L. Bischofs, S. Giesecke, M. Gottschalk, W. Hasselbring, T. Warns, and S. Willer. Comparative Evaluation of Dependability Characteristics for Peer-to-Peer Architectural Styles by Simulation. *Journal of Systems and Software, Special Issue: Architecting Dependable Systems*, 79, October 2006.

[7] L. Bischofs and U. Steffens. Organisation-oriented Super-Peer Networks for Digital Libraries. In M. Agosti, H.-J. Schek, and C. Türker, editors, *Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures: 6th Thematic Workshop of the EU Network of Excellence DELOS, Cagliari, Italy*, volume 3664 of *LNCS*. Springer, 2005.

[8] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, Feb. 2003.

[9] V. Darlagiannis, A. Mauthe, N. Liebau, and R. Steinmetz. An Adaptable, Role-based Simulator for P2P Networks. In *Proceedings of International Conference on Modeling, Simulation and Visualization Methods, Las Vegas, USA*, 2004.

[10] D. Hildebrandt. The RealPeer Framework. http://sourceforge.net/projects/realpeer/.

[11] D. Hildebrandt. Eine simulationsbasierte Entwicklungsumgebung für Peer-to-Peer-Systeme. Master's thesis, Universität Oldenburg, Mar. 2006.

[12] R. Kurmanowytsch. *Omnix: An Open Peer-to-Peer Middleware Framework*. PhD thesis, TU Wien, Feb. 2004.

[13] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, third edition, 2000.

[14] S. Lin, A. Pan, Z. Zhang, R. Guo, and Z. Guo. WiDS: an Integrated Toolkit for Distributed System Development. In *Proceedings of the 10th USENIX Workshop on Hot Topics in Operation Systems*, June 2005.

[15] L. Melville, J. Walkerdine, and I. Sommerville. D9 - P2P Reference Architectures (Final Version). P2P Architect Project, Deliverable Internal IST-2001-32708, Computing Department, Lancaster University,UK, June 2003.

[16] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. Technical Report HPL-2002-57, HP Laboratories Palo Alto, 8. Mar. 2002.

[17] A. Montresor, G. D. Caro, and P. E. Heegaard. Architecture of the Simulation Environment. Technical Report D11, University of Bologna, Jan. 2004.

[18] A. Oram, editor. *Peer to Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Mar. 2001.

[19] A. Rodriguez, C. Killian, S. Bhat, D. Kostic, and A. Vahdat. MACEDON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks. In *Proceed. of the 1st USENIX Symposium on Networked Systems Design and Implementation*, San Francisco, USA, March 2004.

[20] I. Sommerville. *Software Engineering*. Addison-Wesley, Pearson, seventh edition, 2004.

[21] P. Urbán, X. Défago, and A. Schiper. Neko: A Single Environment to Simulate and Prototype Distributed Algorithms. *Journal of Information Science and Engineering*, 18(6):981–997, November 2002.