Quantification of interacting runtime qualities in software architectures: insights from transaction processing in client-server architectures

Anakreon Mentis[1] *Panagiotis Katsaros*[1] *Lefteris Angelis*[1]

George Kakarontzas[2]

Department of Informatics

Aristotle University of Thessaloniki

54124 Thessaloniki, Greece

tel.: +30-2310-998532, fax: +30-2310-998419

[1] {anakreon, katsaros, lef}@csd.auth.gr

[2] gkakaron@teilar.gr

Abstract

Architecture is fundamental for fulfilling requirements related to the non-functional behavior of a software system such as the quality requirement that response time does not degrade to a point where it is noticeable. Approaches like the Architecture Tradeoff Analysis Method (ATAM) combine qualitative analysis heuristics (e.g. scenarios) for one or more quality metrics with quantitative analyses. A quantitative analysis evaluates a single metric such as response time. However, since quality metrics interact with each other, a change in the architecture can affect unpredictably multiple quality metrics. This paper introduces a quantitative method that determines the impact of a design change on multiple metrics, thus reducing the risks in architecture design. As a proof of concept, the method is applied on a simulation model of transaction processing in client-server architecture. Factor analysis is used to unveil latent (i.e. not directly measurable) quality features represented by new variables that reflect architecture-specific correlations between metrics. Separate Analyses of Variance (ANOVA) are then applied to these variables, for interpreting the tradeoffs detected by factor analysis in terms of the quantified metrics. The results for the examined transaction processing architecture show three latent quality features, the corresponding groups of strongly correlated quality metrics and the impact of architecture characteristics on the latent quality features.

KEYWORDS: Software quality, Software architecture, Runtime software metrics, Design tradeoffs, Architecture validation, Transaction processing, Simulation

1. Introduction

Software architecture is defined as: "*the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them*" [8]. Architecture sets the boundaries for runtime quality (performance, fault handling, shared resource usage, security etc.) although it cannot be a basis for precise predictions [17], since runtime behavior also depends on implementation details. Nonetheless, software architecture can support analyses that provide confidence for the effects of a design decision (e.g. replication) on quality metrics such as reliability.

We have seen the emergence of methods for analysing quality in software architectures like for example ATAM [14], SBAR [10], SAAM [26] and HoPLAA [42]. All methods combine qualitative analysis heuristics, such as questioning techniques and use cases, with quantitative analyses specific to particular metrics. However, a single change in the software architecture may affect multiple metrics, due to interactions between them [29].

We propose a quantitative method for discovering architecture-specific metric correlations that are affected by the combined effect of architecture characteristics (e.g. level of concurrency, degree of process distribution etc.). These correlations are justified by *latent quality features*, which cannot be measured directly, but they can assist in managing potentially complex tradeoffs.

As a proof of concept, the method is applied on a simulation-based evaluation of a transactional architecture that complies with the Process Coordinator pattern [21]. The pattern is commonly used to implement business processes that issue requests to several server components. Issues like decomposition of the software functionality, allocation of shared resources and communication among the architecture's components form a complex and interesting design problem. Furthermore, experience reports [8] indicate large variations in server availability, performance and scalability of distributed transaction management. The quality metrics investigated in this work are also important for other architecture patterns like the Broker and the Publish-Subscribe pattern [21].

In the first stage of the proposed method, *factor analysis* is performed. Factor analysis reveals groupings of highly correlated metrics and estimates new and fewer uncorrelated variables that represent the latent quality features. Members of these groupings are positively or negatively correlated. For the transactional architecture at hand, factor analysis is applied on data obtained by a series of experimental runs of the ACID Sim Tools [1, 36] simulator, developed by the authors.

In the second stage, separate Analyses of Variance (ANOVA) are performed for each latent quality feature.

The proposed method is a systematic way for relating the variability of quality metrics and the implied tradeoffs to specific architecture characteristics. The analysis uses data that can be obtained from an appropriate simulation environment [4, 5] or a benchmarking prototype [7, 9, 55]. It is based on minimal assumptions for the data distributions, irrespective of the analyzed quality metrics and the architecture characteristics that are taken into account. We believe that the method is applicable to all architecture patterns where the investigated metrics matter and that it is effective in any architecture with interacting quality metrics. However, the latter remains to be confirmed in future research work.

Section 2 describes the concerns of architecture tradeoff analysis and the published related work. Section 3 provides the process of the proposed method. Section 4 introduces the metrics of interest in transactional client-server architecture, as well as a synthetic transaction processing workload that utilizes the servers to a considerable extent. Section 5 shows how the first stage of the method is performed. Section 6 shows the application of ANOVA for detecting the architecture characteristics that are critical for the investigated quality metrics. The paper concludes with a brief discussion on the benefits of the method and the future research prospects.

2. Architecture tradeoff analysis and related work

In a software project, the architecture is determined in the early design phase [23, 33, 48], where the cost to fix an error is orders of magnitude less than in later development phases [54]. An *architecture validation process* [21] evaluates design decisions with respect to possibly prioritized and conflicting quality requirements. Fulfillment of a quality requirement has positive or negative effects on other metrics, due to interactions between them. Design decisions that violate a quality requirement are identified as architectural *risks*, while those that improve quality metrics without violating a requirement are identified as "good" decisions [22]. A decision is considered as an architectural risk or an improvement, based on assumptions for the impact on the system's behavior. For example, in a Process Coordinator architecture, the designer assumes deterioration of throughput when increasing the degree of process distribution.

In related bibliography, there are attempts (e.g. [11] and [19]) to tabulate the effects of the different metrics on each other. Most of these tables are developed by logical reasoning and experience, rather than existing evidence on the metrics interaction in some specific architecture. In [45], the authors argue that metric dependencies are a property of the architecture. Reliability and performance, for example, can be in conflict in a design that supports a checksums and retry pattern, but they are simultaneously optimized in a replication-based architecture.

Quantitative techniques should address the fact that metric dependencies are architecture-specific and should provide *evidence* for the existing tradeoffs. They should be general enough in order to be *applicable in diverse architectures*, without being constrained by the specific quality metrics that are analyzed simultaneously. Finally, it is desirable to attribute the variability of the metrics to specific architecture characteristics. This allows identifying what in ATAM [14] is called a sensitivity point or a tradeoff point. A *sensitivity* *point* represents a key architectural decision, which can be a property of one or more components (and/or component relationships [15]), that is *critical* for fulfilling a quality requirement. A *tradeoff point* is a property that affects multiple quality metrics and in fact represents one of the most critical decisions in an architecture design.

Predictable Assembly from Certifiable Components (PACC) [39] is a framework for the evaluation of the ability of a system to meet quality requirements. It supports quantitative evaluation for different metrics, provided that an appropriate analytic theory is available in a form called reasoning framework. Our proposal aims to identify sensitivity and tradeoff points rather than building a reasoning framework for a specific metric. From this point of view, the two approaches focus on supplemental concerns.

Software Performance Engineering (SPE) [49] is a process that uses queuing network models to identify architecture alternatives that meet performance objectives. Evaluation of trade-offs between performance and other quality metrics is beyond the scope of SPE. Furthermore, our method does not presume a specific type of model as opposed to SPE, which is based on queuing network analysis.

Table 1 summarizes the main characteristics of representative quantitative tradeoff analyses found in the bibliography. We refer to a hybrid mathematical programming and analytic approach, a simulation-based measurement technique and a simulation-based statistical analysis that focus on different architecture tradeoff problems. Important observations that motivate the proposed method are:

- Existing evaluation techniques quantify only those quality metrics that are used in the tradeoff analysis. This is restrictive, since the technique *cannot detect unanticipated effects of architecture characteristics to other quality metrics*.
- They assume a priori knowledge of a tradeoff point in a software architecture. Moreover, they cannot discover additional sensitivity and tradeoff points.

- It is not possible to attribute the variability of metrics to architecture characteristics other than the tradeoff point, thus ignoring the effects of other architecture characteristics.

Table 1: Characteristics of quantitative techniques for architecture tradeoff analysis

Authors, year, reference	Litoiu, Rolia, Serazzi 2000 [32]	Paul, Gupta, Badrinath 2003 [44]	Katsaros, Angelis, Lazos 2007 [25]
Scope of quantitative analysis	Distributed application systems	Checkpointing and recovery protocols	Distributed systems with independent checkpointing activities
Tradeoff points	Process replication or threading levels and process activation policies.	Checkpoint intervals	Checkpoint intervals for the independent checkpointing activities.
Runtime qualities	Utilization of processes and devices.	Overhead due to checkpointing and recovery activities and quality of recovery.	Response times for the fault-affected and the fault-unaffected computations.
Aim of quantitative analysis	To determine process replication or threading levels, in order to avoid unnecessarily queuing delays for clients and unnecessarily high consumption of memory.	To assess the protocols' performance in different execution environments.	To determine checkpoint intervals fulfilling the response-times goals at the lowest possible fault-tolerance cost. Trades the gains of a potential improvement in the quality of recovery against the overhead caused in normal processing.
Analysis approach	Hybrid mathematical programming and analytic evaluation.	Simulation-based measurement.	Simulation-based statistical analysis.
Effects of architecture characteristics on the measured runtime qualities	It only takes into account the architecture properties included in the model. Estimations are produced only for the tradeoff points.	Implicitly takes into account all simulated architecture characteristics.	Implicitly takes into account all simulated architecture characteristics. Detailed estimations are produced only for the tradeoff points.

In the following sections, we introduce a method that overcomes the mentioned limitations and provides evidence for architecture tradeoffs. At early design stages, the method supports evaluation of the initial architecture design. At late stages the derived knowledge for metric interactions can be used to fine-tune the system's runtime behavior.

We assume a basis for experimentation provided by a simulation model [4, 5] or a benchmarking prototype [7, 9, 51, 53, 55]. However, the statistical approach is general, in the sense that it is not bound to a particular simulation or to some quantification theory [6] specific to a metric.

3. The proposed method in the context of an architecture process

A process for the software architecture includes phases which collectively contribute to the creation and validation of the architecture against the functional and quality requirements.

Gorton in [21] provides an architecture process (Figure 1) which includes three phases: (i) determination of architectural requirements, (ii) architecture design and (iii) validation.

These three phases are repeated iteratively, since validation may lead to unsatisfactory results, in which case the architect needs to reconsider the architecture design and/or refine or modify the requirements.

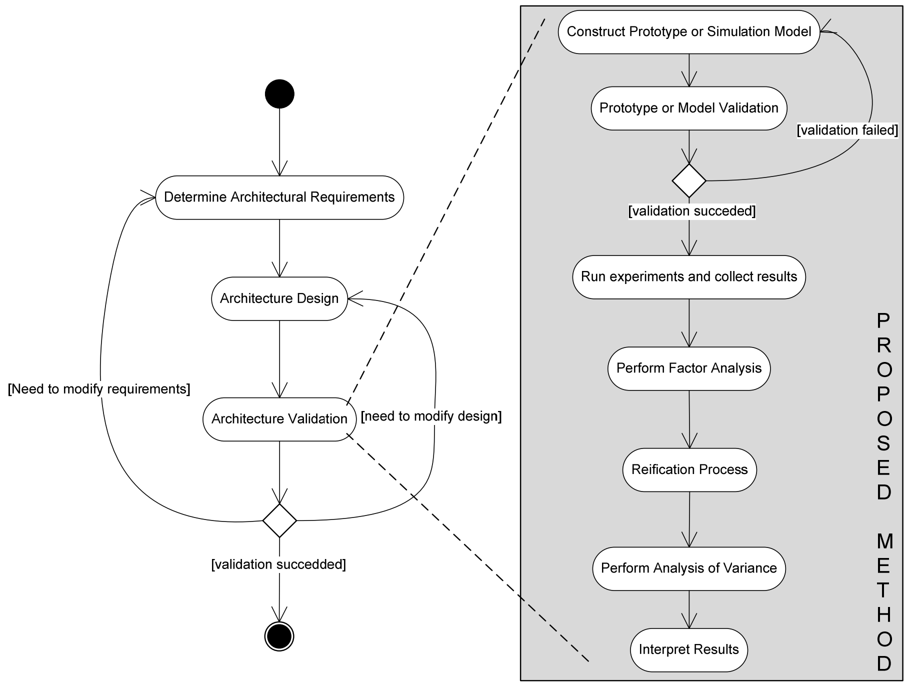


Figure 1: The proposed method in the context of the generic architecture process

The proposed method is applied to the third phase of this process, namely to the validation phase and the method’s steps are shown on the right side of Figure 1. Other concerns like the application of architectural patterns or tactics to improve the architecture design [54], although important, are not directly covered by our proposal. They constitute complementary issues, in the sense that they affect and are affected by the architecture validation. When an architecture design fails to meet the quality requirements the design should be modified. In this case, the proposed method highlights certain quality problems to consider.

Figure 1 shows the steps of the method, in the context of the architecture process:

1. Construction of the prototype or the simulation model.
2. Validation of the prototype or the simulation model: Includes all necessary activities that assure an accurate representation of the real-world. These activities are specific to the adopted prototype or simulation model. Approaches for ensuring validity of simulation models are discussed in [18] and [35].
3. Experimental runs and collection of results for the subsequent statistical analysis.
4. Application of factor analysis to the quality metrics measured in step 3 for estimating new variables representing the latent quality features.
5. Reification Process: It is the process of interpreting axes in multidimensional space [30]. The process is meaningful in principal component analysis and in factor analysis, where new latent variables are extracted. Latent quality features found in the previous step are interpreted.
6. Application of ANOVA for each new variable of step 4, in order to study and interpret design tradeoffs in terms of the quality metrics.
7. General interpretation of the results: The results of the architecture validation are discussed and decisions are made regarding the architecture characteristics which affect the metrics.

The involved roles that interact and cooperate in the various steps of the method are: the *architect*, the *prototype developer* or *modeling expert* and the *statistical expert*. Other stakeholders that provide additional information may be involved, especially during results interpretation, when the architect determines the appropriate course of action.

4. Transactional software architecture and the ACID Sim Tools simulator

In a transaction processing architecture, like the Enterprise JavaBeans [13], clients issue transaction requests to application servers that support a range of configuration and deployment options. A server maintains a repository of components with encapsulated objects and provides their runtime environment. In this environment objects may invoke methods of other objects residing in the same or in another component or server.

A transaction is a program execution that reads and/or modifies the state of persistent objects via the invoked object methods [34]. It consists of a sequence of methods executed on one or more servers. Invoked methods incur computations with specific CPU time demands. A transaction is either committed or is aborted because of inconsistencies in objects state, potential deadlocks or overload conditions. For deadlock resolution, a timeout policy is often applied where a timer is activated upon initiation of a transaction and when the timer expires the transaction is aborted. The objects cooperate with a transaction manager to provide system-wide Atomicity, Consistency, Isolation and Durability (ACID) guarantees for the performed operations.

Typical quality requirements for a successful transactional architecture design are:

- High number of committed transactions per time unit.
- Short response times for committed transactions, where response time is the time span between issuing the transaction request and the instant of transaction commit.
- High server availability.

Transactional applications involve a series of complex architecture tradeoffs that make evident the need for systematic quality design. For example, high availability depends on the recovery time after server failures, which is determined by the size of the log files. A factor that affects the log file size is the frequency of log truncations, called checkpoints. A relatively small log file, which is important for high availability, indirectly reduces the number of committed transactions per time unit, due to increased contention for the available I/O bandwidth. Regarding the locks on the objects involved in aborted transactions, they are withheld and therefore are inaccessible to other transactions until the last valid object state is retrieved from the log file. Thus, it is important to have fast rollbacks of aborted transactions, which is achieved by relatively small log files. But as we already noted small log files are associated with increased I/O contention and this certainly delays the retrieval of log file records.

The architecture design is further complicated by lock contention, which is susceptible for *thrashing phenomena*. Too many transactions running concurrently result in a situation where most transactions are blocked, because of lock conflicts and only few transactions remain active. The maximum number of concurrently executing transactions (multiprogramming level) is used to delimit the degree of concurrency. A low multiprogramming level prevents thrashing but may lead to CPU underutilization and lower throughput.

Checkpoint frequency, multiprogramming level and transaction timeout policy are examples of architecture characteristics that affect one or more quality metrics. A quantitative evaluation technique should provide the means to quantify the significance and the combined effect of these characteristics in different workload scenarios (e.g. local to highly distributed transactions, I/O bound or CPU bound resource contention).

4.1 The ACID Sim Tools simulator

ACID Sim Tools is a toolset that integrates facilities for building, validating and controlling the execution of simulation models for transaction processing architectures. It is based on a minimal set of assumptions that represent an object-based computational model like the OMG Core Object Model [40]. It extends the Objective Modular Network Testbed (OMNeT++) simulation framework [43, 52] and includes:

- A model definition language and a graphical user interface for the specification of the architecture and the associated input parameters;
- Class hierarchies that combined with the code generation functionality allow specification, verification [37] and rapid implementation of simulations for new transaction models;
- Embedded debugging functions and an OMNeT++ graphical user interface that supports interactive simulation and model inspection.

The architecture designer combines a transaction protocol with a concurrency control protocol and simulates the resulting architecture with different parameters. The implemented transactions protocols are the two-phase commit (2PC) variants named presume nothing (PRN), presume commit (PRC) and presume abort (PRA). For concurrency control the tool provides strict two-phase locking and basic timestamp ordering. A simulation model consists of a set of *modules* that communicate by exchanging messages:

- *QSource* - generates arrivals of transaction requests for a specific transaction class. A transaction class represents a sequence of invoked methods that corresponds to an execution path in a program.
- *Atomic Commit Processing* (Acp) - simulates a transaction manager that acts either as a transaction coordinator or a transaction worker. The latter participates in a

transaction and the former collects the workers votes and decides the transaction outcome.

- *Log-Manager* - simulates the stable storage of a transactional server, i.e. the I/O costs for storage - retrieval of object states and the status of processed transactions
- *Concurrency Control* - simulates the concurrency control and isolation guarantees [2]. As in [16], we do not consider resource consumption for concurrency control, since its CPU demands cannot be easily quantified.

A transactional server consists of an Acp, a Log-Manager and a Concurrency Control module. Different implementations can be provided for each module.

The simulation tool supports metrics for systematic experimentation with the tradeoffs between performance and recovery costs (e.g. suspended processing of transactions arrived during server recovery and in workers awaiting the decision for incomplete transactions). The evaluated metrics are:

- server availability;
- percentage of committed transactions - termed as throughput - for (i) distributed transactions, (ii) local transactions and (iii) groups of transaction classes that content for a shared lock;
- mean response times for the aforementioned groups;
- mean blocking times (the time span between the voting of a worker and the receipt of the coordinator's decision), for the same groups.

ACID Sim Tools supports the method of independent replications, for simulation output analysis. The number of simulation runs is determined dynamically, based on the required confidence interval half-width.

4.2 A synthetic transaction processing workload for a case application scenario

In order to demonstrate the effectiveness of the proposed method in exploring the variability and the interactions between quality metrics, we introduce a synthetic transaction workload with relatively high CPU and I/O resource utilization. Other workloads of interest are scenarios with mainly local or highly distributed transactions, as well as scenarios with I/O bound or CPU bound resource contention.

Table 2: CPU time demands of methods and state sizes of persistent objects

Server	Object	State Size (Kb) - exponential -	Method Name	CPU demands (sec) - exponential -	READ/WRITE
acp1	obj1	5	meth111	0.01	READ
			meth112	0.05	READ
	obj2	5	meth121	0.01	READ-WRITE
			meth122	0.01	READ
			meth123	0.01	READ
	obj3	5	meth131	0.04	READ-WRITE
			meth132	0.01	READ
	obj4	5	meth141	0.01	READ-WRITE
			meth142	0.01	READ
			meth143	0.01	READ
	obj5	5	meth151	0.01	READ-WRITE
			meth152	0.01	READ
			meth153	0.01	READ
acp2	obj6	5	meth211	0.05	READ-WRITE
			meth212	0.05	READ
	obj7	5	meth221	0.05	READ-WRITE
			meth222	0.01	READ
	obj8	5	meth231	0.01	READ
			meth232	0.01	READ
	obj9	5	meth241	0.05	READ-WRITE
			meth242	0.05	READ
	obj10	5	meth251	0.05	READ-WRITE
			meth252	0.01	READ
	obj11	5	meth261	0.01	READ
			meth262	0.01	READ
	obj12	5	meth271	0.05	READ-WRITE
			meth272	0.05	READ
	obj13	5	meth281	0.05	READ-WRITE
		5	meth282	0.01	READ

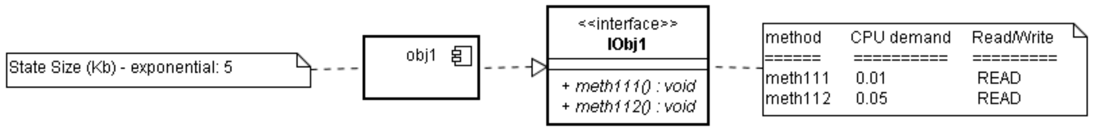


Figure 2: Component interface for obj1 in UML

The selected workload refers to a hypothetical case study, which is the norm in the transaction processing field, even for industrial benchmarking workloads like TPC-C [46]. It uses a number of persistent objects, implemented as components, which are distributed in two servers as shown in Table 2. Components provide their operations through interfaces, such as the component for `obj1` shown in Figure 2.

Table 2 specifies the state size of the objects residing in the two servers, the means of exponentially distributed CPU time demands for the provided methods and whether the methods update their object state.

Table 3: Transactional server system parameters

Network Latency / message:		0.06 sec	
Server	Disk Read Latency	Disk Write Latency	Mean Time To Repair
acp1	4.271e-05 sec/Kb	51.252e-05 sec/kb	4 sec
acp2	4.271e-05 sec/Kb	51.252e-05 sec/kb	4 sec

System-related parameters (Table 3) describe the runtime environment where the transactional architecture is deployed. Disk I/O latency is set based on specifications of commercial products. Fixed network latency is assumed since the two servers exchange only small-size control messages. Communication between servers is considered reliable, i.e. message delivery is guaranteed, which is a Quality of Service possibility foreseen in [41].

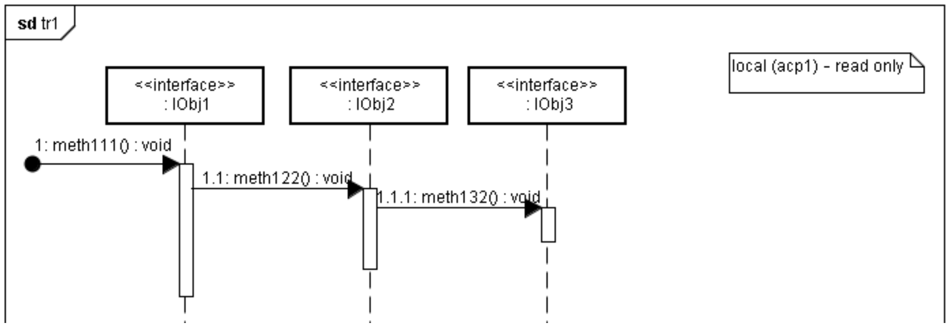


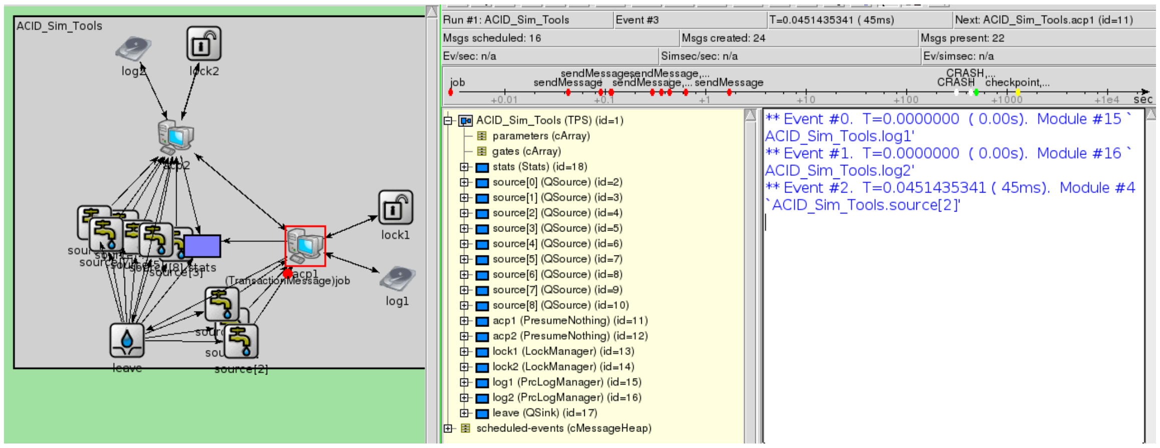
Figure 3: UML sequence diagram for transaction tr1

In the simulated flat transaction model, object methods are invoked sequentially, as shown in the UML sequence diagram for the first transaction class depicted in Figure 3. The synthetic workload includes nine transaction classes, which are given in Table 4 along with their associated parameters.

Table 4: Classes of transaction and object state alterations

Transaction Class	Methods invoked	Characteristics
tr1	meth111, meth122, meth132	local (acp1) – read only
tr2	meth111, meth222, meth112	distributed – read only
tr3	meth112, meth211, meth121	distributed – read/write
tr4	meth242, meth252, meth232	local (acp2) – read only
tr5	meth242, meth142, meth242	distributed – read only
tr6	meth242, meth141, meth251	distributed – read/write
tr7	meth262, meth272, meth282	local (acp2) – read only
tr8	meth262, meth152, meth262	distributed – read only
tr9	meth262, meth151, meth271	distributed – read/write

The modules used to simulate the described transaction workload appear in Figure 4 as they are visualized in the OMNeT++ interactive simulation environment.



```
Run #1: ACID_Sim_Tools    Event #3    T=0.0451435341 ( 45ms)    Next: ACID_Sim_Tools.acp1 (id=11)
Msgs scheduled: 16    Msgs created: 24    Msgs present: 22
Ev/sec: n/a    Simsec/sec: n/a    Ev/simsec: n/a
```

```
ACID_Sim_Tools (TPS) (id=1)
  parameters (cArray)
  gates (cArray)
  stats (Stats) (id=18)
  source[0] (QSource) (id=2)
  source[1] (QSource) (id=3)
  source[2] (QSource) (id=4)
  source[3] (QSource) (id=5)
  source[4] (QSource) (id=6)
  source[5] (QSource) (id=7)
  source[6] (QSource) (id=8)
  source[7] (QSource) (id=9)
  source[8] (QSource) (id=10)
  acp1 (PresumeNothing) (id=11)
  acp2 (PresumeNothing) (id=12)
  lock1 (LockManager) (id=13)
  lock2 (LockManager) (id=14)
  log1 (PrcLogManager) (id=15)
  log2 (PrcLogManager) (id=16)
  leave (QSink) (id=17)
scheduled-events (cMessageHeap)
```

```
** Event #0.  T=0.0000000  ( 0.00s).  Module #15 `
ACID_Sim_Tools.log1'
** Event #1.  T=0.0000000  ( 0.00s).  Module #16 `
ACID_Sim_Tools.log2'
** Event #2.  T=0.0451435341 ( 45ms).  Module #4
`ACID_Sim_Tools.source[2]'
```

Figure 4: Architecture and workload representation (left panel) and the OMNeT++ interactive simulation environment (right panel)

For the verification of transaction models we employ the ACID Model Checker utility [37], which uses the state machine based model definition adopted by the ACID Sim Tools.

Specifically, it is verified that: (i) there are no unreachable states and non-terminating execution paths in the state machines corresponding to the worker and the coordinator, (ii) the transition relation is not partially defined, which would cause simulation run-time errors, (iii) workers conform to the decision of the coordinator, (iv) workers and the coordinator reach exactly one of the two possible decisions (commit or abort), (v) a transaction is committed if and only if all workers have voted for commit, (vi) the transaction participants eventually reach a decision, even in the presence of communication and system failures.

Model validation determines if the model is an accurate representation of the real-world *from the perspective of its intended uses* [18]. In order to confirm the utility and effectiveness of the developed model in realizing the anticipated interactions between the quality metrics, a series of experiments was conducted. The experiments include scenarios with: (i) mainly local transactions, (ii) highly distributed transactions, (iii) I/O bound resource contention and (iv) CPU bound resource contention.

Another validity concern is whether the model's structure contradicts the knowledge about the structure of the real system [35]. ACID Sim Tools simulates systems that implement the architectures specified in [13, 41] and as far as we know there are no discrepancies between the simulated models and the referred specifications.

Validation of consistency [35] includes face validity and parameter verification. Face validity tests if the model abstractions result in a recognizable representation of the real system. Parameter verification tests that model parameters and their numerical values have real system equivalents. Regarding face validity, the ACID Sim Tools simulator models all essential interactions between the components of a transaction processing architecture. Also, parameters (e.g. disk read/write latency and network latency) and their numerical values stem from existing vendor specifications or reported measurements.

Detailed guidelines for the use of ACID Sim Tools in the context of the proposed method are provided in the Appendix.

5. Experimentation, factor analysis and reification for latent quality features

In the described software architecture, quality metrics are influenced by a tradeoff between recovery costs and performance. Recovery costs are not directly measurable (a latent quality feature), while performance is expressed by a number of metrics that disclose different aspects of the runtime behavior (Table 5). In addition to the metrics shown, we also quantified the availability of the two servers, under the considered fault load. Transactions that share an object are placed in the groups of the last three columns of Table 5. Objects of transaction tr2 are not accessed by other transactions and consequently tr2 does not appear in any of the three groups.

Table 5: Performance metrics for the synthetic transaction workload

	All distributed transactions	**All local transactions**	**Group 1: transaction classes tr1, tr3**	**Group 2: transaction classes tr4, tr5, tr6**	**Group 3: transaction classes tr7, tr8, tr9**
Throughput (ratio of committed transactions)	tput_distr	tput_local	tput_confl0	tput_confl1	tput_confl2
Mean response time	response_distr	response_local	response_grp0	response_grp1	response_grp2
Mean blocking time	blocking_distr		blocking_grp0	blocking_grp1	blocking_grp2

The proposed method is based on a two-stage *exploratory study* that *aims to identify all potential tradeoff points, with respect to the quality metrics of interest*. Tradeoff points are selected from the considered architecture characteristics based on evidence, provided by factor analysis, for the statistical significance of the detected quality metrics dependencies.

5.1 Experimental design

A series of experiments is conducted, as required by step 3 of the proposed method (refer to section 3), in order to collect data for the quality metrics of Table 5 and the measured server availability. The experimental research for the synthetic workload at hand complies with the guidelines given in [28]. The research questions we posed are:

i. Question 1: *Which are the latent quality features that determine the dependencies between the analyzed quality metrics*?

ii. Question 2: *Which architecture characteristics are the sensitivity and the tradeoff points?*

iii. Question 3: *Given the tradeoff points, which architecture characteristics have statistically significant effects on the latent quality features?*

iv. Question 4: *How are the quality metrics affected by the statistically significant architecture characteristics?*

The experimental context includes the following background information for experimental runs with the ACID Sim Tools, as well as information for the used statistical analysis software: (i) The length of simulation runs for the selected experimental design was determined through repetitive trials, so that the obtained results quantify the metrics, when being in the steady-state. (ii) For all considered cases of fault load (server failures are by definition "rare" events), the model initialization bias was eliminated when the elapsed simulated time was at least 55h 30m. With this preliminary analysis [27] the length of the performed simulation runs was set to the aforementioned value. (iii) The CPU time required for a simulation run in a personal computer with a single-core processor and 1 GB RAM varied between a few minutes to 12 minutes, depending on the used simulation parameters. ACID Sim Tools exhibited a stable behavior, without memory leaks and the memory usage was between 5 % and 50 % of the system's memory. (iv) Data analysis was conducted using the SPSS statistical software, on simulation results obtained for the 14 performance metrics of Table 5 and the two metrics that quantify the servers' availability (in total 16 dependent variables).

Experimental design was based on preliminary simulation runs that were used to explore the variability of the quality metrics across the architecture design space. In all cases, concurrency control adhered to the widely used strict two-phase locking (2PL) scheme. Experiment factors correspond to the investigated architecture characteristics. Table 6 presents the levels of experiment factors that delimit the experimentation area, where we observed significant variation of the analyzed metrics. Three different fault load levels stressed the simulated model with relatively "rare" or more frequent server failures (Poisson process). The two levels of transaction arrival rates (Poisson process) resulted in a radically different system load.

Table 6: Experimental factors that influence recovery costs and performance

Factors (architecture characteristics)	**Level 1**	**Level 2**	**Level 3**
Atomic commit protocol (ACP) - all servers -	Two-Phase Commit Presume Nothing (PRN)	Two-Phase Commit Presume Commit (PRC)	Two-Phase Commit Presume Abort (PRA)
Multiprogramming Level (MPL) - all servers -	2	3	4
Checkpoint intervals (CI) - periodic (sec) for all servers	500	1300	2100
Transaction timeouts (TT) in sec - all transaction classes -	0.9	1.1	1.3
Mean interarrival times (MIT), exp. - all transaction classes -	0.6	0.4	
Mean interarrival time of server fail -stop failures (MITofSF) - exponential	18 m	5 hours 51m	12 hours

Given the factor levels of Table 6, we performed the full factorial experiment including one simulation run for each combination of factor levels, i.e. simulation results were collected from $3 \times 3 \times 3 \times 3 \times 2 \times 3 = 486$ runs. An appropriate alternative for experimentation with less CPU demands is the use of an experimental design with fewer runs, like the *uniform design* that is utilized in [25].

5.2 Factor analysis and reification

Factor analysis (FA) reduces the problem dimensionality by computing new, fewer dependent variables that contain most of the information provided by the simulated metrics.

The idea is to "group" the 16 dependent variables (the output of the experimental runs) in such a way, that the highly correlated variables will be loaded in the same group. These groupings are represented by new variables in the form of "scores", thus reducing the dimensionality of the original data. The new variables resulting from this procedure are uncorrelated and can be studied separately with respect to the experiment factors.

In essence, FA aims to discover - in the original dataset - latent variables termed factors, by exploiting the correlation structure of the dependent variables. To avoid confusion with the term "factor", as it is also used for the experiment's factors of Table 6, the latent variables from now on are called "components", a terminology consistent with the applied method.

Factor analysis requires that the dependent variables are normally distributed and for this reason we applied Blom's transformation [12] that utilizes the ranks r_i of the *n* values (simulation output) and the cumulative Normal distribution function $\Phi^{-1}(x)$. The formula for the Blom transformation is

$$s_i = \Phi^{-1}\left(\frac{r_i - 3/8}{n + 1/4}\right) \qquad (1)$$

and the transformed metrics were found to almost perfectly fit the standard normal distribution with mean 0 and standard deviation 1.

The applied transformation also preserves the correlation structure of the original dataset. For example, in Figure 5 we see the strong negative correlation between *tput_local* and *blocking_distr* in the original (5a) and in the transformed (5b) variables. Moreover, we observe that the normalization (right panel) portrays better the correlation between them.

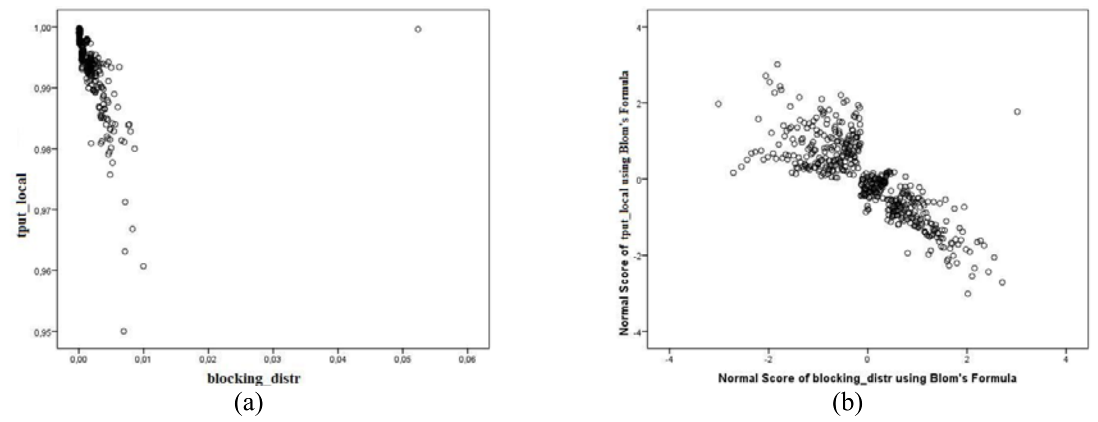


Figure 5. Strong negative correlation between *tput_local* and *blocking_distr* in the original and in the transformed dataset

On the transformed normalized dependent variables we performed a FA procedure based on *Principle Components with Varimax rotation* [20]. Varimax rotation is a procedure which searches for a linear combination of the original components such that the variance of the loadings is maximized. It helps to group the dependent variables since each of them tends to be associated with one (or a small number) of components and at the same time each component represents only a small number of variables.

The new variables, i.e. the component scores, were calculated by the Anderson-Rubin Method [3], which estimates score coefficients. The scores that are produced have a mean of 0, standard deviation of 1, and are not correlated.

The FA resulted in three components that explain 88.81% of the variance of the 16 original variables (metrics). Thus, the dimensions of the output space are reduced significantly by exploiting the correlation structure of the normally transformed outcomes.

Table 7 shows the *loadings* of the variables on the three components and therefore the grouping of the quality metrics. The sign shows the direction of the detected correlation. For example, *tput_local* is loaded to the same component as *blocking_distr* but their loadings have different signs, due to their negative correlation. This is an evidence for the following metric dependency: *longer blocking times for distributed transactions result in lower throughput for locally processed transactions*. The reason is that most withheld locks are managed by blocked distributed transactions and the local transactions are queued in the servers for one or more locks, with increased probability of a transaction timeout. All groupings are presented in Table 8.

Table 7: Rotated component matrix obtained by FA

Normal score using Bloom's Formula	Component		
	C1	C2	C3
availability0	**-.800**	.009	.115
availability1	**-.794**	-.002	.106
tput_local	**-.831**	.132	.406
tput_distr	-.131	-.125	**.978**
tput_confl0	-.166	.098	**.955**
tput_confl1	-.138	-.227	**.950**
tput_confl2	-.140	-.239	**.947**
response_local	.031	**.724**	-.389
response_distr	.095	**.953**	-.203
response_grp0	-.036	**.948**	-.205
response_grp1	-.017	**.975**	.122
response_grp2	-.013	**.989**	.040
blocking_distr	**.967**	.062	-.054
blocking_grp0	**.935**	.068	-.126
blocking_grp1	**.944**	.007	-.034
blocking_grp2	**.948**	.038	-.050

Extraction Method: Principal Component Analysis
Rotation Method: Varimax with Kaiser Normalization

Table 8: Groupings of the quality metrics derived from FA

Component C1 (explains 35.35% of the total variance)	*availability0* (-), *availability1* (-), *tput_local* (-), *blocking_distr* (+), *blocking_grp0* (+), *blocking_grp1* (+), *blocking_grp2* (+)
Component C2 (explains 27.64% of the total variance)	*response_local* (+), *response_distr* (+), *response_grp0* (+), *response_grp1* (+), *response_grp2* (+)
Component C3 (explains 25.83% of the total variance)	*tput_distributed* (+), *tput_confl0* (+), *tput_confl1* (+), *tput_confl2* (+)

The reification process answers to the first posed question (section 5.1) and involves the interpretation of the components derived from FA. Their meanings in the considered transaction processing workload are:

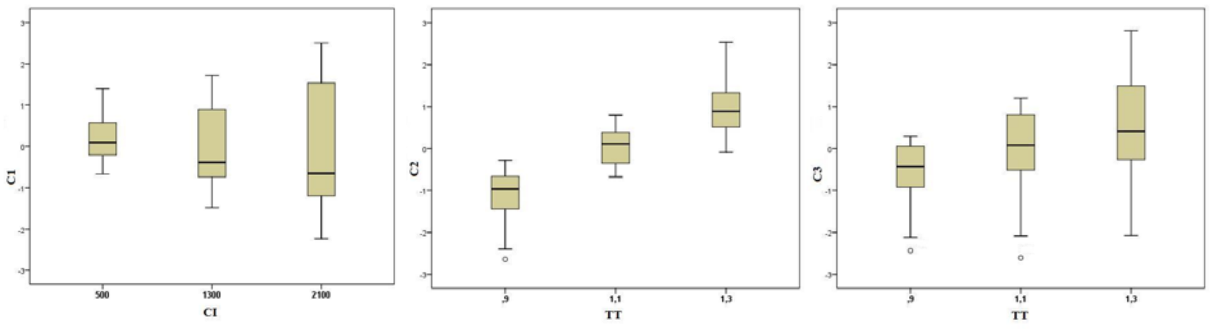
- Component *C1* summarizes the variability of all metrics that are mostly affected by I/O costs, due to checkpoints and recovery. We observe that high values for *C1* are related to high values of blocking times for the distributed transactions and low values of servers' availability and throughput for the locally processed transactions and vise versa.
- Component *C2* summarizes the variability of all response time metrics that were found to be positively correlated with the new variable.
- Component *C3* summarizes the variability of the throughput metrics that seem to be affected mainly by communication latencies. We pinpoint that the throughput for the locally processed transactions was included in component *C1*, as opposed to all other throughput metrics.

6. Analyses of variance for detecting risks related to quality requirements

Sensitivity and tradeoff points are the research focus of the second question stated in section 5.1. They can be initially detected by visual inspection of box-plots, which depict the variability of the three components with respect to each factor. *The impact of the considered factors differs for each component* as shown in Figure 6.

Checkpoint intervals have a significant impact on the quality metrics of *C1* (Figure 6a). This factor is clearly *a sensitivity point* for the servers' availability, the throughput of the locally processed transactions and the blocking times of the distributed transactions, since all these metrics can be simultaneously optimized.

Figures 6b and 6c show that *C2 and C3 increase with increasing values of timeout intervals*. Higher values of C3 correspond to improved throughput while in C2, higher values correspond to longer response times (refer to the correlation signs of Table 8). *Timeout interval affects multiple metrics, which cannot be optimized simultaneously and thus it is a typical tradeoff point*.



(a) (b) (c)

Figure 6. Effects of the experimental factors on the three components

6.1 Analyses of Variance for the three components

For a deeper understanding of the tradeoff points and in order to answer the third research question of section 5.1, we apply Factorial Analysis of Variance (ANOVA) for the three components. ANOVA is used for studying the factor effects to some dependent continuous variable resulting from a multifactor experiment. In our case, we consider each component as dependent variable and for every component a 6-way ANOVA is performed, in order to study the effects of the 6 factors of Table 6 on the component.

ANOVA essentially builds an additive model with the main effects of the factors and their interactions and is accompanied by a number of statistical tests and measures which assist the inference concerning the relation between the factors and the dependent variable (we refer to [20] for a detailed account and their realization in SPSS). More specifically:

- Factors which have significant effect on the dependent variable are assessed by the statistical *F-test*. We consider as statistically significant the effect of a factor or an interaction if the significance of the F-test is less than 0.05.
- The fitting of the ANOVA model, i.e. the proportion of variability of the dependent variable that is explained by the modeled factors is measured by the *R-squared* and the *adjusted R-squared* statistic with values in the [0,1] interval.
- *Partial eta-squared value* shows which effects are the most important for explaining the dependent variable. For each effect (either main or interaction effect), the eta-squared statistic quantifies the degree of association between the effect and the dependent variable and in fact describes the proportion of total variability attributable to a factor.
- *Post hoc tests* are multiple comparison tests that identify statistically significant differences *between the levels* of each experimental factor with respect to the dependent variable.
- *Profile plots* (or *interaction plots*) provide insight into the most important interaction effects. A profile plot is a line plot where each point indicates the estimated marginal mean of a dependent variable at one level of a factor. The levels of a second factor can be used to make separate lines. Each level in a third factor can be used to create a separate plot.

In our analysis we tried for every component (dependent variable) various models starting from a model with the 6 *main* factor *effects*, all possible combinations of unordered pairs of factors (15 two-way interactions) and unordered triples of factors (20 three-way interactions). Interactions of higher order are omitted, because interpretation of the results would be complicated. We present the obtained results for the factors and interactions with F-test value less than 0.05.

6.1.1 ANOVA for component C1

Table 9 shows the statistically significant main and interaction effects for component *C1*. The R-squared statistic implies that this ANOVA model explains 93% of the variability of *C1*. From the partial eta squared statistic we infer that the most important effects are the *MITofSF* (92.1%), the interaction *CI*MITofSF* (71.2%) and the main effects of *CI* (21.3%), *MIT* (14.5%) and *TT* (10.5%). All other effects, although they were found to be statistically significant, explain only small proportions (less than 5%) of the variability of *C1*.

Table 9: Statistically significant effects for component *C1*

Effect	**F-statistic**	**Sig.**	**Partial Eta Squared**
ACP (atomic commit protocol)	5.886	.003	.025
CI (checkpoint intervals)	62.643	.000	.213
TT (transaction timeouts)	27.215	.000	.105
MIT (mean interarrival times)	78.210	.000	.145
MITofSF (mean interarrival time of server fail-stop failures)	2696.656	.000	.921
*CI * MITofSF*	284.876	.000	.712
*CI * MIT*	4.425	.012	.019
*MIT * MITofSF*	4.635	.010	.020
*ACP * MITofSF*	3.499	.008	.029
*TT * MIT*	8.945	.000	.037
R Squared = ,937 (**Adjusted R Squared** = ,934)			

Table 10: Significant differences between factor levels for component *C1*

		500	1300	2100
CI (checkpoint intervals)	500		✓	✓
	1300	✓		
	2100	✓		
		0.9	1.1	1.3
TT (transaction timeouts)	0.9		✓	✓
	1.1	✓		✓
	1.3	✓	✓	

Post hoc tests results are presented in Table 10. Factor levels for the *MIT* and the *MITofSF* are omitted, since we focus only on the examined architecture characteristics. Cells with a tick represent pairs of factor levels which have statistically significant differences. The checkpoint interval of 500 sec differs significantly from the two other levels in terms of *C1* and the same happens with all levels of transaction timeouts.

The profile plot of Figure 7 shows the interaction between the *CI* and *MITofSF* (visualized as non parallel lines), which explains 71.2% of the variability of *C1*. The plot concerns the 2PC PRN protocol and is similar to the omitted plots for the PRC and PRA protocols, since *ACP* does not interact with the two other factors.

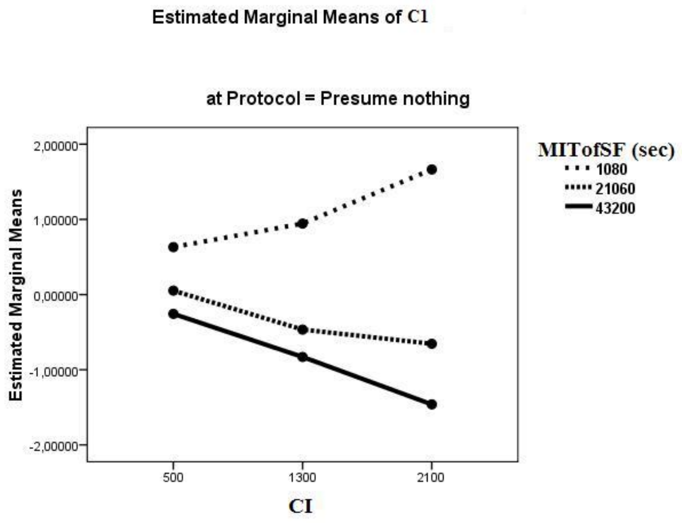


Figure 7. The interaction effect between *CI* and *MITofSF* on component *C1*

CI and *MITofSF* interact as follows. Frequent checkpoints reduce the recovery cost and at the same time increase I/O demand. For rare failures (level 43200 sec) the incurred cost during normal processing is not justified, because it overwhelms the recovery gains. For frequent failures the gains in recovery costs are more significant than the losses in I/O bandwidth thus improving the server availability and the blocking times of the processed distributed transactions.

6.1.2 ANOVA for component C2

Table 11 shows the statistically significant factor effects for component *C2* (response time metrics). The R-squared implies that the model explains 99.5% of the variability of *C2*. All factors have significant main effect, but there are also various significant 2-way and 3-way interactions. The most important effects in terms of the computed partial eta squared statistic are the main effects of *TT* (99.3%), *MIT* (96.5%), *MPL* (95.1%) and their 3-way interaction *MPL*TT*MIT* (60.1%). *ACP* seems to have an important role either as a main effect (40.4%) or in various interactions.

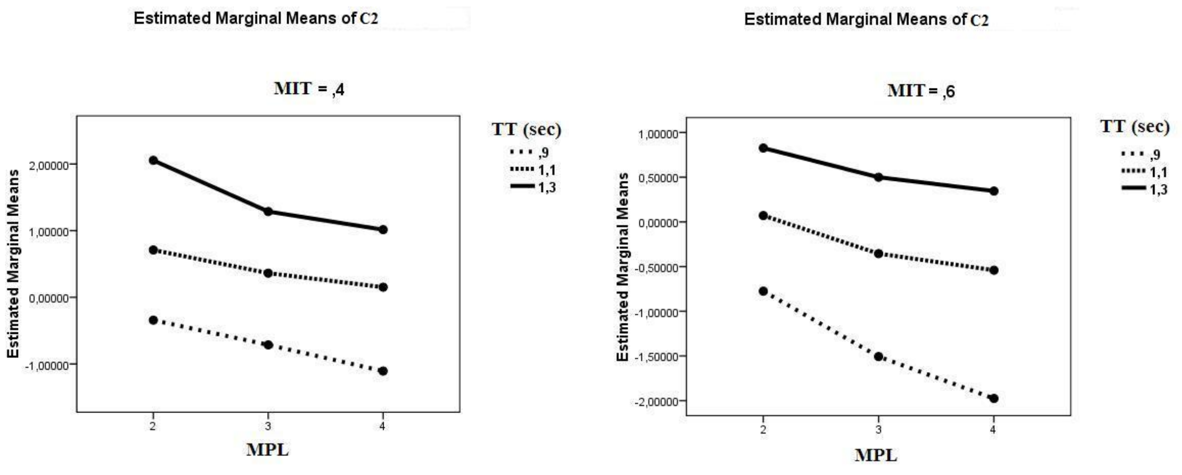
Table 11: Statistically significant effects for component *C2* (response time metrics)

Effect	**F-statistic**	**Sig.**	**Partial Eta Squared**
ACP (atomic commit protocol)	130.303	.000	.404
MPL (multiprogramming level)	3761.003	.000	.951
CI (checkpoint intervals)	18.469	.000	.088
TT (transaction timeouts)	26339.988	.000	.993
MIT (mean interarrival times)	10540.846	.000	.965
MITofSF (mean interarrival time of server fail-stop failures)	37.216	.000	.162
*CI * MITofSF*	5.003	.001	.050
*MIT * MITofSF*	11.016	.000	.054
*ACP * MITofSF*	41.395	.000	.301
*TT * MITofSF*	3.616	.007	.036
*TT * MIT*	84.863	.000	.307
*ACP * TT*	9.833	.000	.093
*MPL * TT*	88.956	.000	.481
*CI * MIT * MITofSF*	4.262	.000	.062
*MPL * CI * MITofSF*	2.227	.010	.065
*ACP * MPL * MITofSF*	2.438	.014	.048
*ACP * TT * MITofSF*	4.526	.000	.086
*ACP * MPL * MIT*	4.241	.002	.042
*ACP * TT * MIT*	2.611	.035	.026
*MPL * TT * MIT*	144.837	.000	.601
*ACP * MPL * TT*	5.764	.000	.107
R Squared = .995 (**Adjusted R Squared** = ,993)			

The results of the post hoc tests in Table 12 suggest that when the used protocol is 2PC PRA, the response times exhibit significant differences compared to the response times in the two other cases. Also, the obtained results for *MPL* and *TT* show that all levels differ significantly, which is in accordance with Figure 6b.

Table 12: Significant differences between factor levels for component *C2* (response time metrics)

		2PC PRN	2PC PRC	2PC PRA
ACP (atomic commit protocol)	2PC PRN			✓
	2PC PRC			✓
	2PC PRA	✓	✓	
		2	3	4
MPL (multiprogramming level)	2		✓	✓
	3	✓		✓
	4	✓	✓	
		.9	1.1	1.3
TT (transaction timeouts)	.9		✓	✓
	1.1	✓		✓
	1.3	✓	✓	



(a) (b)

Figure 8. The interaction effect between *MPL*, *TT* and *MIT* on component *C2*

Figure 8 visualizes the 3-way interaction between the *MPL*, *TT* and *MIT*, which is the most important interaction effect. From the two plots we observe that for *MIT* = 0.4 (Figure 8a) response times are longer than the response times for *MIT* = 0.6 (Figure 8b). We also note a steep descending trend when varying the *MPL* between 2 and 4 and at the same time *MIT* = 0.6 and *TT* = 0.9, meaning that in this case an increase in the number of threads is more effective in achieving shorter response times than it is in all other cases. An important improvement is also evident when increasing *MPL* from 2 to 3 in the case of frequent arrivals (*MIT* = 0.4) and *TT* = 1.3 sec.

6.1.3 ANOVA for component C3

Table 13 shows the ANOVA results for *C3* obtained from a model that explains 99.1% of the component's variability. All factors have significant main effect, along with noteworthy interactions. The most important effects according to the partial eta squared statistic are the main effects of *MIT* (98.5%), *TT* (95.7%), *MPL* (91.0%), *MITofSF* (65.8%) and the 2-way interactions of the factors with the *MIT* and the *MITofSF*.

The performed post hoc tests (Table 14) show that for 2PC PRA, the studied metrics exhibit significant differences when compared with the two other protocols. For the factors *MPL*, *CI* and *TT* all levels differ significantly.

Figure 9 shows the interaction between *TT* and *MIT* that was found having the most significant effect. The plot concerns the 2PC PRN protocol and is very similar to the omitted plots for the PRC and PRA protocols, since *ACP* does not interact with the two other factors. The interesting finding here is that when *TT* is increased from 0.9 to 1.3 and *MIT* is 0.6 there is a much more steep improvement in the metrics of *C3* than the one observed when *MIT* = 0.4. Thus, for improving the metrics of *C3*, an increase in *TT* when having frequent transaction requests is not as effective as it is in the case of rare requests.

Profile plots that are omitted indicate interesting interactions for *the ACP as a sensitivity point of the metrics in C2* and *the CI as a sensitivity point of the metrics in C3*.

Table 13: Statistically significant effects for component *C3*

Effect	**F-statistic**	**Sig.**	**Partial Eta Squared**
ACP (atomic commit protocol)	48.171	.000	.207
MPL (multiprogramming level)	1869.720	.000	.910
CI (checkpoint intervals)	57.664	.000	.239
TT (transaction timeouts)	4126.011	.000	.957
MIT (mean interarrival times)	24767.104	.000	.985
MITofSF (mean interarrival time of server fail-stop failures)	353.506	.000	.658
*CI * MITofSF*	52.601	.000	.364
*CI * MIT*	12.842	.000	.065
*ACP * CI*	5.848	.000	.060
*MPL * CI*	3.535	.008	.037
*CI * TT*	12.760	.000	.122
*MIT * MITofSF*	4.755	.009	.025
*ACP * MITofSF*	46.461	.000	.336
*MPL * MITofSF*	6.558	.000	.067
*TT * MITofSF*	43.777	.000	.322
*MPL * MIT*	311.983	.000	.629
*TT * MIT*	568.012	.000	.755
*MPL * TT*	11.876	.000	.114
*CI * MIT * MITofSF*	2.514	.041	.027
*MPL * CI * MITofSF*	2.959	.003	.060
*CI * TT * MITofSF*	7.165	.000	.135
*ACP * CI * MIT*	2.629	.034	.028
*MPL * CI * MIT*	3.919	.004	.041
*CI * TT * MIT*	3.283	.012	.034
*ACP * MIT * MITofSF*	6.403	.000	.065
*MPL * MIT * MITofSF*	16.472	.000	.152
*TT * MIT * MITofSF*	35.348	.000	.278
*ACP * MPL * MIT*	2.070	.038	.043
*ACP * TT * MIT*	3.220	.001	.065
*MPL * TT * MIT*	14.963	.000	.140
R Squared = .991 (**Adjusted R Squared** = .988)			

In overall, through the shown ANOVA it was possible to identify the statistically significant factor effects and subsequently to interpret them in terms of the way they affect the quality metrics. All important interactions were explored and we commented on possible risks for specific combinations of factor levels. The derived conclusions provide answers to the fourth research question of section 5.1.

Table 14: Significant differences between factor levels for component *C3*

		2PC PRN	2PC PRC	2PC PRA
ACP (atomic commit protocol)	2PC PRN	▨	▨	✓
	2PC PRC	▨	▨	✓
	2PC PRA	✓	✓	▨
		2	3	4
MPL (multiprogramming level)	2	▨	✓	✓
	3	✓	▨	✓
	4	✓	✓	▨
		500	1300	2100
CI (checkpoint intervals)	500	▨	✓	✓
	1300	✓	▨	✓
	2100	✓	✓	▨
		.9	1.1	1.3
TT (transaction timeouts)	.9	▨	✓	✓
	1.1	✓	▨	✓
	1.3	✓	✓	▨

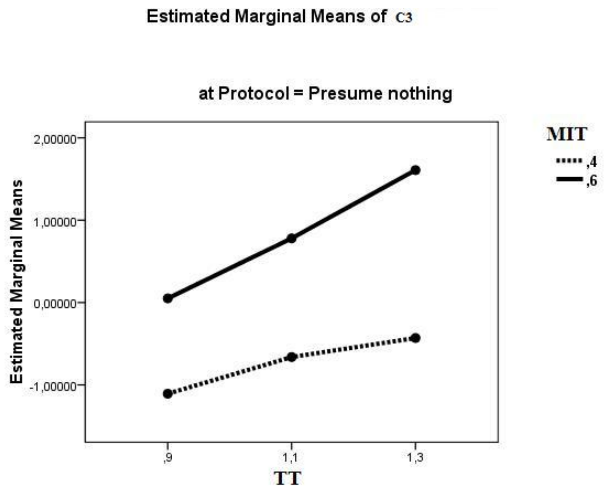
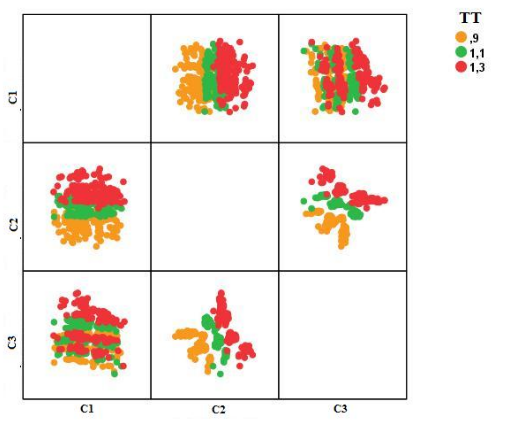


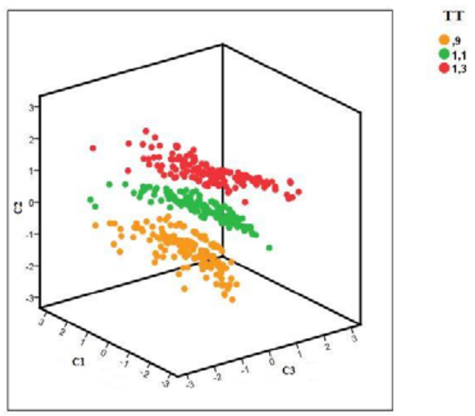
Figure 9. The interaction effect between *TT* and *MIT* on component *C3*

6.2 The architecture runtime behavior visualized in 3-d scatterplots

Having projected the experimental points - which were initially expressed by 16 quality metrics - in a 3-dimentional space, it is also possible to draw conclusions from 3-d scatterplots. All components are shown together, while experimental points are marked with different colors according to the factor levels.



(a)



(b)

Figure 10. 3-d scatterplots of the effects of *TT* on the 3 components

Plots like the ones in Figure 10 help us to designate the factors that have a strong effect on the whole swarm of points, by identifying groupings of the points due to the levels of each factor. More specifically we provide (a) a matrix plot for all couples of components and (b) a 3-d scatterplot for all three components for the factor *TT*.

The plots show a clear grouping of the experimental points, where the points for *TT* = 0.9 are grouped in a swarm representing relatively low values of response times (*C2*) and low values for the throughput metrics of *C3* that are mostly affected by communication latencies. This is explained, because a significant number of transactions exceeding 0.9 sec are aborted and thus they do not contribute to the metrics representing mean response times and throughputs.

Similar groupings of the experimental points were found with respect to the *MIT*, where – as expected – the points for *MIT* = 0.4 are grouped in a swarm representing relatively high values for the metrics of component *C2* and low values for the metrics of component *C3*. A significant number of transactions is aborted for *MIT* = 0.4, thus resulting in lower throughput for the distributed transactions and for the transaction groups of lock-conflicted classes. Furthermore, by these scatterplots it was found that the *MITofSF* distinguishes the points corresponding to the level of 1080 sec, which in any case represents an extreme operating condition with very frequent server failures.

6.3 Threats to validity and additional considerations

Although it is shown that the proposed two-stage method can provide statistical evidence for the architecture tradeoff points and the important factor effects, an obvious threat to the internal validity of our exploratory study is the possibility of confounding. This term is often used to refer to all factors covarying with the considered independent variables (architecture characteristics), which also influence the dependent variables (quality metrics) but have not been included in the experiment's design.

In the considered architecture problem we have not recognized confounding variables. For the sake of illustration of a confounding problem, let us assume that resource consumption for concurrency control affects the simulation results. This would be a threat to the internal validity of the analysis, since concurrency control has not been taken into account in the experiment's design.

Also, if the method is applied on measurements from a benchmarking prototype, we consider that there are increased possibilities for confounding variables (e.g. runtime monitoring overhead), which have not been included in the experimental design.

A threat to the external validity of our method is that *the derived conclusions* for the architecture tradeoff points cannot be generalized to other architecture designs or variants. As we already noted metric dependencies characterize only the analyzed architecture and the obtained results cannot provide evidence for metric interactions in any other architecture. However, we believe that the method is applicable to all architecture patterns where the investigated metrics matter.

Regarding the anticipated conclusion validity, we emphasize that the proposed method is based on an experimental design and a statistical analysis that are both well-established in theory and in practice. For the illustrated simulation-based evaluation of the architecture at hand, the reader is referred to the verification and validation tests for the ACID Sim Tools and the used simulation model (section 4.2). As a consequence of these, there are no obvious threats to the statistical validity of the drawn conclusions.

Finally, an important issue is the extent to which we measure the data relevant to our hypothesis, which is termed in related work as construction validity. Our hypothesis for the analyzed transactional architecture is stated by the quality requirements that are mentioned in section 4. Typical threats like hypothesis guessing, researcher expectancy and evaluation apprehension are not relevant to our study. We used an extensive set of metrics, in order to be able to explore the variability in transaction processing performance and server availability under different circumstances of I/O, CPU and lock contention. The same metrics have been previously proved effective in related work on simulation-based performance evaluation (e.g. in [49]).

An additional consideration regarding the proposed method is that the different views of architecture (logical, process, development and physical views) [31] are usually captured in some form of visual notation and a reliable transformation of this notation to a prototype or a simulation model is not automatic and requires expertise. Part of the problem is that the widely used visual notations like the UML, lack formal semantics [47] and their transformation to a formal simulation model has inherent technical difficulties and pitfalls. However, we believe that as the project teams acquire expertise with the tools that they use, the risk of inaccurate model representation and transformation is gradually reduced. Furthermore, the evolution and the ever-improving maturity of Model-Driven Software Development (MDSD) [50] techniques will eventually push the formalization of UML and other visual notations forward. This will allow automatic or semi-automatic translation into several formal models, suitable for experimentation and simulation in the frame of the proposed method.

7. Conclusion

We introduced a quantitative method for disclosing the statistically significant quality tradeoffs in software architectures and for relating the variability of metrics to specific architecture characteristics. The described two-stage analysis is general, since it is not bound to a specific metric or a specific technique for acquisition of experimental data.

The method is illustrated by evaluating a transactional software architecture based on simulation output for 16 quality metrics that was obtained with the ACID Sim Tools. The factor analysis in the first stage of our approach unveiled a strong correlation of specific metrics on a latent quality feature representing the I/O and recovery costs. Surprisingly, the throughput for the locally processed transactions was found negatively correlated with the aforementioned latent quality and uncorrelated to the throughput of the distributed transactions.

The box-plots for the three components found by the FA revealed sensitivity and tradeoff points, i.e. architecture characteristics that affect one metric or multiple metrics that cannot be optimized simultaneously. In the second stage of the analysis, we performed Analyses of Variance for the components of the FA and in this way we attributed the variability of the analyzed metrics to specific architecture characteristics and interactions between them.

Future research prospects include extensions of the current process, in order to address the problem of quality prediction for an architecture design based on new synthetic metrics [24, 38], as well as the problem of multi-objective optimization under specific cost constraints and tradeoff concerns.

Acknowledgments

We acknowledge the anonymous referees for their helpful comments, which contributed to improving the quality of the article.

References

[1] ACID Sim Tools Site, http://mathind.csd.auth.gr/acid/html/index.html (accessed: Nov. 2008)

[2] Alturi, V., Bertino, E., Jajodia, S. "A Theoretical Formulation for Degrees of Isolation in Databases", Information and Software Technology, 39 (1), 1997, pp. 47-53

[3] Anderson, T. W., Rubin, H. "Statistical Inference in Factor Analysis", Proc. 3rd Berkeley Symposium on Mathematical Statistics and Probability, Vol. 5, 1956, pp. 111-150

[4] Balsamo, S., Marzolla, M. "Simulation modelling of UML software architectures", Proc. of the 17th European Simulation Multiconference (ESM), Nottingham, UK, 2003, pp. 562-567

[5] Balsamo, S., Marzolla, M., Di Marco, A., Inverardi, P. "Experimenting different software architectures performance techniques: a case study", ACM SIGSOFT Software Engineering Notes, 29 (1), 2004, pp. 115-119

[6] Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M. "Model-Based Performance Prediction in Software Development: A Survey", IEEE Transactions on Software Engineering, 30 (5), 2004, pp. 295-310

[7] Bardram, J. E., Christensen, H. B., Corry, A. V., Hansen, K. M., Ingstrup, M. "Exploring quality attributes using architectural prototyping", Proc. of the 1st Int. Conf. on the Quality of Software Architectures & 2nd Int. Workshop on Software Quality (QoSA and SOQUA), LNCS 3712, Springer, 2005, pp. 155-170

[8] Bass, L., Clements, P., Kazman, R. Software Architecture in Practice, 2nd ed., SEI Series in Software Engineering, Addison-Wesley, 2003

[9] Becker, S., Dencker, T., Happe, J. “Model-Driven Generation of Performance Prototypes”, Proc. of the SPEC International Performance Evaluation Workshop (SIPEW), LNCS 5119, Springer, 2008, pp. 79-98

[10] Bengtsson, P. O., Bosch, J. “Scenario-Based Architecture Reengineering”, Proc. of the 5th Int. Conf. of Software Reuse (ICSR), 1998

[11] Berander, P. “Merging perspectives on software quality attributes”, In: Software quality attributes and trade-offs, L. Lundberg, M. Mattsson and C. Wohlin, Eds., Blekinge Institute of Technology, Sweden, 2005

[12] Blom, G. Statistical estimates and transformed beta variables, Wiley, New York, 1958

[13] Burke, B., Monson-Haefel, R., Enterprise JavaBeans 3.0, O’Reilly, 2006

[14] Clements, P., Kazman, R., Klein, M. Evaluating Software Architectures: Methods and Case Studies, SEI Series in Software Engineering, Addison-Wesley, 2001

[15] Crnkovic I., Larsson, M., Preiss, O. “Concerning predictability in dependable component-based systems: classification of quality attributes”, LNCS 3549, Springer, 2005, pp. 257-278

[16] Chrysanthis, P. K., Samaras, G., Al-Houmaily, Y. J. “Recovery and performance of atomic commit processing in distributed database systems”, In: Recovery Mechanisms in Database Systems, V. Kumar and M. Hsu, Eds., Prentice-Hall, 1998, pp. 370-416.

[17] Dobrica, L., Niemela, E. “Survey on Software Architecture Analysis Methods”, IEEE Transactions on Software Engineering, 28 (7), 2002, pp. 638-653

[18] DoD, DoD Instruction 5000.61: “Modeling and Simulation (MandS) Verification, Validation, and Accreditation (VVandA)”, Defense Modeling and Simulation Office, Office of the Director of Defense Research and Engr. (online: http://www.dmso.mil/docslib), 2002

[19] Eguiluz, H. R., Barbacci, M. R. “Interactions among techniques addressing quality attributes”, Tech. Report CMU/SEI-2003-TR-003, Software Engineering Institute, Carnegie Mellon, Pittsburgh, 2003

[20] Field, A. Discovering Statistics Using SPSS, 2nd Edition, SAGE Publications, 2005

[21] Gorton, I. Essential Software Architecture, Springer, 2006

[22] Grunske, L. "Identifying 'Good' Architectural Design Alternatives with Multi-Objective Optimization Strategies", Proc. of the 28th Int. Conference on Software Engineering (ICSE), 2006, pp. 849-852

[23] Grunske, L. "Early quality prediction of component-based systems – A generic framework", The Journal of Systems and Software, 80, 2007, pp. 678-686

[24] Katsaros, P., Iakovidou, N., Soldatos, T. "Evaluation of composite object replication schemes for dependable server applications", Information and Software Technology, 48 (9), 2006, pp. 795-806

[25] Katsaros, P., Angelis, L., Lazos, C. "Performance and effectiveness trade-off for checkpointing in fault-tolerant distributed systems", Concurrency and Computation: Practice and Experience, 19, 2007, pp. 37-63

[26] Kazman, R., Bass, L., Abowd, G., Webb, M. "SAAM: A Method for Analyzing the Properties of Software Architectures", Proc. of the 16th Int. Conf. on Software Engineering (ICSE), 1994, pp. 81–90

[27] Kelton, W. D., Law, A. M. "An analytical evaluation of alternative strategies in steady-state simulation", Operations Research 32, 1984, pp. 169-184

[28] Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El Emam, K., Rosenberg, J. "Preliminary guidelines for empirical research in software engineering", IEEE Transactions on Software Engineering, 28 (8), 2002, pp. 721-734

[29] Klein, J., Weiss, D. "What is Architecture?", In: Beautiful Architecture: Leading Thinkers Reveal the Hidden Beauty in Software Design, D. Spinellis & G. Gousios Eds., O'Reilly, 2009

[30] Krzanowski, W. J., Principles of Multivariate Analysis: A User's Perspective, Oxford Science Publications, 1993

[31] Kruchten, P. "Architectural blueprints – The 4+1 view model of software architecture", IEEE Software, 12 (6), 1995, pp. 42-50

[32] Litoiu, M., Rolia, J., Serazzi, G. "Designing process replication and activation: A quantitative approach", IEEE Transactions on Software Engineering, 26 (12), 2000, pp. 1168-1178

[33] Losavio, F., Chirinos, L., Perez, M. A. "Quality models to design software architectures", Proc. of Tech. of Object-Oriented Languages & Systems (TOOLS 38), 2001, pp. 123-135

[34] Martin, C. P., Ramamritham, K. "Toward formalizing recovery of (advanced) transactions", In: Advanced Transaction Models and Architectures, S. Jajodia and L. Kerschberg, Eds. Kluwer, Boston, 1997

[35] Martis, M. S., "Validation of simulation based models: a theoretical outlook", The Electronic Journal of Business Research Methods, 4 (1), 2006, pp. 39-46

[36] Mentis, A., Katsaros, P., Angelis, L. "ACID Sim Tools: A simulation framework for distributed transaction processing architectures", Proc. of the 1st Int. Conf. on Simulation Tools and Techniques (SimulationWorks Industry Track), Marseille, France, 2008

[37] Mentis, A., Katsaros, P. "The ACID model checker and code generator for transaction processing", Proc. of the 2009 High Performance Computing & Simulation Conference (HPCS), Leipzig, Germany, IEEE, 2009, pp. 138-144

[38] Mentis, A., Katsaros, P., Angelis, L. "Synthetic metrics for evaluating runtime performance of software architectures with complex tradeoffs", (To appear in) Proc. of the 35th Euromicro Software Engineering and Advanced Applications Conference (SEAA), Patras, Greece, IEEE Computer Society, 2009

[39] Merson, P., Hissam, S. "Predictability by Construction", In: Companion to the 20th Annual ACM SIGPLAN Conf. on Object Oriented Programming Systems Languages and Applications (OOPSLA), 2005, pp. 134-135

[40] Object Management Group, Object Management Architecture Guide, revision 3.0, OMG Technical Committee Document ab/97-05-05, June 1995

[41] Object Management Group, Transaction Service Specification, version 1.3, OMG Technical Committee Document ptc/2003-03-08, March 2003

[42] Olumofin, F. G., Misic, V. B., "A holistic architecture assessment method for software product lines", Information and Software Technology, 49, 2007, pp. 309-323

[43] OMNeT++ Community Site, http://www.omnetpp.org/ (accessed: Nov. 2006)

[44] Paul, H. S., Gupta, A., Badrinath, R., "Performance comparison of checkpoint and recovery protocols", Concurrency and Computation: Practice and Experience, 15, 2003, pp. 1363-1386

[45] Reussner, R., Firus, V. "Introduction to overlapping attributes", In: Dependability Metrics, I. Eusgeld, F. C. Freiling and R. Reussner, Eds., LNCS 4909, Springer, 2008, pp. 243-244

[46] Seng, J., "A study on industry and synthetic standard benchmarks in relational and object databases", Industrial Management and Data Systems, 103 (7), 2003, pp. 516-532

[47] Shaw, M., Garlan, D. "Formulations and Formalisms in Software Architecture", In: Computer Science Today: Recent Trends and Developments, J. van Leeuwen Ed., LNCS 1000, Springer, 1995, pp. 307-323

[48] Shaw, M., Garlan, D. Software Architecture: Perspectives on an Emerging Discipline, Englewood Cliffs, NJ, Prentice Hall, 1996

[49] Smith, C. U., Williams, L. G. Performance Solutions: A practical guide to creating responsive, scalable software, Addison Wesley, 2001

[50] Stahl, T., Völter, V., Model-Driven Software Development: Technology, Engineering, Management, Wiley, 2006

[51] Transaction Processing Performance Council, "TPC Benchmark C", Standard Specification, Version 5.4, 2005: available at http://www.tpc.org/tpcc/

[52] Varga, A. "The OMNeT++ Discrete Event Simulation Environment", In Proc. of the European Simulation Multiconference (ESM), Prague, Czech Republic, 2001, pp. 319-325

[53] Vieira, M., Duraes, J., Madeira, H. "Dependability benchmarks for OLTP systems", In: Dependability Benchmarking for Computer Systems, K. Kanoun, L. Spainhower, Eds., IEEE Computer Society & John Wiley & Sons, 2008

[54] Williams, B. J., Carver, J. C. "Characterizing software architecture changes: a systematic review", Information and Software Technology, 52, 2010, pp. 31-51

[55] Zhu, L., Liu, Y., Gorton, I., Bui, N. B. "Customized Benchmark Generation Using MDA", In Proc. of the 5th Working IEEE/IFIP Conference on Software Architecture, Washington DC, USA, 2005, pp. 35-44

Appendix

An effective procedure for using ACID Sim Tools in the context of the proposed method includes the following steps:

1. Obtain the different views of the architecture design [31] in UML or in an architecture definition language.
2. Examine if all architecture characteristics can be simulated by existing ACID Sim Tools modules. If not, use the specification language of ACID Sim Tools [37] to produce an implementation of the missing characteristics.
3. Transform the architecture definition into the OMNeT++ model definition language.
4. Verify that the anticipated correctness properties are fulfilled.

5. Set the values of model parameters by computing resource demands from benchmarks or other representative workloads; values for system parameters should correspond to real system equivalents.
6. Validate the simulation model with respect to the validity concerns of section 4.2.
7. Define the experimental design.
8. Collect the simulation results and proceed to the subsequent two-stage analysis outlined in section 3.

Steps 5 - 8 may be applied to parameter sets that represent different workloads.