LotoTis in Comparison with Other Performance-Oriented LOTOS Extensions

Ina Schieferdecker
GMD FOKUS
Hardenbergplatz 2
D-10623 Berlin
tel: (030) 254 99 222
e-mail: ina@fokus.gmd.de

Adam Wolisz
Technical University Berlin
Einsteinufer 25
D-10587 Berlin
tel: (030) 314 22 911
e-mail: wolisz@ftsu00.ee.tu-berlin.de

Abstract

The paper presents LotoTis — a performance-oriented and upward-compatible extension of Lotos, whose main feature are structured actions. A structured action is parameterized with an interaction time, a priority, a set of requested resources, and a monitoring signal. It allows us to model system tasks of distributed systems with their performance characteristics very easily. In addition, the paper compares LotoTis with related performance-oriented Lotos extensions and identifies the advantages and shortcomings of these approaches.

1 Performance Evaluation based on Formal Specifications

The first concern with a distributed system is its correct behavior according to the intended system functionality. The second concern is to have a proper performance with reasonable costs. While functional correctness has been considered primarily in the past, recent application of computer systems, for instance multimedia applications in broadband communication networks, showed that performance characteristics are of the same importance as the functional correctness of a distributed system.
How to develop functionally correct distributed systems starting with a formal requirement specification down to the system implementation is a well studied area. In contrast to that, it is not unusual that a system is fully implemented (or at least implemented as a prototype) and tested for functional correctness before any attempt is made to investigate its timing behavior and its performance.
The main problem why performance investigations are made very late in the development process, is the separation between formal specifications and performance analysis. Traditional specification techniques are lacking quantified time and other features for performance modeling. They describe only the causal ordering of events, i.e. their logical relations. On the other hand, classical performance models like queueing systems lack the description of the functional behavior, so that logical relations are omitted. Most often a performance model is developed that is independent from the functional model. Thus

it is hard if not even impossible to proof that functional and performance model coincide with each other.
An integrated framework that supports functional and performance-oriented behavior specification and allows us to evaluate performance from that specification would offer major advantages. Costly re-prototyping or even re-implementation can be avoided, if performance could be predicted already within the design phase of a distributed system. Henceforth, new concepts have to be incorporated into formal specification techniques. The need to extend formal specification techniques was recognized already in the mid eighties [14]. First, the necessity to close the gap between formal specification and performance evaluation obtained only little attention. Recently, there had been published a limited number of approaches going in this direction ([6], [3], [5]).
This paper presents the LotoTis approach that is a newly-designed performance-oriented formal description technique (FDT). Its main advantage is the upward compatibility with Lotos that is one of the internationally standardized FDTs [7]. In particular, performance-oriented LotoTis specifications can be derived from existing Lotos specifications by applying a number of refinement rules.
The introduction into LotoTis shortly repeats four architectural concepts that are considered to be essential for performance evaluation based on FDTs. They are namely quantified time, quantified nondeterminism, quantified parallelism, and monitoring. In the context of process algebraic approaches, these concepts lead us to the notion of structured actions and to new operators. Their incorporation into Lotos results in the LotoTis approach.
Subsequent to that, related Lotos extensions are detailed discussed and compared with LotoTis. A summary of the advantages and shortcomings of each of theses approaches finishes the paper.

2 Architectural Concepts Supporting Performance Modeling

The notion of architectural concepts were introduced only recently by C.A. Vissers et al. [13]:

> "Architectural concepts are abstractions (models) of frequently occurring aspects of technical objects. Such concepts are manipulated during the design and implementation process. Examples are service, service access point, service primitive, service primitive parameter, service data unit, protocol, protocol data unit, abstract interface, real interface, etc. Naturally such concepts should find a straightforward reflection in the design language that is used."

Although the above given definition only considers architectural concepts within the light of functional objects such as service, service access point, etc., there is no difficulty to apply the notion of architectural concepts to performance-oriented aspects of technical objects!
In order to identify the architectural concepts needed for performance modeling we have to identify the aspects of distributed systems which influence their performance. The starting point is to consider a distributed system as being composed of a number of tasks.

The execution of that system is the parallel and/or sequentially ordered execution of the tasks. Every task realizes a certain functionality. Besides their functional behavior, tasks are

- time-consuming,
- request resources for their execution, and
- have priorities in access to these resources.

For obvious reasons, the precondition for any performance evaluation is the possibility to express the time consumption of tasks. This leads us to the concept of *quantified time*. The availability of resources influences the grade of parallelism in a distributed system and therefore its performance. Thus, a concept of *quantified parallelism* is required.
Furthermore, distributed systems are inherently nondeterministic due to their complexity and the unpredictable behavior of their environment. Formal specifications represent nondeterminism by means of choices between several possible subsequent behaviors. For the sake of performance evaluation, we have to quantify these choices explicitly. Hence, we have to have a concept of *quantified nondeterminism*.
Last but not least, performance characteristics of a distributed system are determined by the execution of tasks and the time distances between them. They can only be determined if the execution of tasks is observable from outside of the system. A concept of *monitoring* is required in order to make observable any task of interest — independent of whether this task is externally visible or internally hidden.

Let us consider the architectural concepts that are needed to merge FDTs and performance evaluation in the light of process algebras. Nowadays, it is quite clear how to incorporate quantified time into process algebras. A good survey of time-extended process algebras is given in [12].
Besides the concept of quantified time, a performance-oriented process algebra has to offer means for quantified nondeterminism. Let us firstly identify the potential causes of nondeterminism. At choice points several alternative behaviors are simultaneously enabled and the system has to choose one of them for future development. While the enabling of alternative behaviors is determined by the environment, for instance by processes composed in parallel, the final decision which of the enabled behaviors is chosen is made completely internally. All alternative behaviors are equally probable to be chosen in classic process algebras. No means exist to weight the alternatives. Another cause of nondeterminism exists within disabling expressions. In classic process algebras, two behavior expressions can be combined so that one of them may disable the other if its starting action is enabled. However, there are no means to restrict the time points when the disabling may occur. In addition, the occurrence of the enabling action cannot be enforced in most classic process algebras. Although the disabling behavior expression is enabled the disabling may not occur, because it is delayed. Nondeterministic choices can be solved by means of action priorities and weights. Nondeterministic disablings require the specification of time points, when the disabling has to occur.
The third concept for performance analysis is quantified parallelism. The use of action priorities is one possibility to quantify parallelism. Action priorities decide which action is executed first. They assume one imaginary processor which has to be allocated by each action in order to be executed. The access to this processor is adjusted by the action priorities. However in reality, the grade of parallelism is restricted by the number of processors

and/or the number of other resources offered by the execution environment which is in general greater than one. Hence the first rule is too strict, so that we appreciate the use of resources as a separate modeling concept for the description of execution environments. The access to resources is again ordered by action priorities and/or weights.
Monitoring is the fourth concept needed for performance evaluation. Actions of interest can be made observable by designating them with monitoring signals. Monitoring signals are a new feature in process algebras. Once an action designated with a monitoring signal occurs, the monitoring signal is offered to the environment with its unambiguous identifier and the actual system time.

3 LotoTis — A Performance-Oriented Lotos Extension

We decided to use the process algebraic framework for performance-oriented specifications due to several advantages like the well-defined semantics, the possibilities of abstraction and composition, and the existence of equivalence notions for behavior specifications. The second objective of this work was to promote the standard conform development of performance-oriented specifications. Since Lotos is the only process-algebraic, internationally standardized FDT, the Lotos extension LotoTis has been defined.
The origins of LotoTis are

1. Lotos: Language of Temporal Ordering Specification — a FDT that is based on process algebras (CCS, CSP) and algebraic data type specifications (Act One)

2. Tis: Timed Interacting Systems Approach ([18],[16]) — a performance-oriented specification language.

The main feature of LotoTis are structured actions, which replace the classic actions used in process algebras and model the performance characteristics of system tasks. The idea of structured actions was mainly developed within Tis.

3.1 The Notion of Structured Actions

The idea of parameterizing actions with performance characteristics of system tasks makes the performance-oriented modeling of distributed systems conceptually easy. There are no separate modeling features for time, priorities, resources, and monitoring. All these features belong to the notion of a structured action. While a classic action represents the functionality of a system task, the parameters of a structured action represent the performance characteristics of that task. Structured actions [17] are the basic concept we adopt to specify performance-oriented behavior. The parameters of a structured action (Fig. 1) are explained in the following.

1. The interaction time defines the length of an interaction with synchronization partners[1]. The interaction time makes structured actions non-instantaneous, so that they describe true-concurrent behavior.

[1]A structured action without any synchronization can be considered to be a special case of synchronizing structured actions.

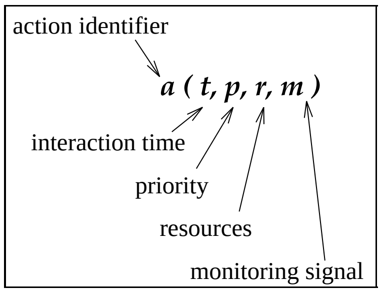

Figure 1: The Notion of Structured Actions

```
... external a(t, 1, [R(2)]) in
    ( ...; a; ... ) |[a]| ( ...; a; ... ) ...
```

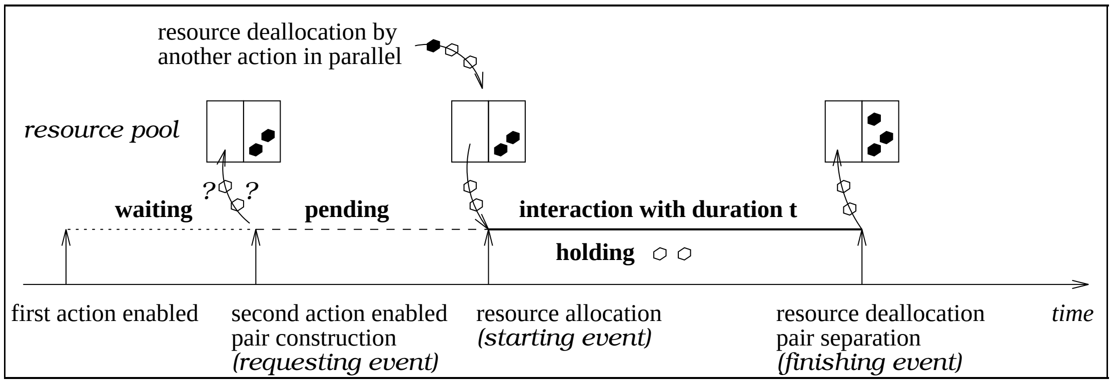

Figure 2: Synchronization of Structured Actions

2. The priority orders simultaneously enabled actions and determines the subsequent behavior. In addition, the access to resources is adjusted by priorities.

3. The set of resources has to be allocated before any interaction[2]. In addition to action priorities, resource disciplines determine the order of resource allocations.

4. The monitoring signal makes action occurrences observable from outside. Whenever an action attached with a monitoring signal occurs, the monitoring signal offers the signal identifier and the absolute time of the action occurrence to the specification environment.

The behavior of structured actions is explained for three basic cases, which are (1) a single structured action, (2) the synchronization of structured actions, and (3) the interleaving of structured actions.

Single action Whenever a structured action is enabled it tries to allocate needed resources immediately. A — not explicitly specified — pending period until the successful resource allocation may occur. Resources are assigned to pending actions on

[2]Although resources and the resource management during system execution could be represented by additional processes and data types, we adopt the use of explicit resources as a very concise and succinct modeling feature.

```
... external a(0, 3, [R]), b(0, 2, [R]), c(0, 1, [R]) in
    ( a; ... ||| b; ... ||| c; ... )
```

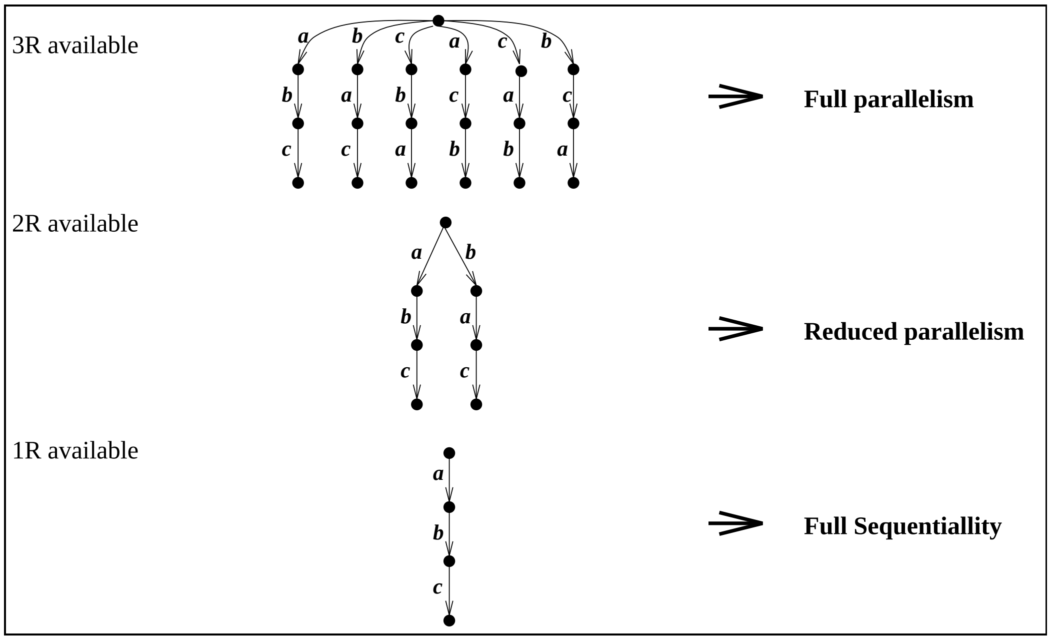

Figure 3: Independent Structured Actions

an all-or-nothing principle, meaning that either all needed resources are assigned, or no assignment takes place. If possible, pending actions are assigned resources without any delay. Resources are assigned on the action priority and resource discipline[3] basis. After resource allocation the interaction period starts immediately. The duration of the interaction period is determined by the interaction time. Resources are occupied throughout the interaction period.

Synchronization Figure 2 depicts the synchronization of two structured actions `a` with interaction time `t`, priority `1`, and two requested resources of type `R`. Synchronization partners become in general enabled at a different point in time. In Fig. 2 the first action `a` becomes enabled earlier than the other one. It has to wait for its synchronization partner before requesting resources. Thus, there is a — not explicitly specified — waiting period for synchronization partners. Eventually, both actions are enabled and form an action tuple. This action tuple behaves like a single action: it requests resources, waits for the resource allocation, interacts, and deallocates resources.

Interleaving Interleaved structured actions, i.e. parallel composed structured actions without synchronization, allow us to model full parallelism, reduced parallelism, and full sequentiality. Figure 3 gives three independent, instantaneous structured actions `a, b,` and `c`. Each of them needs one resource `R`. The grade of parallelism is determined by the number of available resources `R`. The three cases — at most one `R`, two `R`, and at least three `R` — could be also depicted as `1.` $\{$`a,b,c`$\}$,

[3]For time being, we restrict ourselves to the case of **`fifo`** (first in first out).

2. `{a,b}, {c}` and 3. `{a}, {b}, {c}`, where actions in parenthesis denote the parallel execution of actions while commata denote their sequential ordering.

3.2 New Performance-Oriented Operators

While structured actions can be used to model quantified time, quantified parallelism, and action monitoring, we need additional operators for the modeling of quantified nondeterminism.
An operator to quantitatively choose between several alternative behaviors is required. Therefore, we decided to use a *probabilistic choice operator* which weights the alternative behaviors according to some probability. The expression B_1 `[p]` B_2 with `p` $\in$ `[0,1]` denotes that the left behavior will be chosen with probability `p`, while the right behavior will be chosen with probability `1-p`.
Nondeterminism resulting from disabling is quantified by the use of timeout operators. A timeout operator has a time parameter `t`, that determines the time point at which the disabling behavior expression is enabled. We distinguish between two forms of the timeout operator — the *hard timeout* and the *soft timeout*. While in the first case the disabling always occurs after `t` time, the disabling in the second case only occurs if the behavior expression to be disabled has not yet successfully started the interaction period of its first action. If the left behavior is fast enough to start an interaction before `t` time has passed it cannot be disabled anymore. In both cases, the timeout cannot occur if the behavior expression to be disabled has terminated before time `t`.
Please note that potential nondeterminism still exists. In particular, at time `t` when both the disabling behavior expression and the behavior expression to be disabled are able to execute actions with the same priority, it is undetermined which of them is chosen. Secondly, although the access to resources is adjusted by the assignment of priorities to actions and resource disciplines to resources, there is still the possibility that two equally prioritrized actions request resources at the same point in time.

3.3 Definition of LotoTis

LotoTis is Lotos extended with

1. structured actions that incorporate time, priorities, resources, and monitoring signals,,
2. probabilistic choice operator, and
3. soft and hard timeout operators.

The LotoTis syntax is given in Table 1. Since LotoTis is an extension of Lotos we only give the additional constructs that are marked with `->`. Due to lack of space we can only present the syntax of LotoTis and renounce from the formal semantics definition of LotoTis. For more information please refer to [15]. A LotoTis specification defines the time domain to be discrete or dense. Global available resources are declared on top of the specification. Additionally, each process definition may contain a resource declaration of locally available resources — those resources can only be accessed from process internal actions. Global resources cannot be accessed from internal actions of a locally defined

```
   specification:
     specification-symbol specification-identifier formal-parameter-list
       global-type-definitions
->     [ time-declaration ]
->     [ resource-symbol resource-declarations ]
->     [ external-action-declarations ]
       behavior definition-block
     endspecification-symbol

-> time-declaration: time-symbol time-domain in-symbol
-> time-domain: discrete | dense

-> resource-declarations:
     resource-identifier ‘[’ resource-number [ ‘,’ resource-discipline ] ‘]’
     [ ‘,’ resource-declarations ]

-> action-declarations:
     action-identifier ‘(’ interaction-time [ ‘,’ priority
                           [ ‘,’ resource-request [ ‘,’ monitoring-signal ]]] ‘)’
     [ ‘,’ action-declarations ]
-> external-action-declarations:
     external-symbol action-declarations in-symbol

process-definition:
  process-symbol process-identifier formal-parameter-list define-symbol
->  [ resource-declarations ]
    definition-block
  endprocess-symbol

behavior-expression:
   ...
-> | hide-symbol action-declarations in-symbol
-> | behavior-expression ’[[’ probability ’]]’ behavior-expression
-> | behavior-expression ’[[’ time ’]>’ behavior-expression
-> | behavior-expression ’[[’ time ’>>’ behavior-expression
   | ...
```

Table 1: The Syntax of LOTOTIS

process. Therefore, resources can only be allocated by actions declared on the same level as the resources themselves.
Every structured action is explicitly declared by the use of the external operator (for external gates) or by the use of the hide operator (for internal actions). It is not mandatory to define all parameters of a structured action; default assumptions are zero duration, zero priority, no requested resources, and no monitoring signal.
The new operators have already been discussed above. A summary of all LOTOTIS operators is given in Table 2. They can be used to specify complex behavior composed of structured actions, **stop**, **exit**, and process instances. The result of such compositions are behavior expressions which describe the behavior of a process or a complete system.
We distinguish between basic LOTOTIS and full LOTOTIS. Basic LOTOTIS is LOTOTIS without a data part and without data dependencies in behavior expressions. Full LOTOTIS is basic LOTOTIS extended with data dependencies. It incorporates data type

Operator	Syntax	Comment
External	**external** $g_1(..)$, $\ldots, g_n(..)$ **in** $\mathcal{B}$	The external operator declares external gates $g_1, \ldots, g_n$ of a LotoTis specification with their parameters.
Hide	**hide** $a_1(..)$, $\ldots, a_n(..)$ **in** $\mathcal{B}$	The hide operator declares internal, hidden structured actions $a_1, \ldots, a_n$ with their parameters. These actions are unobservable outside of $\mathcal{B}$.
Action Prefix	$a; \mathcal{B}$	$\mathcal{B}$ becomes enabled after the interaction a has been completed.
Enabling	$\mathcal{B}_1 \gg \mathcal{B}_2$	$\mathcal{B}_2$ becomes enabled after the successful termination of $\mathcal{B}_1$ via the **exit** process.
Parallel Composition	$\mathcal{B}_1 \| [g_1, \ldots, g_n] \| \mathcal{B}_2$	$\mathcal{B}_1$ and $\mathcal{B}_2$ are executed in parallel and interact in their gates $g_1, \ldots, g_n$.
Full Synchronization	$\mathcal{B}_1 \|\| \mathcal{B}_2$	$\mathcal{B}_1$ and $\mathcal{B}_2$ become are executed in parallel. They interact in their externally visible actions.
Interleaving	$\mathcal{B}_1 \|\|\| \mathcal{B}_2$	$\mathcal{B}_1$ and $\mathcal{B}_2$ are executed in parallel and fully independently, i.e. without any interaction.
Choice	$\mathcal{B}_1 [\,] \mathcal{B}_2$	Provided that both behavior expressions are potentially able to execute their first action, only one — either $\mathcal{B}_1$ or $\mathcal{B}_2$ — is chosen. The choice is being made nondeterministically.
Probabilistic Choice	$\mathcal{B}_1 [[p]] \mathcal{B}_2$	Either $\mathcal{B}_1$ or $\mathcal{B}_2$ becomes enabled. The choice is being made randomly, with probabilities p for $\mathcal{B}_1$ and $1-p$ for $\mathcal{B}_2$, respectively. This choice does not take into account whether a behavior expression is potentially able to execute its first action or not. It may happen, that a deadlocked behavior expression is being enabled — this cannot happen with the pure choice operator.
Disabling	$\mathcal{B}_1 [> \mathcal{B}_2$	$\mathcal{B}_1$ becomes enabled immediately. $\mathcal{B}_2$ may disable $\mathcal{B}_1$ at arbitrary time, unless $\mathcal{B}_1$ has already terminated.
Soft Timeout	$\mathcal{B}_1 [[t]> \mathcal{B}_2$	$\mathcal{B}_1$ becomes enabled immediately. $\mathcal{B}_2$ disables $\mathcal{B}_1$ at (relative) time t unless $\mathcal{B}_1$ has not yet started an interaction or has not yet terminated. If $\mathcal{B}_1$ started an interaction before time t it "survives", the disabling becomes impossible.
Hard Timeout	$\mathcal{B}_1 [[t \gg \mathcal{B}_2$	$\mathcal{B}_1$ becomes enabled immediately. $\mathcal{B}_2$ disables $\mathcal{B}_1$ at (relative) time t unless $\mathcal{B}_1$ has not yet terminated. Please note, that in contrast to the soft timeout operator $\mathcal{B}_2$ may disable $\mathcal{B}_1$ also within interaction periods. In that case synchronization partners of the disabled interaction of $\mathcal{B}_1$ are deadlocked as they are waiting for the disabled synchronization partner to finish the common interaction period.

Table 2: Basic LotoTis Operators

Feature	**Syntax**	**Comment**
Value offer	$a\,!v$	Action a offers value v.
Variable offer	$a\,?x : type$	Action a offers variable x and requests a value for x.
Parameterized processes	**process** $P[g_1,\ldots,g_n](x_1 : t_1,\ldots,x_m : t_m)$ .. **endproc**	Process P has formal parameters $x_1,\ldots,x_m$ of type $t_1,\ldots,t_m$, respectively. They are actualized when P is instantiated.
Parameterized exit	**exit** $(x,\ldots)$	Upon successful termination, a list of data values is offered to the subsequent behavior.
Parameterized sequential composition	**exit** $(x,\ldots)$ $\gg$**accept** $x_1 : t_1,\ldots$ **in**B	The exit values are passed to the subsequent behavior.
Local value definition	**let** $x : t = v,\ldots$ **in**B	Variable x of type t is bounded to value v in behavior $\mathcal{B}$.
Guards	$[g]->\mathcal{B}$	Behavior $\mathcal{B}$ is enabled only if the guard g can be evaluated to true.

Table 3: Additional Full LotoTis Features

specifications and the definition of data dependencies in the behavior part. Most importantly, we can use the data part for setting parameters of structured actions and for setting parameters of performance-oriented operators during system execution. It allows us to model the dynamic change of performance characteristics during system run. The aspect of non-static parameters is of particular interest for the modeling of distributed systems since their characteristics often change over time.
The additional features of full LotoTis are informally explained in Table 3. The additional, data-dependent features of full LotoTis are similar to the features of full Lotos. A good introduction and guidelines for their use can be found in [4].

3.4 Properties of LotoTis

The refinement of LotoTis and Lotos is defined as follows. A LotoTis specification $\mathcal{B}_2$ refines a LOTOS specification $\mathcal{B}_1$, denoted by $\mathcal{B}_1 >> \mathcal{B}_2$, if and only if for every interaction a that is started from $\mathcal{B}_2$, there is an action a that is executed by $\mathcal{B}_1$ and the subsequent behavior expressions stand in the refinement relation, too. Hence we compare the occurrences of Lotos actions with the occurrences of corresponding LotoTis interaction periods. Obviously, the refined behavior is a subset of the original behavior. Three transformations from Lotos to LotoTis can be identified, which result in a LotoTis specification that refines the Lotos specification. In addition, every LotoTis specification is a refinement of its underlying LOTOS specification. The transformations are

1. Actions $\mapsto$ Structured actions, i.e. $a \mapsto a(t,p,r,m)$
2. Choice $\mapsto$ Probabilistic choice, i.e. $\mathcal{B}_1[\,]\mathcal{B}_2 \mapsto \mathcal{B}_1[[p]]\mathcal{B}_2$

3. Disabling $\mapsto$ Timeout, i.e. $\mathcal{B}_1[>\mathcal{B}_2 \mapsto \mathcal{B}_1[[t\gg\mathcal{B}_2$ or $\mathcal{B}_1[[t]>\mathcal{B}_2$

The refinement relation between LotoTis and Lotos allows us to proof that LotoTis is an upward compatible extension of Lotos:

1. Every Lotos specification is syntactically a LotoTis specifications.
2. The semantics of a Lotos specification is preserved when it is interpreted as a LotoTis specification.

For a formal proof of these LotoTis properties please refer to [15].

4 Comparison with Related Work

4.1 Tis — The Timed Interacting Systems Approach

The Timed Interacting Systems approach Tis [18] is a performance-enhanced specification language based on process algebras. Since LotoTis is based on Tis and further developments of LotoTis and Tis happened in parallel, both languages have many concepts in common.
A Timed Interacting System (`TIS`[4]) is a tuple consisting of

1. a ticker,
2. a set of facilities,
3. the behavior description,
4. the time instant when the operation of the `TIS` begins, and
5. a monitoring environment supporting the observation of system behavior.

The ticker generates ticks providing a means of time progress common to the whole `TIS`. Ticks can be counted by process local clocks (which are set when a process is enabled or reset during the execution of timed action).
Facilities are memoryless, reusable, non-sharable resources, structured into different types with, in general, a different number of elements for each type. The elements of a given type are identical. In addition a service discipline is specified for each facility type, such as FCFS (first come first served).
A `TIS` is described on three levels.

1. The system composition level allows the parallel compositions of several `TIS`s.
2. The system level allows the definition of facilities with their service disciplines, time distribution functions, probabilities, and the definition of the `TIS` as a composition (not necessarily parallel composition!) of several processes.
3. The process level allows the definition of the process behavior consisting of timed actions, operators used for the composition of timed actions, and the port list, specifying which timed actions are candidates for synchronization with other processes.

[4]Both the Timed Interacting Systems approach and a Timed Interacting System are denoted by TIS due to historical reasons. However, we use different type-settings: Tis and `TIS`, respectively.

TIS distinguishes between two types of timed actions:

1. Relatively timed action: The action time counts relatively to the enabling of the action.

2. Absolutely timed action: The action time counts absolutely to process local clocks.

Time may be explicitly defined in a TIS specification by assigning non-null duration to timed actions. The preparation time of a timed action defines a time interval which passes independently of other behavior expressions and starts as soon as a timed action has been enabled. After completion of the preparation period, the interaction period starts if the facilities required for this action are available. Both preparation and interaction periods may be defined as random variables.
The parameters of a timed actions are the following ones:

1. The preparation time is in case of a relatively timed action a random variable of a specified distribution function or in case of an absolutely timed action an absolute time value of a clock and describes the individual preparation of each action for an interaction. An action that is in its preparation period can be interrupted.

2. The interaction time is a random variable of a specified distribution function and describes the duration of an interaction. The interaction period cannot be interrupted at all.

3. Facilities are a tuple of requested facilities that have to be allocated before the interaction can start. Competition for facilities is resolved based on the priority of a timed action and the service discipline of the facilities.

4. Priority is an integer describing the priority in access to facilities. Negative values of the priority describe a preemtable access to facilities.

5. Clocks are a list of clock identifiers that are reset at the end of an interaction.

Besides the complex functionality of timed actions, TIS offers a number of "untraditional" operators. While action prefix, enabling, and interleaving operator have similar functionalities as in other classical process algebras, the semantics of the parallel composition operator $\Uparrow$ is more complex.
Let us consider a parallel composition of m processes $P_1[a_1] \Uparrow P_2[a_2] \Uparrow \ldots \Uparrow P_m[a_m]$ where $a_i, i = 1, 2, \ldots, m$ denote lists of actual ports. Each action a belonging to more than one of the gate lists a_i must be executed by the respective two or more processes synchronously for an interaction. Each action a that takes part in the interaction is called replica. Replicas may have different parameters, i.e. different interaction times, different facilities, and different priorities! Upon enabling, each replica enters its preparation period, whose duration is given by its preparation time. When the last replica of the tuple has been completed its preparation period, the interaction is called pending. A pending interaction can start an interaction period if the logical sum of all requested facilities is available. On the other hand, the interaction must start if the facilities are available. Facilities are assigned to pending actions on an "all or nothing" principle, i.e. either all needed facilities are assigned or no assignment takes place. The interaction period takes the maximal amount of time given by the interaction time of its replica. Replicas having

shorter interaction times leave the interaction earlier and deallocate their facilities, that do not belong to the logical sum of requested facilities of remaining replicas.
TIS supports two choice operators. Alternative behaviors in choice expressions are called branches. The probabilistic choice with the semantics of an internal choice explicitly defines the probabilities of each of the branches. One and only one of the branches is selected and enabled for each execution according to these predefined probabilities. Priorities of the actions, or lack of facilities cannot influence the selection, these factors can only influence the evolution of the selected branch.
A racing operator has the semantics of an external choice. In the case of the racing operator, all branches are simultaneously enabled. The parallel development of the branches continues only as long as all timed actions in all the branches remain either within their preparation periods or remain pending. As soon as the first action starts the interaction period, the branch containing this action "wins the competition" and all other behavior expressions will be disabled. The selection of the "winner" is possibly influenced by the execution environment and by behavior expressions, that are not directly involved in the parallel racing. This is due to the fact, that parallel actions with higher priorities and/or lack of facilities may clearly influence the moment of allocating facilities and starting the interaction period.
Monitoring signals are attached to timed actions for the definition of performance characteristics directly in a TIS specification. The monitoring signal and the actual time is passed to the monitoring environment, when the interaction period starts. The monitoring environment includes pre-specified functions to derive stochastic performance characteristics during execution of a system. Therefore the precise definition of performance characteristics is already supported in the specification of a protocol.
TIS was one of the first attempts to define a performance-oriented specification technique based on process algebras. It provides a powerful simulation tool SIMTIS with a graphical interface for the user. SIMTIS allows us to animate and simulate TIS.
However, TIS didn't got a broader acceptance since it is not a proper extension of LOTOS (or any other established specification technique) and requests therefore for acquiring an additional language (not only additional constructs). Furthermore it was shown that the transformation from LOTOS to TIS is only possible in a restricted manner. Thus TIS forbids the direct derivation of performance-oriented specifications from existing LOTOS specifications.
Besides that, the functionality of a TIS is too complex in order to define completely its semantics. In fact, [16] defines formally only a proper subset of TIS. Both aspects enforced the development of LOTOTIS as an upward compatible extension of LOTOS which offers a complete, well-defined semantics. The main differences between TIS and LOTOTIS are enumerated below.

1. The differentiation of TIS's system composition, system, and process level is lightened in LOTOTIS: a LOTOTIS system and its processes are specified by means of behavior specifications, and thus allowing the use of timed action and any available operator already on "system level". Furthermore, LOTOTIS resources can be defined locally for a process and not only globally for a system. Hence there are no differences between the system and the process level, except that the system level defines the uppermost scope of the LOTOTIS specification. The system composition level is needless for LOTOTIS.

2. LotoTis structured actions correspond to Tis relatively timed actions. There is no concept of absolutely timed actions. Clocks are therefore omitted. However, absolutely timed action can be specified by referring to the actual time or by referring to additional LotoTis processes, that represent clocks.

   LotoTis structured actions have no preparation time since the preparation time of Tis timed actions can always be described by an internal LotoTis structured action with non-zero duration.

3. LotoTis considers structured actions to describe exactly one action, task, process, etc. from the real world. LotoTis uses an action declaration statement for defining the parameters of structured actions and therefore forbids different parameters for the same structured action.

4. LotoTis offers an algebraic data type part that is identical to the data type part of Lotos. Tis offers a functional data type part. Besides the differences between algebraic and functional data types, LotoTis supports additionally the dynamic setting of action parameters during system execution by the use of the data part.

5. Time parameters of LotoTis structured actions are value expressions of the specific type `TIME`, that is used to represent the time domain of a LotoTis specification. On the other hand, preparation and interaction times of Tis timed actions are random variables with certain distribution functions.

6. Instead of supporting the racing operator of Tis, LotoTis offers two timeout operators (one of them resembles the functionality of the racing operator).

   Tis provides the parallel composition operator that enforces the communication in common ports (a slight difference to the Lotos parallel composition operator), and the interleaving operator. It has no parallel operator that allows the explicit specification of the ports that have to synchronize! In contrast to this, LotoTis offers the full synchronization, the parallel composition, and the interleaving operator known from Lotos. Nevertheless, it uses the extended semantics of an interaction from Tis.

The restriction of Tis concepts, that are used within LotoTis on the one hand side and the incorporation of new concepts into LotoTis on the other side resulted in a specification technique with several advantages.

1. LotoTis supports the step-by-step development of specifications by composition of existing specifications.

2. LotoTis has a completely defined formal semantics.

3. LotoTis supports the derivation of performance-oriented specifications from existing Lotos specifications since it is an upward compatible extension of Lotos.

4.2 Lotos-TP

The Lotos extension Lotos-TP [11] is the combination of a timed Lotos extension, called Lotos-T, and a probabilistic Lotos extension, called Lotos-P. Lotos-TP has

a formally defined semantics which permits the evaluation of the system performance by means of performance analysis and of simulation. For the sake of simulation, LOTOS-TP offers the simulation tool TOPO-SIM. TOPO-SIM evaluates the system performance using simulation techniques for discrete event simulation.
Timed behavior is expressed with two new operators.

Timed action prefix: "`a{t}; B`" denotes that action `a` occurs when time `t` has elapsed (relatively to the enabling of the timed action prefix).

Timed successful termination: "`exit{t}`" denotes the successful termination when time `t` has elapsed.

Since the time parameter `t` is a value expression that can be set using the LOTOS data part, it is possible to express any type of time constraints like delays, waiting periods, and timeouts. Hence there are no major differences in the ability to express timed behavior within LOTOS-TP and LOTOTIS.
LOTOS-TP expresses probabilistic behavior by means of experiments and experiment trials. It distinguishes between experiments with

finite experiment space:
"`EXPERIMENT E: sort PROBABILITY P: sort -> sort`" defines an experiment `E` with probabilities assigned to every possible experiment outcome and with

infinite experiment space: "`EXPERIMENT E: sort RANDOMvar v: sort DISTRIBUTION d`" defines an experiment `E` for the random variable `v` with its associated distribution function `d`.

A probabilistic choice is expressed as an experiment trial by the use of the random choice operator

```
random v:  E in B.
```

The random choice operator denotes that the variable `v` takes a value of the experiment `E` space with a probability or according to the distribution function that are given by the semantics of `E`, respectively. The random choice cannot be influenced by the environment, it is a completely internal choice only restricted by the semantics of `E`.
LOTOS-TP experiments can be used to describe probabilistic choices or probabilistic time durations within the timed operators. The semantics of experiments is defined by means of elementary probabilistic theory. This approach is necessary to define the semantics of probabilistic behavior in a clean manner. It is important to point out that this semantics definition exceeds, however, the semantical model of standard LOTOS based on structured labeled transition systems.
When comparing LOTOS-TP with LOTOTIS, LOTOS-TP provides a clean semantics for probabilistic behavior based on distribution functions, the latter offers "only" the probabilistic choice operator. However, the probability in a LOTOTIS probabilistic choice operator can be dynamically set during system execution.
While LOTOS-TP has advantages in describing probabilistic behavior, it lacks the feature of expressing quantified parallelism. It neither supports a concept of priorities nor a concept of resources.
Another aspect of comparing both approaches is their ability to derive performance characteristics of a formally specified system under study. LOTOS-TP defines a simulation by

"`STOCHASTICprocess p IN s`" where

1. `p` denotes a list of performance definitions of the form "`g (m)`" with `m` being a list of performance measures containing `average, variance, maximum` and/or `minimum`, and

2. `s` denotes the LOTOS-TP system specification.

The defined simulation is executed by the simulation tool TOPO-SIM and the performance measures are evaluated. The advantage of this approach is to offer a well-defined syntax for the description of simulations. The advantage is, however, also a drawback since the evaluation of other characteristics of stochastic processes is not supported. Since LOTOTIS does not offer a simulation tool, any comparison makes no sense.
Much more important is the observation that LOTOS-TP does not support the evaluation of performance characteristics that are determined by the occurrence of internal actions (a simulation is only defined for gates which are externally visible actions of a LOTOS-TP specification). However, it is often necessary to define performance characteristics that are based on occurrences of internal actions. Since these performance characteristics are of the same importance for performance analysis as those that are based on gates, LOTOTIS provides the concept of monitoring signals. Monitoring signals make visible the occurrence of any action of a LOTOTIS specification. Hence monitoring signals are the central semantical concept to derive any performance measure of interest.

4.3 ET-LOTOS

Extended-Timed LOTOS (ET-LOTOS [9]) is based on the timed LOTOS extension (T-LOTOS) [2]. Its main intention is the integration of performance analysis and formal verification.
[9] defines a mapping from ET-LOTOS specifications to performance models that are based on finite state transition systems. The performance models have the same observable behavior as the ET-LOTOS specification, so that the mapping to performance models preserves observational equivalence. In addition, LOTOS verification methods are applicable to ET-LOTOS specifications since ET-LOTOS maintains the same formal structure of LOTOS. Consequences of both aspects are:

1. ET-LOTOS can be used to formally verify distributed systems (using the notion of weak bisimulation of LOTOS),

2. verification results in the observational behavior also holds for the derived performance model, and

3. the performance model coincides with the ET-LOTOS specification.

Let us go deeper into details. Whereas T-LOTOS contains the `timer` operator which permits the specification of global and local timing, ET-LOTOS is restricted to the specification of global timings. This decision was made in order to get clear specification of timing constraints at the expense of a reduced compositionality in the language.
The `timer` operator is extended with probabilistic aspects. It is defined as follows:

`timer a<t1, t2, pdf(), priority, weight> in B` where

1. `a` is a gate,
2. `t1, t2` with `t2` $\geq$ `t1` describe the domain of the random variable representing the delay associated with the gate
3. `pdf()` is a function `[t1,t2]` $\longmapsto$ `[0,`∞`)` that describes the characteristic of the pdf of the same random variable with $\int_{t1}^{t2} \mathtt{pdf}(x)dx = 1$,
4. `priority` is the priority to solve nondeterminism in choice expressions in a deterministic way,
5. `weight` is a weight used to solve nondeterminism, that is not solved by priorities, in a probabilistic way, and
6. `B` is a ET-LOTOS behavior expression containing no `timer` operator.

The introduction of random variables to describe time durations on gates is the main semantical extension of ET-LOTOS as compared to T-LOTOS. This extension makes the specification of probabilistic aspects possible and therefore makes ET-LOTOS applicable to performance modeling ([1]).
Furthermore, ET-LOTOS defines two additional timer operators which can describe specific situations. While the pre-synchronization time can be defined by the T-LOTOS `timer` operator using a macro expansion rule, the memory timer is a real semantical extension.

Pre-synchronization timer: A `timer` operator expresses situations in which processes that aim to synchronize in a gate, have firstly to commit their intention to synchronize. Afterwards, they must wait for a given time before actually being committed in the synchronization. However, each of these processes may execute another action during the waiting period, so that the synchronization is aborted.

The `p_timer` operator expresses the other situation. Whenever the processes agreed on a synchronization, they must wait and cannot execute other actions. The waiting period cannot be aborted.

Memory timer: Let us consider a gate whose waiting period is aborted. Normally, the re-enabling of this gate would cause the waiting period to start again, so that all waiting time that passed previously is lost. However, another typical scenario of distributed systems is that the gate resumes its work from the point at which it was interrupted.

The `m_timer` operator describes how at the specified gate a synchronization can take place if the total amount of time during which the gate has been enabled reaches the waiting time. The enabling time is not necessarily continuous, it may be the sum of time durations of several enabling periods.

The three timer operators are the main features of ET-LOTOS. It is important to notice that ET-LOTOS is not a LOTOS extension in the usual sense:

1. It extends only basic LOTOS and excludes the data part.
2. It does not provide all operators of LOTOS. In particular, ET-LOTOS forbids the use of disabling and enabling operators. Henceforth, ET-LOTOS is a "downward" compatible extension with LOTOS in the sense that by neglecting timed and probabilistic extensions one get a standard basic LOTOS specification.

Comparing ET-LOTOS with LOTOTIS we observe that two basic concepts are not contained in this approach: the concept of resources and the concept of monitoring signals.
On the other side, ET-LOTOS offers the memory timer, which makes it possible to describe the preemptive execution of an action. This situation is also expressible within TIS by the use of preemptive resource allocations, but is not expressible in LOTOTIS. The other concepts of ET-LOTOS and LOTOTIS are equally expressive and can be translated into each other.
Besides the question of expressiveness of timed and probabilistic extensions a major difference between ET-LOTOS and LOTOTIS is that the first is a downward compatible extension while the latter is an upward compatible extension of LOTOS.
Last but not least, the main emphasize of ET-LOTOS lies in the numerical analysis of the derived performance model, while LOTOTIS are mainly developed for simulation and formal verification purposes. As this is only a concern of the usage of a specification language, which results in different research directions, it can be expected that research results for ET-LOTOS and LOTOTIS are applicable to the other specification language.

Let us sum up the discussion on related work. To the author's best knowledge, only two LOTOS extensions are comparable to the one presented in this paper, namely [8] and [11][5]. The first, however, is not a proper extension of LOTOS since it excludes the disabling and enabling operator of LOTOS. In addition to a time and a priority/weight concept, the second contains the concept of random experiments to express stochastic behavior. In that respect, it is even more expressive than LOTOTIS since LOTOTIS does not contain any means to specify random variables having certain distribution functions. Both approaches cover quantified time and quantified nondeterminism, but the possibility to express quantified parallelism and action monitoring is not their target.

5 Conclusions

This paper identified the architectural concepts of quantified time, quantified nondeterminism, quantified parallelism, and monitoring as basic concepts for performance evaluation based on formal specification techniques.
Afterwards, it presented structured actions as a powerful concept to describe performance-related issues of distributed systems. We applied structured actions to LOTOS in order to support the standard conform development of distributed systems. The resulting specification technique is called LOTOTIS.
Most importantly, LOTOTIS supports the derivation of performance-oriented specifications from existing LOTOS specification. A number of refinement rules from LOTOS to LOTOTIS were given.
A comparison with related performance-oriented specification techniques finishes the paper. The major advantages of LOTOTIS are are the following: it has (1) a concept of quantified parallelism, (2) a concept of monitoring, and (3) contains a data part. In addition, the comparison identifies one shortcoming of LOTOTIS — the absence of random variables. This shortcoming is lightened by the possibility to express dynamically changing system characteristics by use of the LOTOTIS data part. However, we can see no serious problems when incorporating random variables into LOTOTIS. This would

[5]Please note, that the fairly new approach given in [10] is not considered yet.

require the extension of the LotoTis semantics with aspects from probabilistic theory as it is similar done in [11].

References

[1] G.v. Bochmann and J. Vaucher. Adding performance aspects to specification languages. In *Proc. of the 8th Intern. Symp. on Protocol Specification, Testing and Verification*, pages 19–31, 1988.

[2] T. Bolognesi and F. Lucidi. LOTOS-like process algebras with urgent or timed interactions. In *Participant's Proceedings of the IFIP Intern. Conf. on Formal Description Technique IV*, pages 255–270, 1991.

[3] P. Dembinski. Queueing models for ESTELLE. In *Proc. of the 5th Intern. Conf. on Formal Description Techniques*, pages 73–86, 1993.

[4] K.J. Turner (editor). *Using Formal Description Techniques.* John Wiley & Sons, Chichester, 1993.

[5] N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In *Participant's Tutorial Proc. of the 16th Intern. Symp. on Computer Performance Modelling, Measurement and Evaluation (PERFORMANCE '93), Rome, Italy.* Springer Verlag, 1993.

[6] E. Heck, D. Hogrefe, and B. Müller-Clostermann. Hierarchical performance evaluation based on formally specified communication protocols. *IEEE Transaction on Computers*, 40(4):500–513, 1991.

[7] ISO. Information processing systems - open system interconnection - LOTOS - a formal description technique based on the temporal ordering of observational behaviour. ISO/IEC 8807, 1988.

[8] M.A. Marsan, A. Bianco, L. Ciminiera, R. Sisto, and A. Valenzano. Integrating performance analysis in the context of LOTOS-based design. In *Proc. of MASCOTS'94*, pages 292–298, 1994.

[9] M.A. Marsan, A. Bianco, L. Ciminiera, R. Sisto, and A. Valenzano. A LOTOS extension for the performance analysis of distributed systems. Extended version of [8], 1994.

[10] A. McClenaghan. *Architecture-Driven Specification Using Extended LOTOS.* PhD thesis, University of Stirling, 1994.

[11] C. Miguel, A. Fernández, J.López, and L. Vidaller. A LOTOS based performance evaluation tool. *Computer Networks and ISDN Systems*, 25(7):791–814, 1993.

[12] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In *Proceedings of the Intern. Conference on Computer-Aided Verification (CAV'91)*, pages 1–21, 1991.

[13] L.F. Pires, C.A. Vissers, and M.v. Sinderen. Advances in architectural concepts to support distributed systems design. In *Proc. of the 11th Simp. Brasileiro de Redes de Computadores*, 1993.

[14] H. Rudin. Time in formal protocol specification. In *Proc. of the GI/NTG Conf. on Communication in Distributed Systems, Karlsruhe*, pages 575–587, 1985.

[15] I. Schieferdecker. *Performance-Oriented Specification of Communication Protocols and Verification of Deterministic Bounds of Their QoS Characteristics.* PhD thesis, Technical University Berlin, 1994. (Upon formal approval).

[16] I. Schieferdecker and A. Wolisz. Operational semantics of timed interacting systems: an algebraic performance oriented formal description technique. Technical Report 92/19, Department of Computer Science, Technical University Berlin, 1992.

[17] I. Schieferdecker and A. Wolisz. Structured actions and their use for the design of time-critical distributed systems. In *Participant's Proc. of the 1st Intern. Conf. on Open Distributed Processing (ICODP'93)*, pages 375–281, 1993.

[18] A. Wolisz. A unified approach to formal specification of communication protocols and analysis of their performance. *Journal of Mathematical Modelling and Simulation in Systems Analysis, Special issue on System Analysis in Informatics*, (1993)(10), 1993.