Journal of Systems Architecture 54 (2008) 849–860

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Simulation-based development of Peer-to-Peer systems with the RealPeer methodology and framework

Dieter Hildebrandt [a,*], Wilhelm Hasselbring [b]

[a] OFFIS Institute for Information Technology, Business Information Management, Escherweg 2, 26121 Oldenburg, Germany
[b] University of Oldenburg, Software Engineering Group, 26111 Oldenburg, Germany

ARTICLE INFO

Article history:
Received 2 July 2007
Received in revised form 14 November 2007
Accepted 10 January 2008
Available online 14 March 2008

Keywords:
Peer-to-Peer system
Modelling
Simulation
Software development

ABSTRACT

In the process of developing Peer-to-Peer (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.

To bridge this gap, we put forward the idea of simulation-based development of P2P systems. In this approach, an initial simulation model of a P2P system is iteratively transformed into the intended real P2P system. As a concretion of this approach, we propose a methodology and a framework for the simulation-based development of P2P systems. 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 [1].

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" [2] 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 [3]. In our approach, within an iterative, evolutionary development process, an initial model of a P2P system is transformed stepwise into the intended real P2P system.

* Corresponding author.
E-mail addresses: dieter.hildebrandt@offis.de (D. Hildebrandt), hasselbring@informatik.uni-oldenburg.de (W. Hasselbring).

1383-7621/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2008.01.010

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 its respective central element.

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 [4]. This open source, platform independent, object-oriented software framework was designed to meet the following basic requirements:

(i) The framework must support the *modelling and simulation* as well as the *development* of P2P systems.
(ii) 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.
(iii) 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.
(iv) 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.
(v) In order to support the simulation of large-scale P2P systems, the framework should be as *scalable* and lightweight as possible.
(vi) 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 utilising the proposed methodology and framework are in the first instance: *reusability* and *comparability*. Parts of a model of a P2P system can be reused as parts of 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 summarises the paper and Section 7 outlines current and 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 [5] and SSFNet [6]) and more than 30 P2P simulators [7], including PeerSim [8], Omnix/Simix [9], Darlagiannis et al. [10] and OverSim [11]. None of the assessed simulators met the requirements stated in the introduction. For example, most simulators fail to allow the reuse of a model as part of a real P2P system. In this regard, Omnix/Simix and OverSim are exceptions. However, Omnix/Simix fails in conducting controlled experiments whereas OverSim is not fully generic and is confined to the development of P2P protocols (as opposed to complete P2P applications).

Development tools allowing to reuse the same code in simulations and in real systems: In the literature, the benefits of reusing code in P2P simulation models and in real systems already have been recognised (e.g. [12]). Neko [13] 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 [14] 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 [15] 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 specifying 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. [16], Dabek et al. [17], JXTA [18] and Melville et al. [19]. 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* [2]. 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 such as entities, processes, events, queues and so forth [2]. 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

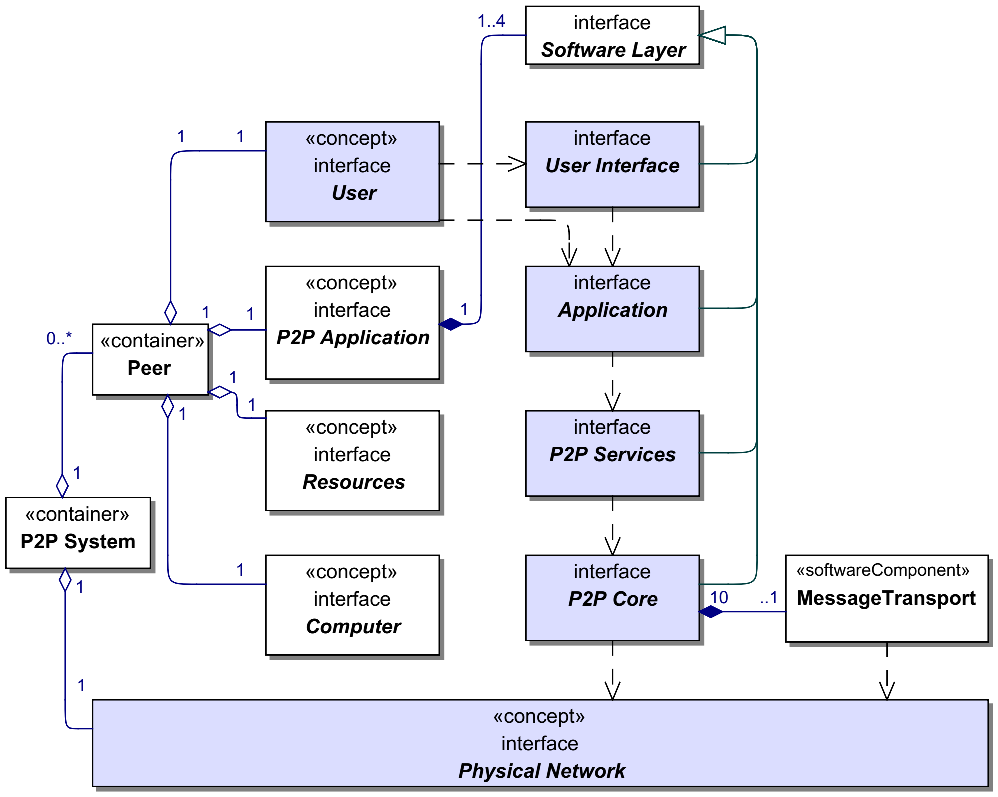


Fig. 1. Domain model for P2P systems.

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.

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.

Fig. 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 such as maintenance of the overlay network, routing, resource location), *P2P Services* (common services such as 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 (see Fig. 2).

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 [21]. In this paper, we focus on the basic method and leave the elaboration of a combined process model to be addressed in future work.

Fig. 3 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 hand side). Over time, these models are refined until they correspond to the intended real P2P system at the end of the development process (right hand side). 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 in the Figure) are replaced.

Fig. 4 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.

Fig. 5 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 (a darker shade indicates the respective areas of the layers). The model constitutes a complete P2P system with a specified number *n* of peers. It is executed on a single computer in simulated time. Execution in real-time is possible but not considered useful.
- *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 (GUI) (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.

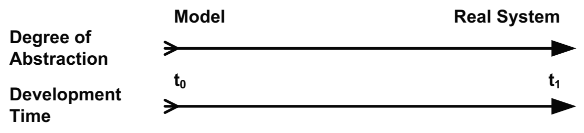


Fig. 2. Coupling of the degree of abstraction of a model of a P2P system with the development time within the iterative, evolutionary development process.

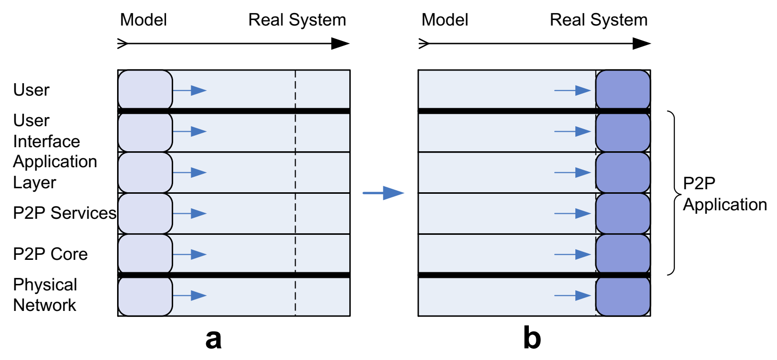


Fig. 3. Ideal–typical use.

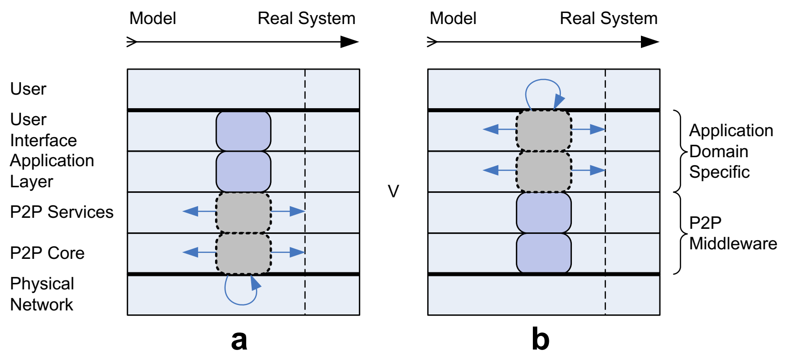
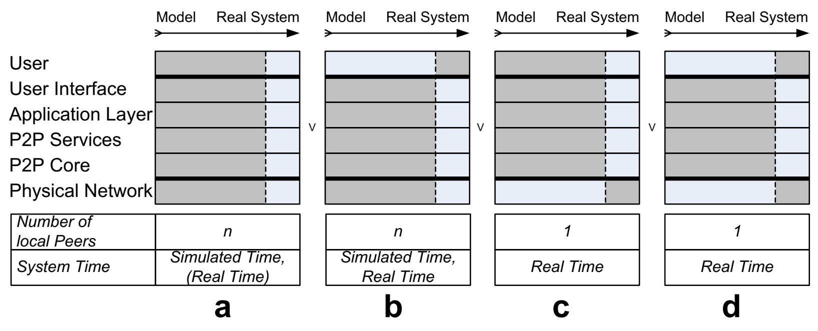
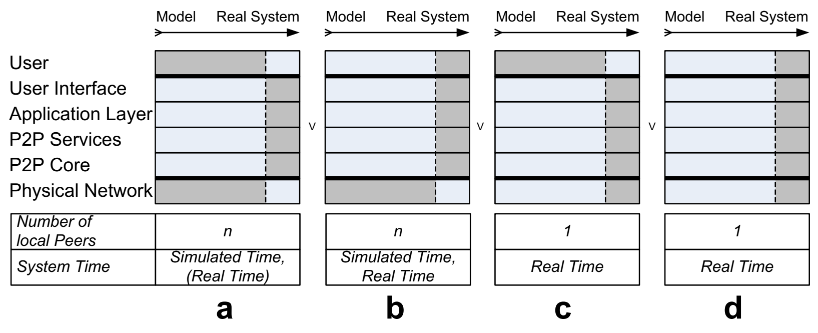


Fig. 4. Freely combining model elements.



	a	b	c	d
Number of local Peers	n	n	1	1
System Time	Simulated Time, (Real Time)	Simulated Time, Real Time	Real Time	Real Time

Fig. 5. P2P application in the role of a model.



	a	b	c	d
Number of local Peers	n	n	1	1
System Time	Simulated Time, (Real Time)	Simulated Time, Real Time	Real Time	Real Time

Fig. 6. P2P application in the role of a real system.

– *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".

Fig. 6 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 Figs. 5 and 6, 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.

Fig. 7 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).

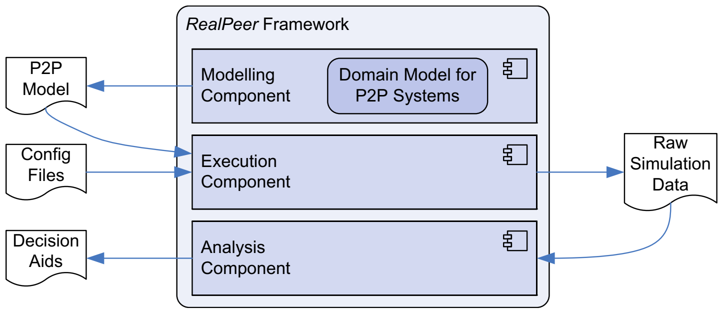


Fig. 7. Components of the framework.

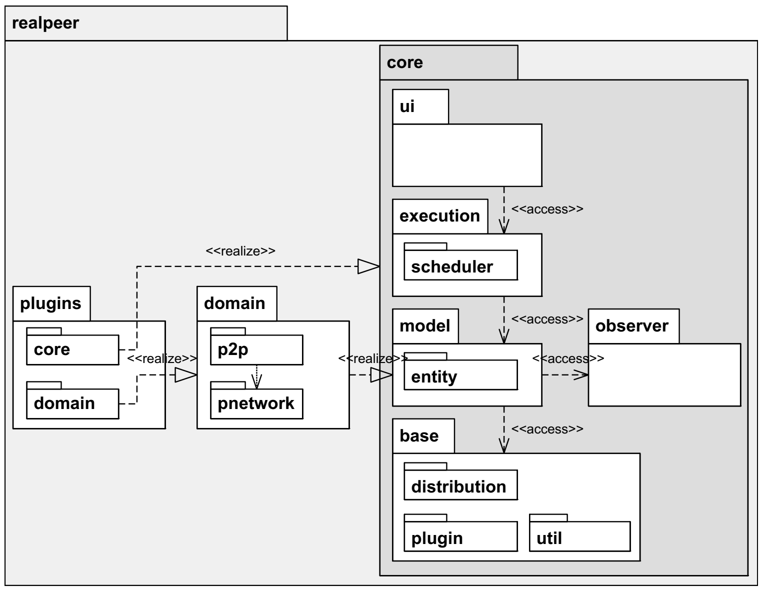


Fig. 8. Package overview.

4.1. 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* [22] 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 [23]), 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 [20] (from the area of application programming) and the event-scheduling modelling style [2] (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.

4.2. Architecture overview

Fig. 8 gives an overview of the static structure of the framework's architecture. On the highest level, the framework is decomposed into the packages *core*, *domain* and *plugins*.

The package *core* constitutes the domain independent core of the framework and offers functionality for modelling and executing models and for collecting simulation data while executing a model. Since the core is domain independent, it is not tied to the modelling and execution of models of a specific domain (like P2P systems). To be of practical use, the core has to be extended by a domain model. For this purpose, the core provides hot-spots that are extended by the plug-ins of a domain model. The core is structured into several contained packages. 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 *base* contains base functionality for the implementation of the core and its extensions, like plug-in management, random number generators and distribution functions.

The package *domain* holds domain models that extend the framework's core. Domain models are developed by domain experts and should be reusable for a wide range of concrete models of the targeted domain. In order to build a concrete, executable model of a domain, a developer extends the hot-spots of a given domain model. Currently, the package 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. Presently, this package contains six basic plug-ins that extend hot-spots of the core and the domain model of P2P systems. The simulated time *Scheduler* processes the events that the model generates as soon as possible. Thus, models are enabled to execute faster than real-time. In contrast, the real-time Scheduler processes events not until their scheduled system time equals real-time. The first *PhysicalNetwork* plug-in is a minimal TCP/IP Internet model with constant latency whereas the second PhysicalNetwork plug-in is an adapter to a real TCP/IP Internet connection. The fully featured *MessageTransport* plug-in uses a provided physical network model for sending and receiving protocol messages whereas the optimised MessageTransport bypasses the physical network model and its overhead and transmits messages directly via the Scheduler.

Finally, in order to build a concrete model of a P2P system, a developer extends the framework by developing plug-ins for the hot-spots of the domain model and by reusing ready to use plug-ins that are provided by the framework.

5. Evaluation

5.1. Evaluation model and experiments

We implemented a model of a P2P system for the purpose of evaluation using the proposed methodology and the RealPeer framework: Gnutella [24] (protocol version 0.4). We chose the Gnutella protocol for the initial evaluation for three reasons. First, the protocol is well documented and exhibits only moderate complexity. Second, at present real Gnutella P2P systems exist on the internet that the evaluation model can connect to. Finally, many publications are available that document characteristics of Gnutella P2P systems that can be utilised for validation. Hence, as a first step, we validated the Gnutella model and the simulator in a series of simulation experiments by reproducing simulation results presented in the literature [25].

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 is 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 and 300 search queries). In a series of 20 simulation runs, we successively 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 running on Windows XP.

Figs. 9–11 provide graphs that demonstrate the consumption of processing time and memory and the volume of generated data for communication. In the experiments, the performance of two different MessageTransport plug-ins was measured (optimised 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 that generated more than 9 million protocol messages and over 330 MB of raw message data was simulated in approximately 3 min. 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.

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*). Fig. 12 depicts screenshots of the Gnutella

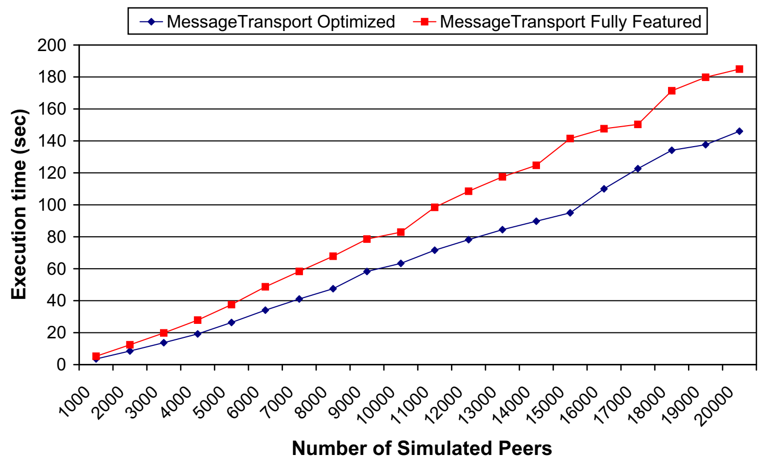


Fig. 9. Execution time.

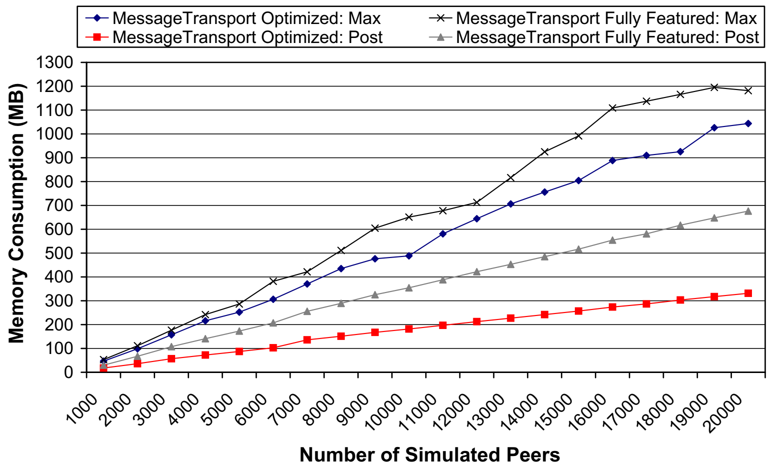


Fig. 10. Memory consumption.

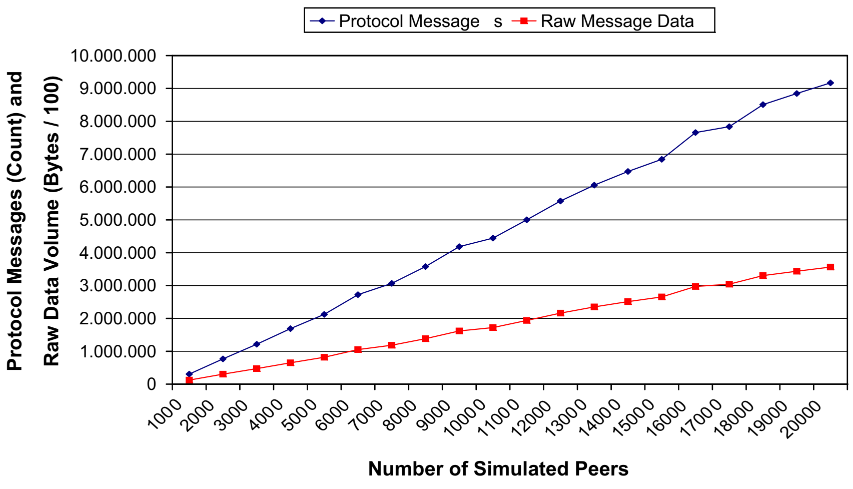


Fig. 11. Message and raw data volume.

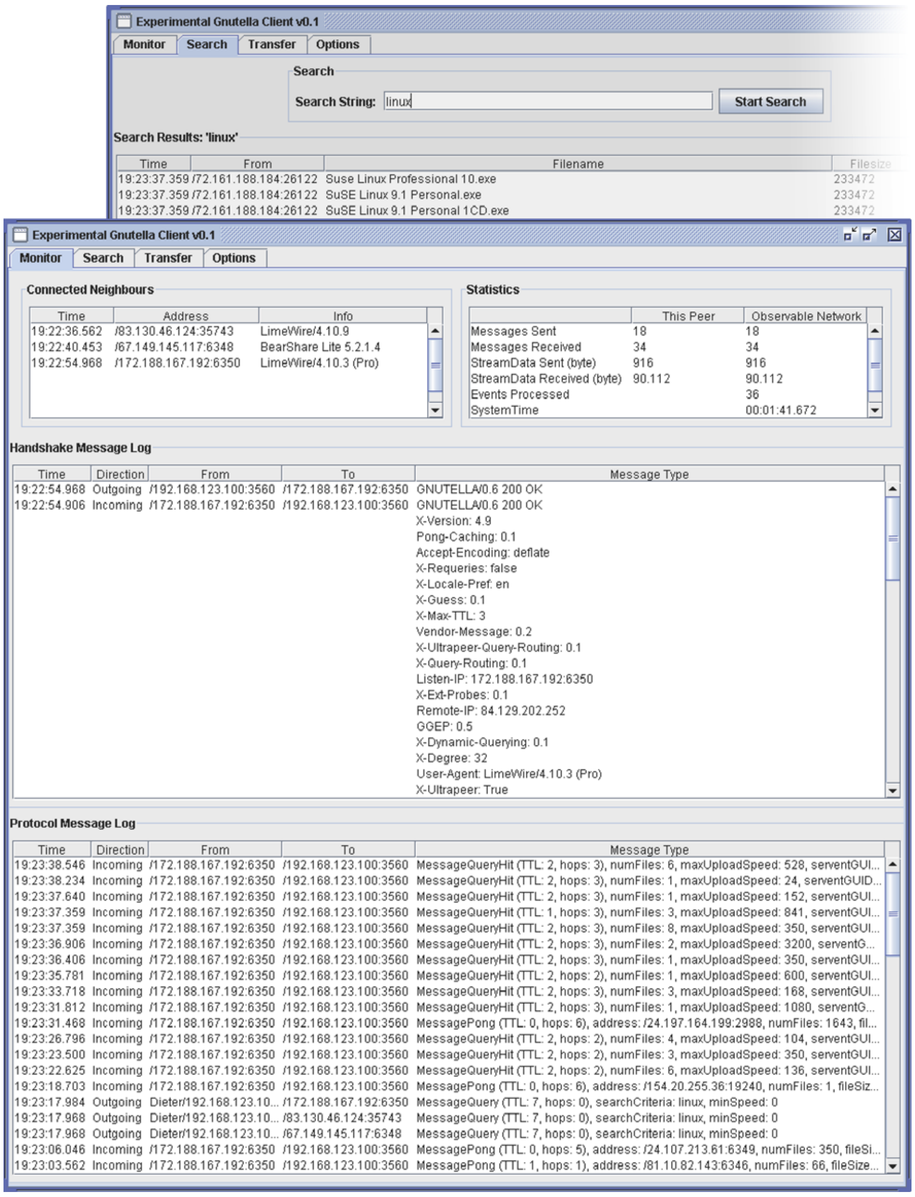
P2P application connected to a real Gnutella network on the Internet executing within the framework. This effectively demonstrates the reuse of a model of a P2P system as part of a real P2P system.

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.

5.2. Discussion

As stated in the introduction, the primary benefits of utilising the proposed methodology and framework are reusability and comparability. Furthermore, the framework is lightweight and allows a developer to choose the required type of the physical network and to control and optimise the binary layout of protocol messages. Hence, as an example, the framework is suitable for developing P2P applications that are executed on mobile devices.

Besides the benefits, the approach has some limitations and drawbacks. First, utilising a framework for the development of a P2P application immediately increases the complexity of the development and makes the application dependent on the framework. Second, the framework limits a developer's degrees of freedom. For instance, a developer must obtain random numbers, system time and network addresses from the framework exclusively, avoid static variables and so forth. Additionally, the employed event-driven control model dictates that there is only one control flow and that execution of the model is strictly sequential. Hence, a developer cannot use multithreading and every operation of the model must return control to the framework eventually and cannot be pre-empted by others. The sooner operations return control the more reactive an application appears and the less they hinder competing time-critical operations. Moreover, programming and debugging distributed applications with the event-driven control model tends to be cumbersome. A developer must spread logic over several event-handling operations instead of structuring the program as transparently as possible for a human. This control model results in a rather technical and "low level" way of implementing communication that obfuscates much of the original algorithm.



Search String: linux

Search Results: 'linux'

Time	From	Filename	Filesize
19:23:37.359	/72.161.188.184:26122	Suse Linux Professional 10.exe	233472
19:23:37.359	/72.161.188.184:26122	SuSE Linux 9.1 Personal.exe	233472
19:23:37.359	/72.161.188.184:26122	SuSE Linux 9.1 Personal 1CD.exe	233472

Connected Neighbours

Time	Address	Info
19:22:36.562	/83.130.46.124:35743	LimeWire/4.10.9
19:22:40.453	/67.149.145.117:6348	BearShare Lite 5.2.1.4
19:22:54.968	/172.188.167.192:6350	LimeWire/4.10.3 (Pro)

Statistics

	This Peer	Observable Network
Messages Sent	18	18
Messages Received	34	34
StreamData Sent (byte)	916	916
StreamData Received (byte)	90.112	90.112
Events Processed		36
SystemTime		00:01:41.672

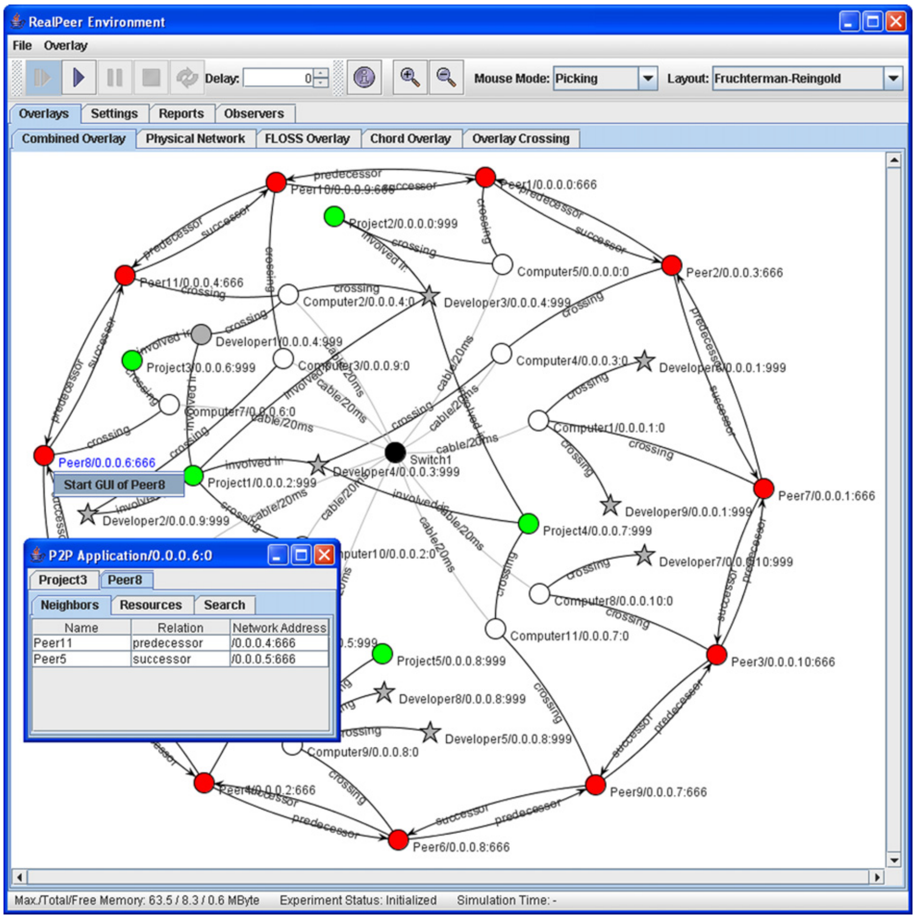
Handshake Message Log

Time	Direction	From	To	Message Type
19:22:54.968	Outgoing	/192.168.123.100:3560	/172.188.167.192:6350	GNUTELLA/0.6 200 OK
19:22:54.906	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	GNUTELLA/0.6 200 OK
				X-Version: 4.9
				Pong-Caching: 0.1
				Accept-Encoding: deflate
				X-Requeries: false
				X-Locale-Pref: en
				X-Guess: 0.1
				X-Max-TTL: 3
				Vendor-Message: 0.2
				X-Ultrapeer-Query-Routing: 0.1
				X-Query-Routing: 0.1
				Listen-IP: 172.188.167.192:6350
				X-Ext-Probes: 0.1
				Remote-IP: 84.129.202.252
				GGEP: 0.5
				X-Dynamic-Querying: 0.1
				X-Degree: 32
				User-Agent: LimeWire/4.10.3 (Pro)
				X-Ultrapeer: True

Protocol Message Log

Time	Direction	From	To	Message Type
19:23:38.546	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessageQueryHit (TTL: 2, hops: 3), numFiles: 6, maxUploadSpeed: 528, serventGUI...
19:23:38.234	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessageQueryHit (TTL: 2, hops: 3), numFiles: 1, maxUploadSpeed: 24, serventGUID...
19:23:37.640	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessageQueryHit (TTL: 2, hops: 3), numFiles: 1, maxUploadSpeed: 152, serventGUI...
19:23:37.359	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessageQueryHit (TTL: 1, hops: 3), numFiles: 3, maxUploadSpeed: 841, serventGUI...
19:23:37.359	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessageQueryHit (TTL: 2, hops: 3), numFiles: 8, maxUploadSpeed: 350, serventGUI...
19:23:36.906	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessageQueryHit (TTL: 2, hops: 3), numFiles: 2, maxUploadSpeed: 3200, serventG...
19:23:36.406	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessageQueryHit (TTL: 2, hops: 3), numFiles: 1, maxUploadSpeed: 350, serventGUI...
19:23:35.781	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessageQueryHit (TTL: 2, hops: 2), numFiles: 1, maxUploadSpeed: 600, serventGUI...
19:23:33.718	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessageQueryHit (TTL: 2, hops: 3), numFiles: 3, maxUploadSpeed: 168, serventGUI...
19:23:31.812	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessageQueryHit (TTL: 2, hops: 3), numFiles: 1, maxUploadSpeed: 1080, serventG...
19:23:31.468	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessagePong (TTL: 0, hops: 6), address: /24.197.164.199:2988, numFiles: 1643, fil...
19:23:26.796	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessageQueryHit (TTL: 2, hops: 2), numFiles: 4, maxUploadSpeed: 104, serventGUI...
19:23:23.500	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessageQueryHit (TTL: 2, hops: 2), numFiles: 3, maxUploadSpeed: 350, serventGUI...
19:23:22.625	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessageQueryHit (TTL: 2, hops: 2), numFiles: 6, maxUploadSpeed: 136, serventGUI...
19:23:18.703	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessagePong (TTL: 0, hops: 6), address: /154.20.255.36:19240, numFiles: 1, fileSiz...
19:23:17.984	Outgoing	Dieter/192.168.123.10...	/172.188.167.192:6350	MessageQuery (TTL: 7, hops: 0), searchCriteria: linux, minSpeed: 0
19:23:17.968	Outgoing	Dieter/192.168.123.10...	/83.130.46.124:35743	MessageQuery (TTL: 7, hops: 0), searchCriteria: linux, minSpeed: 0
19:23:17.968	Outgoing	Dieter/192.168.123.10...	/67.149.145.117:6348	MessageQuery (TTL: 7, hops: 0), searchCriteria: linux, minSpeed: 0
19:23:06.046	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessagePong (TTL: 0, hops: 5), address: /24.107.213.61:6349, numFiles: 350, fileSi...
19:23:03.562	Incoming	/172.188.167.192:6350	/192.168.123.100:3560	MessagePong (TTL: 1, hops: 1), address: /81.10.82.143:6346, numFiles: 66, fileSize...

Fig. 12. Screenshots of the Gnutella Application connected to a real Gnutella Network on the Internet.



Delay: 0

Mouse Mode: Picking

Layout: Fruchterman-Reingold

Name	Relation	Network Address
Peer11	predecessor	/0.0.0.4:666
Peer5	successor	/0.0.0.5:666

Max./Total/Free Memory: 63.5 / 8.3 / 0.6 MByte Experiment Status: Initialized Simulation Time: -

Fig. 13. RealPeer front-end.

Scalability is an important non-functional requirement of the framework. Riley et al. [26] point out that it is often not possible to obtain significant evaluation results from simulations with small numbers of peers (e.g. thousands), when the intended P2P system is required to contain a large number of peers (e.g. millions). As the evaluation of the framework revealed, memory consumption seems to be the major bottleneck that limits scalability. We expect that as models evolve into detailed real systems, the number of peers that can be simulated on a single machine is decreasing fast. To illustrate this: The simple evaluation model that we have implemented consists of about 12 k lines of Java code, whereas LimeWire [27], a mature file-sharing application, consists of more than 400 k lines of Java code. To overcome the scalability limits, parallel and distributed simulation can be leveraged. However, literature (e.g. [28,29,15]), indicates that this task is not trivial and communication overhead degrades performance when the number of computing nodes exceeds a critical size. Alternatively, parts of the model and its state could be swapped to hard disc, which would improve scalability while dramatically degrading performance.

In summary, in order to profit from the proposed methodology and framework, a developer has to accept some limitations. With respect to the current state-of-the-art of developing P2P applications, we do not regard these limitations as significant. As an overall result, we argue that our approach can improve the process of developing P2P systems.

6. Summary

In this paper, we presented the RealPeer methodology and 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. 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. Finally, we discussed benefits and limitations of the RealPeer framework.

7. Current and future work

The RealPeer methodology and framework was already used to implement and evaluate an organisation-oriented P2P system [30] that optimises the efficiency of P2P search methods by using two or more overlay networks. Additionally, simulation performance was already improved by deploying Grid middleware. In this effort, the same experiment is executed in parallel on a number of Grid nodes with different random seeds. Subsequently, the individual results are averaged to factor out randomness.

Currently, a graphical front-end for the RealPeer framework is being developed, which integrates the RealPeer framework and several other tools and provides a GUI for the integrated components.

It allows a user to configure the framework, models and experiments and to execute models within the framework. Existing topology generators were integrated that can supply simulation models with starting topologies for their overlay networks. Visualisation components are integrated that visualise overlay networks, gathered simulation data and analysis results. The RealPeer front-end is widely generic and supports P2P systems that are based on one or more overlay networks [30]. Fig. 13 depicts the GUI of the front-end visualising an example of a combined overlay network, which consists of a simplified Chord ring [31] and a hierarchical overlay network to allow hash based and flooding search methods within one P2P system. The objective of using combined overlay networks is to improve desired system properties such as efficiency or quality of service. Several observers were implemented to collect data for calculating different efficiency metrics (e.g. number of hops, number of messages, etc.). These metrics can be visualised during a simulation run, so that a developer can observe the effects of model changes immediately. In addition, the front-end offers functionality to display P2P application user interfaces of selected peers to allow the interaction with a P2P system during a simulation run. This functionality is very useful for debugging and testing purposes.

Our future work targets the construction of additional models in order to evaluate further the methodology and framework. We intend to fully develop a process model for the simulation-based development of P2P systems. Furthermore, we plan to investigate alternative approaches for specifying P2P applications. The RealPeer framework utilises a bottom-up approach for developing P2P applications, which exhibits a low level of abstraction. As a proximate approach that borrows ideas from JEE application servers and Microsoft's "managed code" proposition, the software framework could be replaced by an execution environment that accepts models as "plain old Java objects" (POJOs) instead of models that inherit and implement interfaces and classes of a framework. This would hide some of the complexity of using the execution environment and help a developer to focus on the P2P application logic. Additionally, we intend to investigate the utilisation of domain-specific languages and the concepts of model-driven development [32] in order to increase the level of abstraction. Finally, we target to further develop the RealPeer front-end, which is a step towards a comprehensive integrated development environment (IDE) for P2P systems.

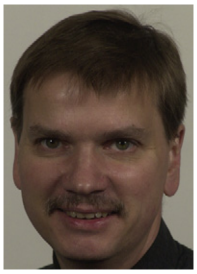
Acknowledgement

We would like to thank Ludger Bischofs for providing valuable feedback and suggestions. In addition, we would like to thank the reviewers of this journal for their valuable comments.

References

[1] D.S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, Z. Xu, Peer-to-Peer Computing, Technical Report HPL-2002-57, HP Laboratories Palo Alto, 2002.
[2] A.M. Law, W.D. Kelton, Simulation Modeling and Analysis, third ed., McGraw-Hill, 2000.
[3] L. Bischofs, S. Giesecke, M. Gottschalk, W. Hasselbring, T. Warns, 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.
[4] D. Hildebrandt, The RealPeer Framework, 2007. <http://sourceforge.net/projects/realpeer/>.
[5] The Network Simulator – ns-2, 2007. <http://www.isi.edu/nsnam/ns/>.
[6] Scalable Simulation Framework, 2007. <http://www.ssfnet.org/>.
[7] D. Hildebrandt, Eine simulationsbasierte Entwicklungsumgebung für Peer-to-Peer-Systeme, Master's thesis, Universität Oldenburg, 2006.
[8] A. Montresor, G.D. Caro, P.E. Heegaard, Architecture of the Simulation Environment, Technical Report D11, University of Bologna, 2004.
[9] R. Kurmanowytsch, Omnix: An Open Peer-to-Peer Middleware Framework, Ph.D. thesis, TU Wien, 2004.
[10] V. Darlagiannis, A. Mauthe, N. Liebau, 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.
[11] I. Baumgart, B. Heep, S. Krause, OverSim: a flexible overlay network simulation framework, in: Proceedings of 10th IEEE Global Internet Symposium, 2007, pp. 79–84.
[12] M.B. Jones, J. Dunagan, Engineering Realities of Building a Working Peer-to-Peer System, Technical Report MSR-TR-2004-54, Microsoft Research, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, 2004.
[13] P. Urbán, X. Défago, A. Schiper, Neko: a single environment to simulate and prototype distributed algorithms, Journal of Information Science and Engineering 18 (6) (2002) 981–997.
[14] A. Rodriguez, C. Killian, S. Bhat, D. Kostic, A. Vahdat, MACEDON: methodology for automatically creating, evaluating, and designing overlay networks, in: Proceedings of the First USENIX Symposium on Networked Systems Design and Implementation, San Francisco, USA, 2004.
[15] S. Lin, A. Pan, R. Guo, Z. Zhang, Simulating large-scale P2P systems with the WiDS toolkit, in: Proceedings of the 13th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005), 27–29 September 2005, Atlanta, GA, USA, IEEE Computer Society, 2005, pp. 415–424.
[16] K. Aberer, L.O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, M. Hauswirth, The essence of P2P: a reference architecture for overlay networks, in: Fifth IEEE International Conference on P2P Computing, Germany, 2005.
[17] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, I. Stoica, Towards a common API for structured Peer-to-Peer overlays, in: Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS03), Berkeley, CA, 2003.
[18] JXTA, 2007. <http://www.jxta.org/>.
[19] L. Melville, J. Walkerdine, I. Sommerville, D9 – P2P Reference Architectures (Final Version), P2P Architect Project, Deliverable Internal IST-2001-32708, Computing Department, Lancaster University, UK, 2003.
[20] I. Sommerville, Software Engineering, seventh ed., Addison-Wesley, Pearson, 2004.
[21] J. Banks (Ed.), Handbook of Simulation: Principles, Methodology, Advances, Applications and Practice, A Wiley-Interscience Publication, 1998.
[22] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, R. Stafford, Patterns of Enterprise Application Architecture, Addison Wesley, 2002.
[23] J. van Gurp, J. Bosch, Design, implementation and evolution of object oriented frameworks: concepts and guidelines, Software – Practice and Experience 31 (3) (2001) 277–300.
[24] The Annotated Gnutella Protocol Specification v0.4, 2007. <http://rfc-gnutella.sourceforge.net/developer/stable/>.
[25] T. Hong, Performance, in: A. Oram (Ed.), Peer to Peer: Harnessing the Power of Disruptive Technologies, O'Reilly and Associates, 2001.
[26] G.F. Riley, M.H. Ammar, Simulating large networks: how big is big enough?, in: Proceedings of First International Conference on Grand Challenges for Modeling and Simulation, 2002.
[27] LimeWire – Open Source P2P File Sharing, 2007. <http://www.limewire.org/>.
[28] R.M. Fujimoto, Parallel and Distribution Simulation Systems, A Wiley-Interscience Publication, 2000.
[29] Q. He, M. Ammar, G. Riley, H. Raj, R. Fujimoto, Mapping peer behavior to packet-level details: a framework for packet-level simulation of Peer-to-Peer systems, in: MASCOTS 2003, 2003.
[30] L. Bischofs, U. Steffens, Organisation-oriented Super-Peer Networks for digital libraries, in: M. Agosti, H.-J. Schek, C. Türker (Eds.), Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures: Sixth Thematic Workshop of the EU Network of Excellence DELOS, Cagliari, Italy, LNCS, vol. 3664, Springer, 2005.
[31] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable Peer-to-Peer lookup service for internet applications, in: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM Press, 2001, pp. 149–160.
[32] OMG Model Driven Architecture, Object Management Group, 2007. <http://www.omg.com/mda/>.

Dieter Hildebrandt is a research assistant at the Business Information Management Division of the OFFIS Institute for Information Technology, Germany, and he is currently pursuing a Ph.D. in Computer Science. He received his Diploma degree in Computer Science from the University of Oldenburg, Germany. For four years he was head of the Software Engineering Department of Piranha Bytes GmbH, Germany. His research interests include simulation and modelling of P2P systems and service-oriented architectures. He is a member of the German Association for Computer Science GI.



Wilhelm Hasselbring is a Professor of Software Engineering, Chair of the Graduate School TrustSoft on Trustworthy Software Systems at the University of Oldenburg (Germany), and a Scientific Director in the OFFIS Institute for Information Technology. He received his Diploma degree in Computer Science from the Technical University of Braunschweig, Germany, in 1989; and the Ph.D. degree in Computer Science from the University of Dortmund, Germany, in 1994. From 1998 to 2000, he was with Tilburg University, Netherlands. His research interests include software engineering and distributed systems, particularly software architecture design and evaluation. He is a member of the ACM, the IEEE Computer Society, and the German Association for Computer Science GI. Contact him at the University of Oldenburg, D-26111 Oldenburg, Germany; hasselbring@informatik.uni-oldenburg.de; http://se.informatik.uni-oldenburg.de/.