Comparative Evaluation of Dependability Characteristics for Peer-to-Peer Architectural Styles by Simulation [⋆]

Ludger Bischofs, Simon Giesecke, Michael Gottschalk, Wilhelm Hasselbring [∗], Timo Warns, Stefan Willer

Software Engineering Group
Carl von Ossietzky University Oldenburg
26111 Oldenburg, Germany

Abstract

An important concern for the successful deployment of a dependable system is its quality of service (QoS), which is significantly influenced by its architectural style. We propose the comparative evaluation of architectural styles by simulation. Our approach integrates architectural styles and concrete architectures to enable early design-space exploration in order to predict the QoS of peer-to-peer systems. We illustrate the approach via two case studies where availability of resources and performance of peer-to-peer search methods are evaluated. Based on our experience with these simulation environments, we sketch tool support for simulating architectural changes at runtime.

Key words: Architectural Style, Dependability, Peer-to-Peer System

[⋆] This work is supported by the German Research Foundation (DFG), grant GRK 1076/1, and by the EU Network of Excellence DELOS, contract nr. 507618.
[∗] Corresponding author. Telephone: +49-441-798-2992, Fax: +49-441-798-2196
Email addresses: `ludger.bischofs@informatik.uni-oldenburg.de` (Ludger Bischofs), `simon.giesecke@informatik.uni-oldenburg.de` (Simon Giesecke), `michael.gottschalk@informatik.uni-oldenburg.de` (Michael Gottschalk), `hasselbring@informatik.uni-oldenburg.de` (Wilhelm Hasselbring), `timo.warns@informatik.uni-oldenburg.de` (Timo Warns), `stefan.willer@informatik.uni-oldenburg.de` (Stefan Willer).

1 Introduction

Architectures of peer-to-peer systems are constrained by architectural styles that are specializations of the abstract peer-to-peer style. The selected concrete architectural style for a specific system has a crucial impact on the emergent dependability characteristics of the realized system services. The decision for an architectural style is one of the earliest design decisions. A change of such a decision is usually not feasible with reasonable effort, because this decision impacts all subsequent development phases. Therefore, different architectural styles should be rigorously assessed early in the development process.

Different alternatives must be evaluated to enable such design decisions. The evaluation is difficult in such a context, because formal analysis or observation of real-world systems is still an unsolved problem for large-scale peer-to-peer systems. The dynamic evolution of a peer-to-peer topology is an additional obstacle. Even if systems with a comparable style are already deployed, they may not allow for appropriate measurements. Developing a prototype system and deploying it for measurements in a production environment requires high efforts.

According to the ISO/IEC 14598-1 (1999) standard, we follow the terminological conventions: A *characteristic* (e.g., performance) is a high-level quality property of a software system, which is refined into a set of *sub-characteristics* (e.g., time efficiency), which are again refined into quality attributes (e.g., throughput). Quality *attributes* can be measured using a quality metric. A *metric* is a measurement scale combined with a procedure describing how measurement should be conducted (e.g., number of processed jobs per second). The application of a quality metric to a specific software system yields *measurements.*

Avižienis et al. (2004) structure dependability of computing systems by the *threats* to dependability, the *characteristics*[1] of dependability, and the *means* to attain dependability. Dependability engineering aims at delivering system services that can be justifiably trusted. Dependability integrates the basic characteristics availability, reliability, safety, confidentiality, integrity, and maintainability. The characteristics availability, reliability, and performance determine the *Quality of Service (QoS)* of a system.

[1] Avižienis et al. (2004) call the characteristics attributes. We follow the differentiation among characteristics and attributes of the ISO/IEC 14598-1 (1999) standard; thus use the term characteristic.

Any software system has a software architecture that may or may not be described explicitly (Medvidovic and Taylor, 1998). We refer to the definition of software architecture as given in the ANSI/IEEE Standard 1471-2000 (Maier et al., 2001):

> “The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.” (IEEE, 2000)

The underlying *principles* for the architecture of some concrete software system include constraints such as architectural styles (Garlan et al., 1994).

Peer-to-peer systems deserve particular attention in dependability engineering as they are increasingly deployed in application domains with high dependability requirements. Usually, peer-to-peer systems are large as they may consist of several thousand peers. Thus, it is unrealistic to evaluate their emergent quality characteristics only analytically by means of formal reasoning. Complementary to analytical reasoning, we propose the comparative evaluation of architectural styles by means of simulation. A peer-to-peer system may dynamically change its architecture at runtime as peers join or leave the system. However, these architectural changes are constrained by the architectural *styles* imposed on the system. We identify basic characteristics of architectural styles for peer-to-peer systems and use architectural descriptions as input for simulation tools to predict the QoS of real-world systems.

The contribution of this paper is a simulation-based approach that supports early design decisions of architectural styles for peer-to-peer systems. Simulation allows to evaluate architectural styles for systems with a large number of participating peers. It allows to take the evolution of the topology of the system under development into account and enables useful measurements. The approach integrates peer-to-peer architectural styles, architectures, systems, and quality characteristics. We demonstrate the usefulness of the approach by presenting two example simulation systems. Future work will emphasize the simulation of architectural changes at runtime, based on our hitherto existing experience with different simulation systems.

The paper is organized as follows. Section 2 introduces the basics of peer-to-peer systems and peer-to-peer architectural styles. Section 3 illustrates the proposed approach to simulation-based evaluation of architectural styles and presents the results of two case studies that evaluate different architectural styles with respect to their influence on availability and performance. Section 4 describes the architectures of our current and future simulation environments. Section 5 discusses related work. Section 6 draws some conclusions and indicates directions for future work.

2 Peer-to-Peer Architectural Styles

The peer-to-peer style is an extension to the client-server style for architecting distributed systems. In a client-server system, each node has a fixed role of either acting as a client or as a server. In a peer-to-peer system, any node may act both as a client and as a server, that is, any node may provide and consume services. Schollmeier (2001) calls nodes taking both roles "*servents*", which combines the terms "*serv*er" and "cli*ent*".

A peer-to-peer architectural style is a specialization of the generic peer-to-peer style and defines a family of peer-to-peer systems. Peer-to-peer architectural styles can be characterized by the *type of decentralization*, *type of structure*, and the *type of communication*.

The **type of decentralization** can essentially be *decentralized*, *hybrid*, or *super-peer*. In a decentralized system, all services are decentralized and can be offered by any peer. For example, the developers of the *Gnutella* (Gnutella, 2006) file-sharing system employ a decentralized style allowing each peer to issue search requests, to answer search requests, and to share requested files. In a hybrid system, some services, such as directory services, are centrally offered by dedicated servers, which do not act as regular peers. In contrast to client-server architectures, other services remain decentralized. For example, *Napster* (Napster, 2006) is a hybrid file-sharing system with centralized servers that maintain indexes of files to answer search requests. Regular peers search for files with the help of these servers. Like peers of Gnutella, they share their files decentrally. In a super-peer system, some peers are elevated to a distinguished status, either by configuration or by self-organized election. These super-peers help structuring the topology and offer additional services. Figure 1 illustrates an architecture with a super-peer style. For example, the file-sharing system *KaZaa* (Kazaa, 2006) is a system with super-peers that provide an indexing service for data resources. KaZaa peers with high-bandwidth network connections automatically become super-peers and host indexes for regular peers. The type of decentralization has an impact on the types of nodes and on the constraints for run-time evolution of the topology.

The **type of structure** imposes global constraints on the topology. A peer-to-peer architectural style may impose certain restrictions on the topology. For example, peers may organize themselves to form a tree, mesh, or ring. A structure can be utilized to improve the performance of a system, for example by reducing communication overhead for searching. The costs to maintain a topology following structural constraints can be high when peers enter and leave the system frequently and need to find their position within the topology. For example, Stoica et al. (2001) present the architecture *Chord* where the peers are organized to form a ring. Each peer maintains information about the ring to improve the performance of search operations. Most super-peer systems and many systems with an unstructured topology exhibit the so-called *small-world phenomenon*. Originally, Milgram (1967) examined this phenomenon in sociology. The phenomenon means that there exists a short chain of social acquaintances for any two humans. Watts and Strogatz (1998) use the term for networks that have any two nodes connected by a short path of intermediate nodes. Networks with such structural constraints are highly scalable, because communication is efficient in spite of a large number of nodes. For example, Clarke et al. (2000) present the file-sharing system *Freenet* that exhibits the small-world phenomenon through the use of adaptive routing strategies.

The **type of communication** may essentially be distinguished into *direct* and *indirect* communication. Communication is direct if peers only have direct connections of the underlying network layer between them. For example, the peers of the Gnutella system only use direct communication. The communication is indirect if peers can communicate via intermediate peers. Intermediate peers mediate the communication between end-to-end communication partners. For example, the peer-to-peer protocol *JXTA* (JXTA, 2006) enables peers to relay messages via intermediate peers. The choice of a communication type determines the type of communication links and influences the rules for creating new communication links.

Often a concrete peer-to-peer architectural style implies trade-offs among different quality characteristics. For example, the developers of Gnutella sacrificed performance for improving resilience. They employed an architectural

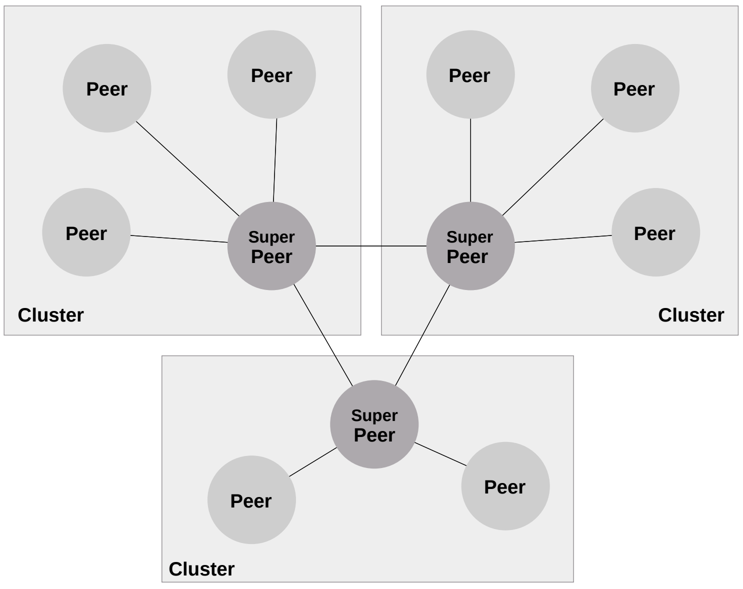


Fig. 1. Example architecture in a super-peer style. A regular peer is connected to a single super-peer. A super-peer forms a cluster with its regular peers. Each pair of super-peers is connected by a link. Indirect communication is allowed among regular peers along the links.

style of complete decentralization without any single-point-of-failure to achieve high resilience despite of node failures. However, searching in Gnutella tends to be inefficient as the employed algorithms flood the network with query messages. Developers of other systems, such as Napster, made a different trade-off in favor of performance by choosing a style with central servers that maintain complete indexes.

A peer-to-peer architectural *style* also constrains the run-time evolution of the network topology, which is not included in the classification scheme. For example, when peers join or leave a system, other peers may need to change their links in order to satisfy constraints of the architectural style.

3 Simulation of Peer-to-Peer Styles

Figure 2 illustrates our simulation approach to the evaluation of the influence of peer-to-peer architectural styles on the quality characteristics of services within concrete architectures and systems. We define concrete architectures of peer-to-peer systems based on the style and based on fault characteristics of the participating peers and connections. This definition is based on observations of similar real-world systems and on assumptions about future systems. The architectural style constrains peers' interconnections. Furthermore, the system is influenced by fault characteristics that describe when and how peers and connections fail. These characteristics determine the periods of correct service and outage. A real-world system corresponds to a concrete architecture. We predict the quality characteristics of the real-world system by performing measurements on a simulated system that corresponds to the same architecture as the real-world system. The validity of the prediction is determined by the accuracy and the precision of the measurements of the simulation.

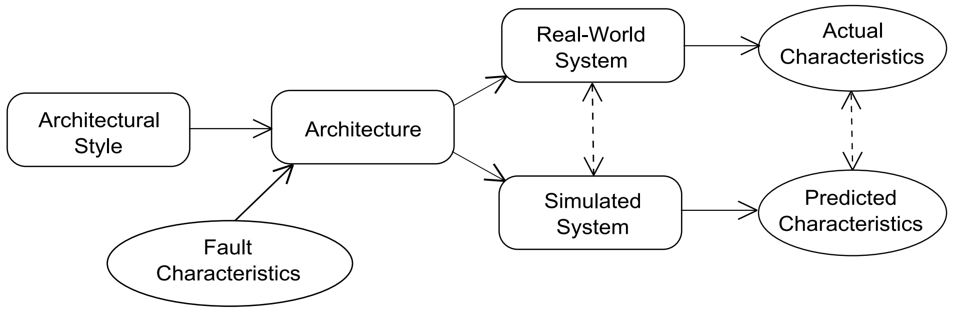


Fig. 2. Overview of our evaluation approach. Concrete architectures are defined based on the style and on fault characteristics. The simulated system corresponds to such an architecture and models a real-world system conforming to the same architecture. The real-world system may not readily exist, e.g., when styles for future systems are evaluated. Quality characteristics are measured for the simulated system to predict characteristics of the real-world system.

In the next two subsections, two exploratory case studies are presented to illustrate our approach to the comparative evaluation of peer-to-peer architectural styles. The first case study evaluates the availability of access to replicated resources. The second case study evaluates the performance of searching in peer-to-peer systems with different architectural styles.

3.1 Case Study to Evaluate Availability

In our first case study, we compare the availability of access to replicated resources for (1) three different architectural styles, (2) two traditional voting-based replication strategies, (3) two mappings of replicas to peers, and (4) three different peer failure distributions. Many peer-to-peer systems offer system services such as searching for data resources. A service depends on resources held by certain peers. If a single peer owns a resource that is required by a system service, a failure of this peer leads to a failure of the service as a whole. In other words, the service jointly fails with the peers owning the required resources. The overall system needs to be able to tolerate faults such as peer and connection failures to improve the dependability of its services. Exemplarily, we consider a system offering a service that enables access to a resource.

We assume the availability, which is the readiness for correct service (Avižienis et al., 2004), to be equivalent to the availability of *access* to a resource that is the probability that the replicated resource can be accessed successfully. We use the number of successful accesses per overall number of attempts as the metric for availability. Availability of access is impaired by failures of peers and connections, for example if the failure of an intermediate peer renders communication between two other peers impossible.

In traditional distributed systems, the problem that the failure of individual nodes disables the access to resources is addressed by replication techniques. Replication is realized by creating *replicas*, which are copies of data resources, and by maintaining their consistency. The techniques that realize consistency require access to more than one replica for read and write operations. Therefore, reading and writing a resource within a peer-to-peer system requires access to more than one peer owning a replica, respectively. These peers may not be accessible directly if the peer-to-peer system employs indirect communication. Hence, other peers may be used to realize communication among hosting peers. The availability of access to the replicated resource depends on the availability of all peers and connections that are involved. As the architectural style determines how peers are connected and communicate, it has a major impact on the availability of access to a resource.

Many widely distributed peer-to-peer systems suffer from the effect that peers leave the system and do not return at all. The rate of peers that leave the system permanently is called the *churn rate* of a peer-to-peer system. Traditional replication strategies are static in nature, that is, replicas remain hosted on the same nodes over the whole lifetime of the system. Therefore, they are not very well suited for peer-to-peer systems, because a permanently leaving peer may have participated in the replication strategy. If too many of such peers fail, the replicated resource becomes inaccessible forever. However, Bhagwan et al. (2003) observed that the distribution of peer failures within a real-world file sharing system can be described by an overlay of a short-term and a long-term distribution. The long-term distribution reflects the churn rate and the short-term distribution reflects time-of-day effects. Within our case study, we focused on traditional replication strategies that may tolerate failures caused by short-term time-of-day effects. Peers leave the system for short periods of time per day. If such outages do not affect too many peers, traditional replication strategies are able to tolerate the peer failures, because a sufficient number of replicas is always available. Tolerating long-term effects requires other strategies with relaxed consistency constraints or techniques for relocating replicas among peers.

Simulation Scenarios

The input parameters for a simulation run are defined in a scenario. A scenario is an abstraction of a global view on a peer-to-peer system for a certain period of time. It defines the inputs that are required to perform a simulation such that meaningful measurements are possible. For our case study, a scenario defines sets of peers and links, their failure distributions, and a mapping of replicas to peers. The architecture of such a scenario conforms to a specific architectural style. Usually, a style does not restrict the number of participating peers, but the possible topologies. However, a scenario defines a limited number of participating peers. Therefore, a style implies an infinite number of different scenarios. In this paper, we exemplarily evaluate one scenario per style.

Section 2 discusses several options for peer-to-peer architectural styles. Manifold architectures can be combined from these options for design-space exploration. The type of decentralization determines the possible distribution of services. Exemplarily, we consider decentralized and super-peer architectures. The type of communication determines whether peers may communicate indirectly with each other. We only consider architectures that offer indirect communication. The type of structure determines the restrictions on the network topology. As illustrated in Figure 3, we consider unstructured, mesh-like, and a type imposed by a super-peer approach for the type of structure:

- Unstructured styles only impose few restrictions on the topology. In the scenario, the peers are connected randomly by 2 to 5 connections per peer.
- Styles with mesh-like structures have their peers connected to form a mesh, whereby each peer is connected to four neighboring peers.
- The super-peer structure is formed as follows: The super-peers form an inner ring within the system. Each super-peer is connected to its two succeeding and two previous neighboring peers on the ring. Each regular peer is connected to one super-peer only, i.e., there is no redundancy of super-peers for a client peer. This architecture exhibits the small-world phenomenon.

In real-world systems, peers with varying failure distributions join a system. This needs to be reflected in the input for the simulation. A scenario describes the failure distributions of individual peers and connections by the *mean time to failure* (MTTF) and *mean time to repair* (MTTR) values. We assume crash-recovery failures for peers and connections only, that is, peers and connections may halt and recover. The failure distribution itself is assumed to be exponential. We employ three different assignments of MTTF and MTTR values to peer and links for each scenario. The first assignment gives constant good values (MTTF = 3 days, MTTR = 6 seconds) to each peer to indicate high overall availability. The second assignment gives constant medium values (MTTF = 17 hours, MTTR = 12 minutes) to each peer to indicate worse availability than before. The third assignment employs a normal distribution of values (MTTF = 38 hours, σ = 22 hours, MTTR = 72 minutes, σ = 44 minutes). Hence, few peers exhibit low availability, few peers exhibit high availability, and most peers exhibit medium availability. The values are derived from the observation of a similar real-world system from Bhagwan et al. (2003), who found that a peer joins and leaves the Overnet system 6.4 times per day on

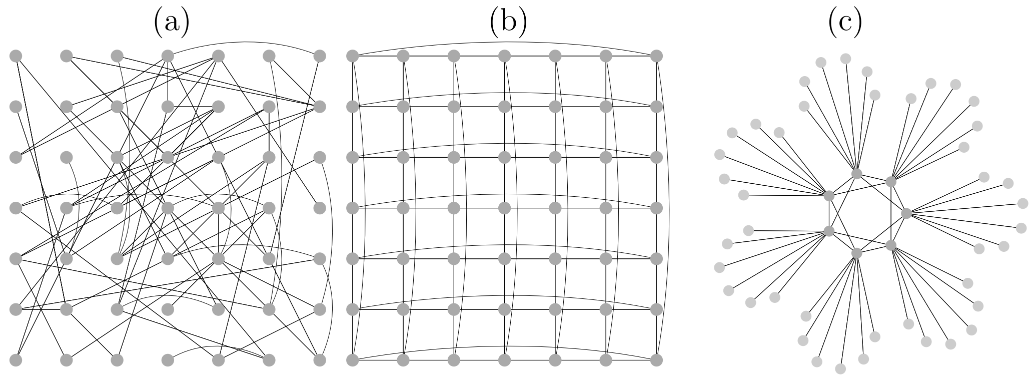


Fig. 3. Illustration of scenarios. The graph (a) exemplarily shows a unstructured network. The peers are connected randomly. The graph (b) exemplarily shows a fully connected mesh-like architecture. Each peer is connected to four neighboring peers. The graph (c) shows a super-peer architecture. Each regular peer is connected to one super-peer only. Each super-peer is connected to four other super-peers.

average. Therefore, an overall MTTF of 24h/6.4 ≈ 225min can be assumed for peers within the Overnet system. We assume higher values as the simulation is meant to evaluate systems deployed in a local area network in which the peers can be assumed to be more reliable.

A mapping must be defined that assigns replicas to hosting peers. For the case study, each mapping is defined to be static, that is, the number and placement of replicas do not change during the simulation. Our decision where to place the replicas is based on the availability of the peers. The first assignment of the mapping is done by assigning the replicas to peers with the highest availability values. These values are determined by the predefined MTTF and MTTR. The second assignment proceeds according to a normal distribution. Therefore, most replicas are hosted by peers that have medium availability.

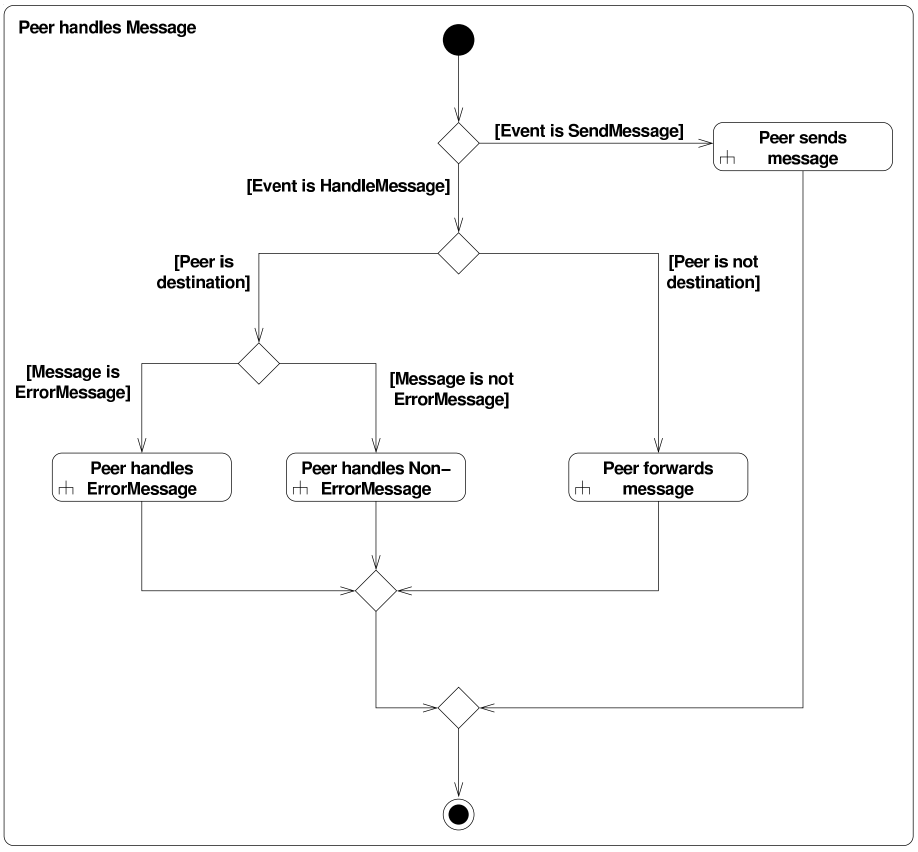

Peer handles Message
[Event is SendMessage]
Peer sends message
[Event is HandleMessage]
[Peer is destination]
[Peer is not destination]
[Message is ErrorMessage]
[Message is not ErrorMessage]
Peer handles ErrorMessage
Peer handles Non–ErrorMessage
Peer forwards message

Fig. 4. Activity diagram “Peer handles message”. The corresponding event to handle a message by a peer either defines to process or to send the message. If it needs to be sent, the corresponding activity is invoked. Otherwise, the simulator checks whether the peer, for which the message is scheduled, is the destination peer of the message. In this case, the corresponding handling activity is invoked while separating error messages and non-error messages. If the peer is not the destination peer of the message, the peer acts as a mediator and forwards the message.

Simulation Model

We developed a discrete-event simulator for a simulation model that approximates the behavior of peers in real-world systems like Freenet as described by Clarke et al. (2000). Due to space limitations, we cannot describe the simulation model in full detail. Exemplarily, Figure 4 illustrates how a single peer handles a message when the simulator processes a message event. This activity is embedded in a more general context of how the simulator handles events and how to perform the simulation in general. It requires more refined activities for behavior like *peer sends message*. For simplicity, we assume a constant transit time of 0.1 seconds for all message transfers and a constant service time of 0.3 seconds for the handling of a message by a peer. In addition to the behavior of real-world peers, the simulation model covers two traditional replication strategies, that is, the peers offer access to a replicated resource as a service.

Each scenario is simulated with replication strategies that are special instances of the *Weighted Voting* replication strategy, which was introduced by Gifford (1979). A number of votes is assigned to each node of a specific set of nodes. Usually, a node is in the set if and only if it hosts a replica of the resource, but other strategies are possible. For simplicity, we assign one vote to each peer that hosts a replica. A transaction that tries to access the replicated resource must collect a quorum of read votes for a read operation and a quorum of write votes for a write operation from the nodes with assigned votes. In order to preserve *one-copy equivalence*, a peer that updates a resource synchronously writes into all replicas that are hosted by the peers of the collected write quorum. The strategies exemplarily used for the simulation are *Read-One-Write-All* (ROWA) and *Majority Consensus*:

- The ROWA strategy requires a transaction to collect a single vote for a read operation and all votes for a write operation.
- Thomas (1979) introduced Majority Consensus, which requires a transaction to collect a simple majority of votes for a read resp. a write operation.

Refer to Bernstein et al. (1987) for a more detailed description of the strategies. Additionally, each scenario is simulated without any replication to compare these results to the results with replication.

Simulation Results

Combining (1) the replication strategies, (2) different mappings of replicas, (3) fault distributions, and (4) peer-to-peer styles from above yields 36 scenarios to simulate. Table 1 shows the results for each scenario. The relative influence of the replication strategies compared to the scenarios without replication is illustrated in Figure 5.

We formulated some fundamental expectations for the replication strategies in advance to show that the simulation setup is reasonable. For example, we expected ROWA to improve the availability of reading for all scenarios. The results confirm our expectations with minor deviations. Both replication strategies improve the availability of read operations, ROWA impairs write operations, and Majority Consensus improves write operations. With a significance threshold of 1%, the deviations from fundamental expectations (e.g., Majority Consensus improves reading more than ROWA for scenario class 2) are not significant. However, it is surprising that the influence of ROWA is almost equal to the influence of Majority Consensus for most scenarios. As could be expected, the influence of Majority Consensus is almost equal for reading and writing as the same number of votes must be collected. Availability for writing in the ROWA strategy significantly decreases for the normal distribution compared to the constant values, because some peers that host replicas required for the write quorum have low availability.

For scenarios with high availability of peers, the replication strategies have almost no influence for any architectural style. For medium availability of peers, the replication strategies influence reading resp. writing, but there is no significant difference for the different architectural styles. For a normal distribution of peer failures and a replica distribution to peers with highest availability, the systems with a mesh-like topology show worse availability

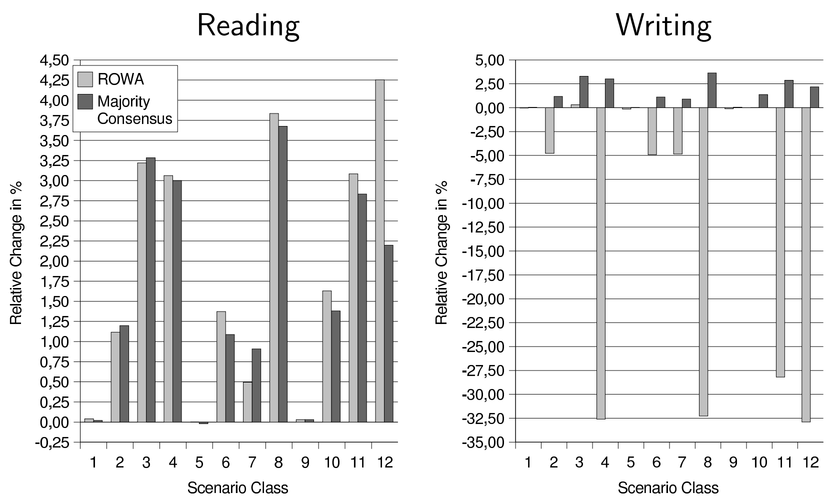
Resulting Availability for Reading				**Replication Control Strategy**					
				None		*Read-One-Write-All*		*Majority Consensus*	
				Replica Distribution		**Replica Distribution**		**Replica Distribution**	
				Best Peers	*Normal Distr.*	*Best Peers*	*Normal Distr.*	*Best Peers*	*Normal Distr.*
Structure	*Unstructured*	**Fault Distribution**	*Constant Good*	0.9993		0.9997		0.9995	
			Constant Medium	0.9762		0.9871		0.9879	
			Normal Distr.	0.9470	0.9468	0.9775	0.9758	0.9781	0.9752
	Grid-Like	**Fault Distribution**	*Constant Good*	0.9998		0.9998		0.9996	
			Constant Medium	0.9752		0.9886		0.9858	
			Normal Distr.	0.9692	0.9387	0.9740	0.9747	0.9780	0.9732
	Super-Peer	**Fault Distribution**	*Constant Good*	0.9994		0.9997		0.9997	
			Constant Medium	0.9628		0.9785		0.9761	
			Normal Distr.	0.9209	0.9099	0.9493	0.9486	0.9470	0.9299

Resulting Availability for Writing				**Replication Control Strategy**					
				None		*Read-One-Write-All*		*Majority Consensus*	
				Replica Distribution		**Replica Distribution**		**Replica Distribution**	
				Best Peers	*Normal Distr.*	*Best Peers*	*Normal Distr.*	*Best Peers*	*Normal Distr.*
Structure	*Unstructured*	**Fault Distribution**	*Constant Good*	0.9992		0.9989		0.9995	
			Constant Medium	0.9763		0.9296		0.9879	
			Normal Distr.	0.9470	0.9467	0.9499	0.6382	0.9781	0.9753
	Grid-Like	**Fault Distribution**	*Constant Good*	0.9996		0.9981		0.9998	
			Constant Medium	0.9750		0.9271		0.9858	
			Normal Distr.	0.9690	0.9388	0.9220	0.6359	0.9778	0.9730
	Super-Peer	**Fault Distribution**	*Constant Good*	0.9994		0.9983		0.9999	
			Constant Medium	0.9627		0.9627		0.9760	
			Normal Distr.	0.9207	0.9098	0.6612	0.6106	0.9471	0.9297

Table 1
Resulting availability values of the first case study. The table is separated into three main rows for each structure of a peer-to-peer system: unstructured, mesh-like, and super-peer. Each main row is separated into three subrows for fault distributions for the peers: constant good, constant medium, and normal distribution. Furthermore, the table is separated into three main columns for each replication strategy: no replication, ROWA, and Majority Consensus. Each main column is separated into two subcolumns for the mappings from replicas to peers: mapping to most available peers and according to a normal distribution. As both yield the same distribution for the constant availability values, their subcolumns are merged.

values than the others. One reason is that, although many redundant paths between any two peers exist, few peer failures make short paths unavailable. The simulation model incorporates a fixed *hop limit* for messages such that the remaining longer paths cannot be utilized for communication. For the super-peer architecture in scenario class 12, ROWA yields the largest improvement of the whole case study and significantly improves reading better than Majority Consensus. One reason is the benefit of the short paths of the super-peer architecture. The absence of redundancy of super-peers may have caused the worse results for Majority Consensus.

The validity and further generalizations of the results are limited for several reasons. Besides general limitations of simulation, a single system is simulated for each architectural style only. Other systems adhering to the same architectural style may yield deviating results. The convergence of results across different systems for one architectural style is a topic for future research.



Scenarios

1	Unstructured, Constant Good	5	Mesh, Constant Good	9	Super Peer, Constant Good
2	Unstructured, Constant Medium	6	Mesh, Constant Medium	10	Super-Peer, Constant Medium
3	Unstructured, Norm. Distr., Best Peers	7	Mesh, Norm. Distr., Best Peers	11	Super-Peer, Norm. Distr., Best Peers
4	Unstructured, Norm. Distr., Norm. Distr.	8	Mesh, Norm. Distr., Norm. Distr.	12	Super-Peer, Norm. Distr., Norm. Distr.

Fig. 5. Relative influence for reading resp. writing. The left chart shows the relative influence of ROWA and Majority Consensus on the availability of read operations compared to simulations without replication. The right chart shows the relative influence for write operations.

3.2 Case Study to Evaluate Performance

Unlike the presented case study in Section 3.1, the following emphasises the performance characteristic of a software system: the performance behavior of search methods in peer-to-peer systems.

Peer-to-peer search methods are essential for the design and implementation of robust and scalable peer-to-peer systems. A survey and a classification of such methods are given by Wang and Li (2003). According to this survey, the first generation of search methods includes protocols like Napster and Gnutella, which are based on unstructured architectures. Many methods from the second generation use distributed hash-tables (DHTs) to achieve load balancing and logarithmic look-up times. The third generation aims to increase the resilience and availability of nodes.

Bischofs and Steffens (2005) introduce a new approach for a class of peer-to-peer systems based on organization-oriented super-peer architectures, which are able to map organizational structures to peer-to-peer-systems. The selection of adequate search methods for such new, not yet existing systems is difficult. Therefore the *eaSim* simulator (described in Section 4.1) has been developed. In the following performance study, simulation results of search methods for the virtual layer of organization-oriented peer-to-peer architectures are described. The virtual layer contains typical peer-to-peer networks and requires efficient search methods to support the top-level organizational layer (Bischofs and Steffens, 2005).

Simulation Scenarios

In this case study, we evaluate several combinations of architectural styles (Section 2) and search methods with respect to their performance. We give a brief description of the input settings and metrics and discuss some results.

Both decentralized and super-peer styles are considered. The topologies of the decentralized systems are mesh-like, random, or ring-based. Peers in a mesh-like topology are connected to four neighbors in an undirected way, whereas randomly connected peers (created by yEd (2006)) have one to four connections. The ring-based structure was adapted from Stoica et al. (2001). All peers are placed on a ring and obtain five pointers to other peers with an exponential distance. All simulated topologies contain 1024 peers. For the super-peer scenarios, 32 super-peers are present; the remaining regular peers are uniformly assigned to the super-peers.

The behavior of each simulated peer is defined by a search method. In this study, we evaluate two different approaches. The first is based on Random

Walks as presented by Lv et al. (2002). Any incoming message is forwarded to a predefined maximum number of neighbors and a maximum hop depth. A history can be used for backtracking, if the maximum hop depth is not exceeded. The second method is based on the Chord protocol (Section 2) and is inspired by Mizrak et al. (2003). Only super-peers are placed on the Chord ring. A special neighborhood (finger-table) guarantees a routing time in $O(\log S)$, where S is the number of super-peers. Each super-peer, as a part of the Chord-ring, is responsible for all resources mapped to a unique identifier. Using a distributed hash-table, the nearest super-peer can be determined. The maintained index of this super-peer provides the necessary information about the hosting peer.

Suitable metrics need to be defined to evaluate the simulated behavior and, therefore, enable a useful simulation run. Exemplarily, the measurements for four metrics are shown in the Figures 6 to 9. They visualize the success rate, the aggregate and individual network load, and the response times for retrieving answers of combined input settings:

- The *query success rate* indicates how many answered queries arrive over time. It shows the share of answered queries and the time when a required threshold has been achieved.
- The *aggregate load* reports the sum of all actually processed messages at each simulation step. It displays the global impact of message occurrence in the network.
- The *average queue size* represents the individual load of each node, measured by the average size of each incoming message queue. This size is an indicator for the node activity.
- One of the most important criteria for characterizing search methods is the time for retrieving an answer, which is given by the *hops per answer*.

At each of the first ten simulation steps, two query messages are initiated from randomly chosen peers. The search criterion is a unique peer name. Initial queries are randomly distributed to participating peers chosen via a linear congruence method from Knuth (1997). All messages are terminated after at most ten visited hops, such that after 20 simulation steps no more query messages are in the system. After this hop-to-life limit only the received responses will be routed the reverse path.

The search methods based on Random Walks send three walkers for each forwarded message at most. The Chord-based method only forwards the original message to a nearer node on the ring. The receiver node is computed by a hash function. This function provides a number for a given peer name, which represents the place on the ring where the searched node is placed.

Simulation Model

The simulation is event driven and time discrete. The respective simulation elements and messages are modeled as entities. All occurring messages are endued with time stamps and an initial sending peer. At each specified time, they are placed into the processing queue by a global dispatcher. The simulation model consists of three layers (Figure 10 in Section 4), where at each layer a message with a specific header can start. To reduce the development effort of search methods, each layer can be simulated separately or in combination (adjacent ones). To combine adjacent layers, a static mapping is defined to map each element from the higher layer to another of the lower one. The search behavior of an element is strictly separated from its state. Moreover, for each element type of an architectural style only one search strategy exists. Fulfilling this, a static strategy and a dynamic element state are connected via the Visitor pattern by Gamma et al. (1995).

At each simulation step, all messages from a peer‘s incoming message queue is processed. The parallelism of messages is realized via an inner cycle. To retrieve simulation results at runtime, the specified metrics are measured and visualized step by step.

Simulation Results

The simulator eaSim (see Section 4.1) is used in this case study. We evaluate four combinations of architectural styles and search methods with static topologies.

- sp_rw_random: All 32 super-peers are randomly connected. The remaining peers are equally distributed over the 32 clusters. Super-peers use the Random Walks method, which forwards a query to at most three directions.
- sp_chord_ring: The super-peers are placed on a ring and follow the Chord-based approach.
- p2p_rw_random: 1024 peers are randomly connected undirectionally to average four neighbors and use the Random Walks search method.
- p2p_rw_mesh: Peers are mesh-structured with two to four undirected edges.

First, we explain the irregular behavior in the first ten steps, as seen in Figure 6. As described before, a new message is initiated at each of these steps, such that the success rate can be higher. The number of queries increases.

The combination of the peer-to-peer architectural styles with mesh or random topologies and the search method Random Walks cause the worst results. In the mesh like system only 20% of the initiated queries were answered by sending over 700 messages, whereas in the random version over 5000 messages are needed for 90% of the queries. Inside the super-peer topologies the number of messages is significantly reduced and most queries are satisfied. The Chord-based search method even ensures a theoretical maximum query success rate. Despite the good performance behavior of the super-peer architectures, Figure 8 shows emerging bottlenecks, because the super-peers act as a gateway for their cluster. The Chord-based approach obviously shows the best performance of this comparison. However, if the dynamic within the network is considered, the maintenance cost of the distributed hash-table increases.

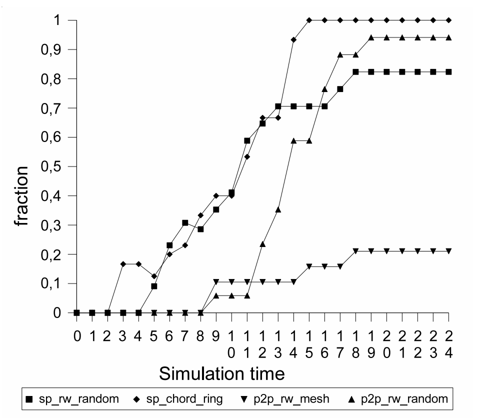


Fig. 6. Time to match a query depicted as query success rate. Five combinations of architectural styles and search methods are shown.

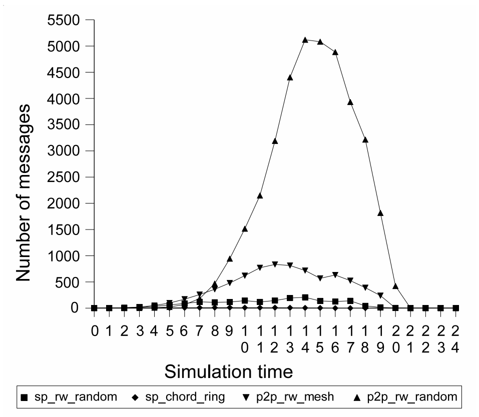


Fig. 7. Resulting messages as aggregate load caused by ten initial queries in the first ten simulation steps. Five combinations of architectural styles and search methods are shown.

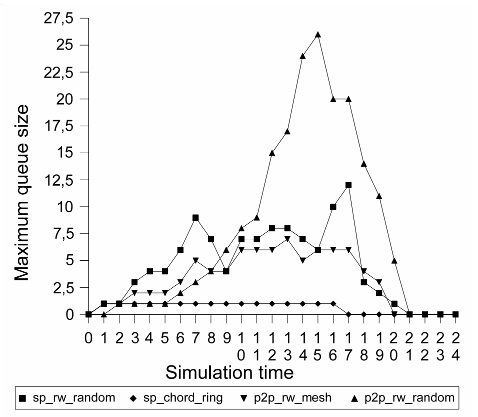


Fig. 8. The maximum queue size at each simulation step to measure the individual load.

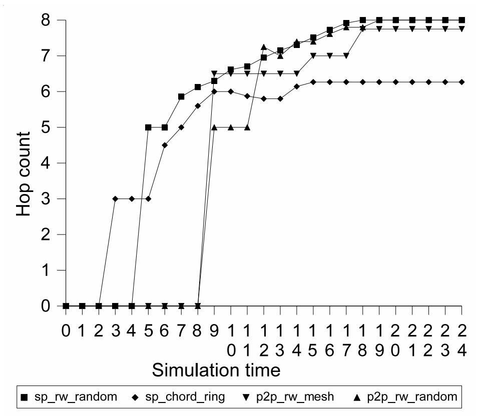


Fig. 9. Average number of hops for each successful message including the one-hop reverse path.

4 Tool Support

The case studies presented in Section 3 employed two specialized simulation environments. Exemplarily, we present the eaSim tool in Section 4.1, which is used in Section 3.2. This tool is available from Sourceforge (eaSim, 2006). We do not only use eaSim for research purposes, but also for teaching our students in evaluating complex software systems. For instance, it is used in the practical assignments in the course Software Systems Engineering at the University of Oldenburg. Section 4.2 sketches our future Eclipse-based simulation environment, that will be based on our experience with the case studies in Section 3. This new tool allows for simulating architectural changes at runtime.

4.1 The eaSim Simulation Tool

The eaSim simulation tool is a time discrete, event-based peer-to-peer simulator based on DESMO-J, a framework developed by Page and Kreutzer (2005). Unlike other peer-to-peer simulators, which are discussed in Section 5, eaSim follows a layered approach for simulating each layer separately and in combination (see Figure 10). The state and the behavior of a peer are modeled independently to allow exchanging the search and routing behaviour. Wizards assist the user during the simulation process. Figure 11 shows the choice, configuration and editing options for element type based behavior. Predefined metrics can be activated and visualized directly at simulation time (see Figure 12).

4.2 Future Eclipse-based Simulation Environment

We are currently developing a new integrated modeling and simulation environment for peer-to-peer systems based on Eclipse (2006) in order to support our framework with additional tools (see Figure 13). The most distinctive feature will be the modeling component providing means for specifying architectural styles with a state-chart-like notation. The aim is to simplify the creation of new architectures and to support fast modifications to existing ones. It will be possible to define the behavior and the properties of the simulated nodes in a scenario. This includes the possibility of dynamically changing the network structure during a simulation run, which was not feasible with eaSim. In order to support the comparison of different architectures, the same scenario can be simulated with different architectural models. The visualization component will be able to compare the results of simulations using diagrams and tables of data.

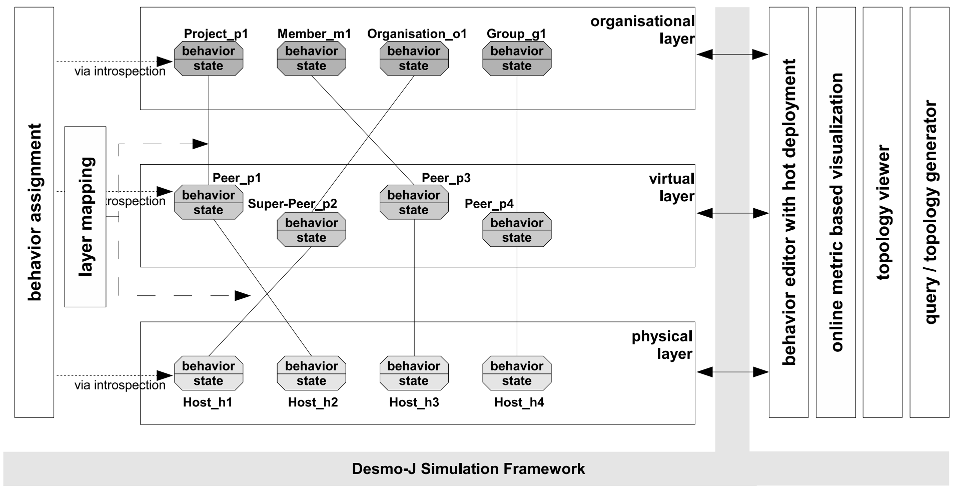
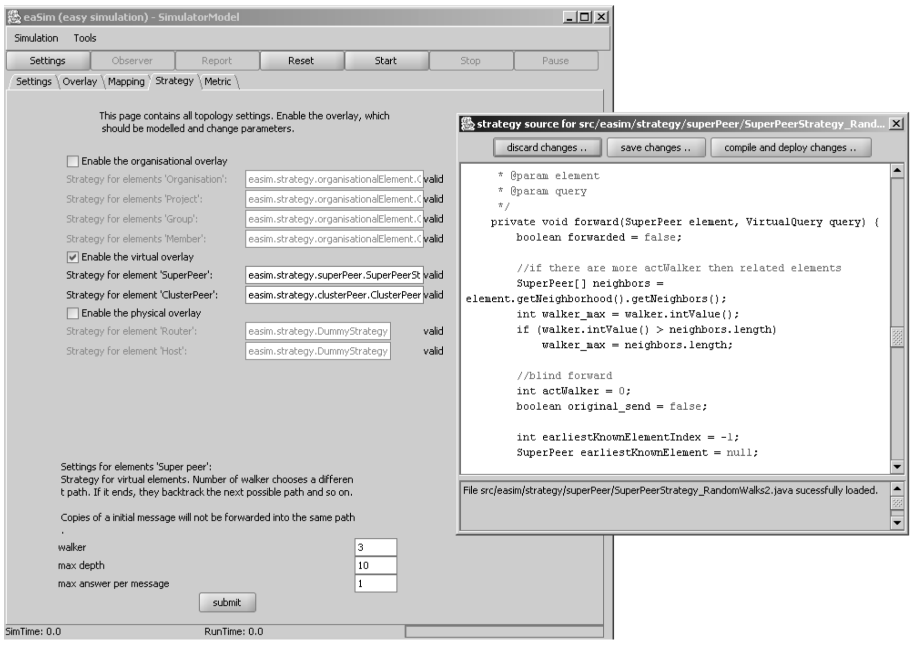


Fig. 10. The architecture of eaSim is divided into three simulation layers on the left and the functionality on the right. At the lowest layer the physical hosts are connected. The middle layer describes the logical interconnection of the (super-)peers. In the upper layer the relationships among organizational units can be seen. Furthermore all adjacent layers are connected, such that each element of the higher layer is hosted on an element of the lower one. Each element is divided into one stateless part for modeling the behavior and one stateful part for modelling the element specific information like neighborhood, cache and resources. Each element type behavior can be assigned to a set of elements.


eaSim (easy simulation) - SimulatorModel
Simulation
Tools
Settings
Observer
Report
Reset
Start
Stop
Pause
Settings
Overlay
Mapping
Strategy
Metric
strategy source for src/easim/strategy/superPeer/SuperPeerStrategy_Rand...
discard changes ..
save changes ..
compile and deploy changes ..
submit

This page contains all topology settings. Enable the overlay, which should be modelled and change parameters.

☐ Enable the organisational overlay		
Strategy for elements 'Organisation':	easim.strategy.organisationalElement.(	valid
Strategy for elements 'Project':	easim.strategy.organisationalElement.(	valid
Strategy for elements 'Group':	easim.strategy.organisationalElement.(	valid
Strategy for elements 'Member':	easim.strategy.organisationalElement.(	valid
☑ Enable the virtual overlay		
Strategy for element 'SuperPeer':	easim.strategy.superPeer.SuperPeerSt	valid
Strategy for element 'ClusterPeer':	easim.strategy.clusterPeer.ClusterPeer	valid
☐ Enable the physical overlay		
Strategy for element 'Router':	easim.strategy.DummyStrategy	valid
Strategy for element 'Host':	easim.strategy.DummyStrategy	valid

Settings for elements 'Super peer':
Strategy for virtual elements. Number of walker chooses a differen
t path. If it ends, they backtrack the next possible path and so on.

Copies of a initial message will not be forwarded into the same path
.

walker	3
max depth	10
max answer per message	1

SimTime: 0.0 RunTime: 0.0

```
     * @param element
     * @param query
     */
   private void forward(SuperPeer element, VirtualQuery query) {
       boolean forwarded = false;

       //if there are more actWalker then related elements
       SuperPeer[] neighbors =
element.getNeighborhood().getNeighbors();
       int walker_max = walker.intValue();
       if (walker.intValue() > neighbors.length)
           walker_max = neighbors.length;

       //blind forward
       int actWalker = 0;
       boolean original_send = false;

       int earliestKnownElementIndex = -1;
       SuperPeer earliestKnownElement = null;
```

File src/easim/strategy/superPeer/SuperPeerStrategy_RandomWalks2.java sucessfully loaded.

Fig. 11. The displayed wizard enables the user to select and configure the search behavior of each element type. Via introspection the configuration parameters are displayed dynamically, such that the behavior specification is strictly separated from the visualization. Furthermore each behavior can be modified with an integrated editor and subsequently compiled and deployed.

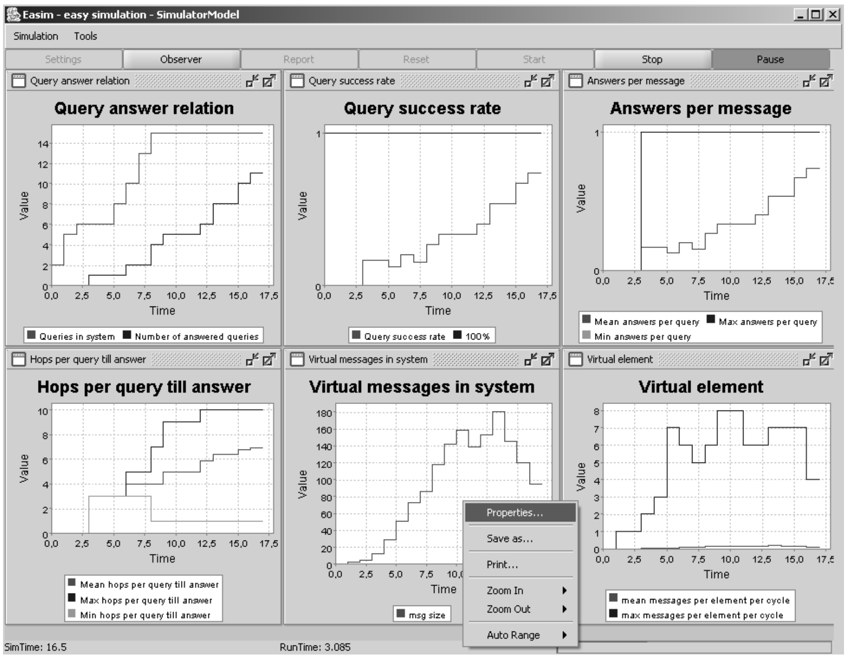


Fig. 12. Each selected metric can be evaluated and is displayed immediately at runtime by JFreeChart (2006). The status bar shows the simulation time, the real execution time and the progress of the current simulation run.

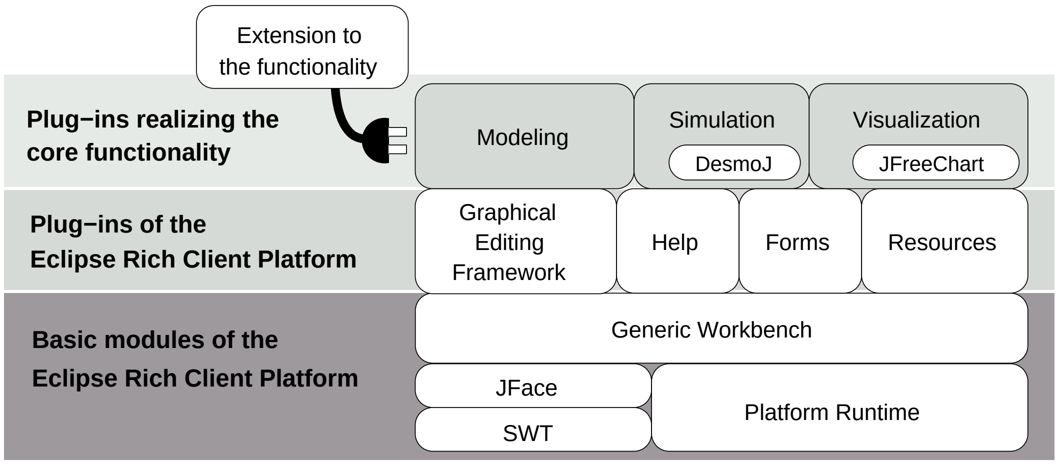

Extension to the functionality
Plug−ins realizing the core functionality
Modeling
Simulation
DesmoJ
Visualization
JFreeChart
Plug−ins of the Eclipse Rich Client Platform
Graphical Editing Framework
Help
Forms
Resources
Basic modules of the Eclipse Rich Client Platform
Generic Workbench
JFace
SWT
Platform Runtime

Fig. 13. Layered architecture of our future modeling and simulation environment. All modules with a white background are re-used components. The Eclipse Rich Client Platform was chosen since it provides a powerful foundation. Its plug-in concept is used internally to separate the three parts of the program and is also used to provide extensibility of the application. The modeled architectures, scenarios, and the simulation results are managed by the resource plug-in. A state-chart-like editor is implemented with the Graphical Editing Framework. Furthermore, DESMO-J is used as a simulation framework and charts are visualized with JFreeChart.

5 Related Work

Simulation-based approaches to the evaluation of peer-to-peer systems can be characterized on the basis of their simulation model. Joseph (2003) divides simulators into four categories: numerical, flow-based, event-based, and packet-based. Related to our work are event-based approaches. Several peer-to-peer projects such as Pastry (FreePastry, 2006; Rowstron and Druschel, 2001) and Freenet (Freenet, 2006) developed their own dedicated simulators. Some examples:

- Joseph (2003) developed the simulator NeuroGrid to compare Freenet, Gnutella, and his own protocols. Extensibility in NeuroGrid is achieved by means of an object-oriented framework that can be used via inheritance from abstract classes.
- As a part of the IRIS project (IRIS, 2006) the peer-to-peer simulator p2psim was developed. The creation of new peer-to-peer protocols is supported to implement join and lookup methods. So far, p2psim supports various structured peer-to-peer protocols like Chord (Stoica et al., 2001), Tapestry (Zhao et al., 2004), and Kademlia (Maymounkov and Mazieres, 2002). Other architectural structures are not supported yet.

Although these systems allow to simulate different peer-to-peer protocols, they are not primarily concerned with architectural styles. They focus on concrete systems and do not address the underlying architectural styles.

Some more general approaches to simulating peer-to-peer systems exist:

- Ting and Deters (2003) designed a layered time-discrete peer-to-peer simulator and aimed to increase extensibility and usability based on an evaluation of several existing simulators. A solution to these problems is proposed by a layered approach with network, protocol, and user layers. Message routing and processor/network delay modeling are supported.
- García et al. (2005) developed PlanetSim, an object-oriented Java-based simulation framework for overlay networks. They follow a hierarchical three level approach with physical, virtual, and application layers. The latter layer is defined by a shared API (Dabek et al., 2003) to offer a well-formed interface.

These generic approaches could be used to simulate different architectural styles, but so far there exists no publications on such work.

6 Conclusions and Future Work

We presented a new approach to simulation-based comparative evaluation of peer-to-peer architectural styles. With respect to peer-to-peer systems, we identified basic characteristics of various peer-to-peer architectural styles.

Exemplarily, we illustrated our simulation approach by means of two case studies that evaluate different architectural styles with respect to (1) their influence on availability of resources and (2) performance of peer-to-peer search algorithms. The two studies satisfied predefined expectations and gave insight into simulation of peer-to-peer systems for assessment of design decisions on architectural styles.

With respect to tool support, we present the eaSim tool. Based on our experience with this and other simulation environments, we indicate our future tool support for evaluating peer-to-peer architectural styles. Based on the Eclipse platform (Eclipse, 2006), we are developing an integrated simulation environment to support our simulation with appropriate tools. This new tool allows for simulating architectural changes at runtime. With our new tool architecture, we are able to re-use a lot of functionality of the Eclipse platform, of DESMO-J and of JFreeChart. Other Eclipse-based environments can also integrate our plug-ins, which may enlarge the potential user community significantly.

Future work on evaluation of peer-to-peer architectural styles will emphasize the simulation of architectural changes at runtime, based on our hitherto existing experience with different simulation systems. The characteristics for various architectural styles will be further elaborated and investigated.

References

Avižienis, A., Laprie, J.-C., Randell, B., Landwehr, C. E., 2004. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1 (1), 11–33.

Bernstein, P. A., Hadzilacos, V., Goodman, N., 1987. Concurrency Control and Recovery in Database Systems. Addison-Wesley.

Bhagwan, R., Savage, S., Voelker, G. M., 2003. Understanding availability. In: Peer-to-Peer Systems II, Second Intl. Workshop. Vol. 2735 of Lecture Notes in Computer Science. Springer, pp. 256–267.

Bischofs, L., Steffens, U., 2005. Organisation-oriented super-peer networks for digital libraries. In: Agosti, M., Schek, H.-J., Türker, C. (Eds.), Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures: 6th Thematic Workshop of the EU Network of Excellence DELOS, S. Margherita di Pula, Cagliari, Italy, 24-25 June, 2004. Revised Selected Papers. Vol. 3664 of Lecture Notes in Computer Science. Springer, pp. 45–62.

Clarke, I., Sandberg, O., Wiley, B., Hong, T. W., 2000. Freenet: A distributed anonymous information storage and retrieval system. In: Federrath, H. (Ed.), Proceedings of the Workshop on Design Issues in Anonymity and Unobservability. Vol. 2009 of Lecture Notes in Computer Science. Springer, pp. 46–66.

Dabek, F., Zhao, B. Y., Druschel, P., Kubiatowicz, J., Stoica, I., 2003. Towards a common API for structured peer-to-peer overlays. In: Kaashoek, M. F., Stoica, I. (Eds.), Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Revised Papers. Vol. 2735 of Lecture Notes in Computer Science. Springer, pp. 33–44.

eaSim, 2006. easim - p2p simulator. Last checked on 2006-01-25.
URL `http://sourceforge.net/projects/easim`

Eclipse, 2006. Eclipse. Last checked on 2006-01-23.
URL `http://www.eclipse.org/`

Freenet, 2006. The Free Network Project. Last checked on 2006-01-23.
URL `http://freenet.sourceforge.net`

FreePastry, 2006. Pastry – a substrate for peer-to-peer applications. Last checked on 2006-01-23.
URL `http://freepastry.org/`

Gamma, E., Helm, R., Johnson, R., Vlissides, J., Mar. 1995. Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley.

García, P., Pairot, C., Mondéjar, R., Pujol, J., Tejedor, H., Rallo, R., 2005. Planetsim: A new overlay network simulation framework. In: Software Engineering and Middleware, 4th International Workshop, SEM 2004, Linz, Austria, September 20-21, 2004, Revised Selected Papers. Vol. 3437 of Lecture Notes in Computer Science. Springer, pp. 123–136.

Garlan, D., Allen, R., Ockerbloom, J., 1994. Exploiting style in architectural design environments. In: SIGSOFT '94: Proceedings of the 2nd ACM SIGSOFT symposium on Foundations of software engineering. ACM Press, pp. 175–188.

Gifford, D. K., 1979. Weighted voting for replicated data. In: SOSP '79: Proceedings of the Seventh ACM Symposium on Operating Systems Principles. ACM Press, pp. 150–162.

Gnutella, 2006. Gnutella. Last checked on 2006-01-23.
URL `http://www.gnutella.com/`

IEEE, 2000. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE, IEEE Standard 1471-2000.

IRIS, 2006. Iris: Infrastructure for resilient internet systems. Last checked on 2006-01-23.
URL `http://project-iris.net/`

ISO/IEC 14598-1, 1999. ISO/IEC 14598-1: Information technology – Software product evaluation – Part 1: General overview. ISO/IEC, published standard.

JFreeChart, 2006. Jfreechart. Last checked on 2006-01-25.
URL `http://www.jfree.org/jfreechart`

Joseph, S., Nov. 2003. An extendible open source P2P simulator. P2P Journal, 1–15.

JXTA, 2006. JXTA. Last checked on 2006-01-23.
URL `http://www.jxta.org/`

Kazaa, 2006. Kazaa. Last checked on 2006-01-23.
URL `http://www.kazaa.com/`

Knuth, D. E., 1997. The Art of Computer Programming, Volume 2: Seminumerical algorithms. Addison-Wesley.

Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S., 2002. Search and replication in unstructured peer-to-peer networks. In: ICS '02: Proceedings of the 16th international conference on Supercomputing. ACM Press, pp. 84–95.

Maier, M. W., Emery, D., Hilliard, R., 2001. Software architecture: Introducing IEEE Standard 1471. Computer 34 (4), 107–109.

Maymounkov, P., Mazieres, D., 2002. Kademlia: A peer-to-peer information system based on the XOR metric. In: Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers. Vol. 2429 of Lecture Notes in Computer Science. Springer, pp. 53–65.

Medvidovic, N., Taylor, R. N., 1998. Separating fact from fiction in software architecture. In: Proc. 3rd Intl. Software Architecture Workshop (ISAW3). ACM Press, pp. 105–108.

Milgram, S., 1967. The small world problem. Psychology Today (2), 60–67.

Mizrak, A. T., Cheng, Y., Kumar, V., Savage, S., 2003. Structured superpeers: Leveraging heterogeneity to provide constant-time lookup. In: WIAPP '03: Proceedings of the The Third IEEE Workshop on Internet Applications. IEEE Comp. Soc. Pr., Washington, DC, USA, pp. 104–111.

Napster, 2006. Napster. Last checked on 2006-01-23.
URL `http://www.napster.com/`

Page, B., Kreutzer, W., 2005. The Java Simulation Handbook: Simulating Discrete Event Systems with UML and Java. Berichte aus der Informatik. Shaker.

Rowstron, A. I. T., Druschel, P., 2001. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (Ed.), Middleware 2001, IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12-16, 2001, Proceedings. Vol. 2218 of Lecture Notes in Computer Science. Springer, pp. 329–350.

Schollmeier, R., Aug. 2001. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In: First Intl. Conf. on Peer-to-Peer Computing (P2P 2001). IEEE Comp. Soc. Pr., pp. 101–102.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H., 2001. 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, pp. 149–160.

Thomas, R. H., 1979. A majority consensus approach to concurrency control for multiple copy databases. ACM Trans. Database Syst. 4 (2), 180–209.

Ting, N. S., Deters, R., 2003. 3LS – a peer-to-peer network simulator. In: P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing. IEEE Comp. Soc. Pr., pp. 212–213.

Wang, C., Li, B., 2003. Peer-to-peer overlay networks: A survey. Tech. rep., Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong.

Watts, D. J., Strogatz, S. H., Jun. 1998. Collective dynamics of small-world networks. Nature 393, 440–442.

yEd, 2006. yed - Java Graph Editor. Last checked on 2006-01-25.
URL `http://www.yworks.com/`

Zhao, B. Y., Huang, L., Rhea, S. C., Stribling, J., Joseph, A. D., Kubiatowicz, J. D., Jan. 2004. Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications 22 (1), 41–53.