Performance-Oriented Formal Specifications — the LOTOTIS Approach

Ina Schieferdecker

GMD FOKUS, Hardenbergplatz 2, D-10623 Berlin
tel: +49 30 254 99 170, e-mail: ina@fokus.gmd.de,
http://www.fokus.gmd.de/htbin/info/minos/ina

Abstract. The paper presents the performance-oriented, LOTOS extension LOTOTIS. LOTOTIS allows us to specify performance-oriented behavior via quantified time, quantified nondeterminism, quantified parallelism, and action monitoring. It offers a set of refinement rules from LOTOS to LOTOTIS. Therefore, LOTOTIS supports the standard conform development of performance-oriented specifications from existing LOTOS specifications.

1 Performance Evaluation based on Formal Specifications

In the mid eighties [10] there was already recognized that specifying performance requirements can be as important as specifying functional requirements of distributed systems. However, formal specification techniques covered so far primarily functional aspects for the investigation of the functional correctness of distributed systems. Hence, it was (and is still) not unusual that a system is fully implemented or at least implemented as a prototype before any attempt is made to investigate its performance. However, costly re-prototyping or re-implementation can be avoided, if performance could be predicted from the specification of a distributed system. A framework, which supports functional and performance-oriented behavior specification, and which allows us to evaluate performance from that specification would offer major advantages. For this purpose, new concepts have to be incorporated into formal specification techniques. Only recently, there had been published some proposals that close the gap between formal specification and performance evaluation [1].

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 [5]. After introducing the main concepts of LOTOTIS, we present the language and give an insight into its semantics. The most important properties of LOTOTIS are given. The paper finishes with a methodology based on the LOTOS/LOTOTIS framework that supports the development of standard conform prototypes and the prediction of their performance.

2 The LOTOTIS Approach

In order to develop a useful and adequate FDT for performance evaluation, 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 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.

We decided to incorporate these concepts into LOTOS [5] — the Language of Temporal Ordering Specification. It is based on process algebras (CCS, CSP) and algebraic data type specifications (ACT ONE). Another origin of LOTOTIS is TIS — the Timed Interacting Systems Approach ([14],[12]).

2.1 The Notion of Structured Actions

The main feature of LOTOTIS are *structured actions*, which replace classic actions used in process algebras. While classic actions represent task functionalities, structured action with their action parameters represent also performance characteristics of tasks. These action parameters make 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. The parameters of a structured action `a(t,p,r,m)` are the following:

1. The interaction time `t` defines the length of an interaction with synchronization partners[1]. The interaction time makes structured actions in general non-instantaneous, so that they describe true-concurrent behavior.
2. The priority `p` orders simultaneously enabled actions and determines the subsequent behavior. In addition, the access to resources is adjusted by priorities.
3. The set of resources `r` 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 `m` makes action occurrences observable from outside. Whenever an action attached with a monitoring signal starts its interaction period, the monitoring signal offers the signal identifier and the current, absolute time to the 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.

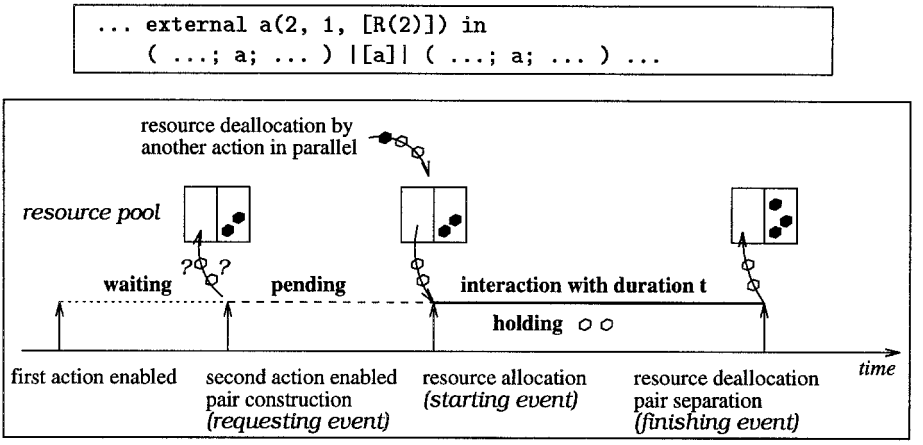

Fig. 1. Synchronization of Structured Actions

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

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

[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; ... )
```

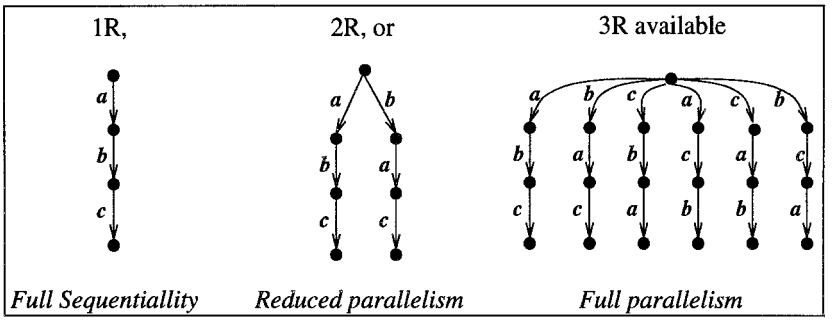

Fig. 2. Independent Structured Actions

pending actions on 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 `p` and the resource discipline[3] basis. After resource allocation the interaction period starts immediately. The monitoring signal `m` informs the environment about the begin of the interaction period. The duration of the interaction period is determined by the interaction time `t`. Resources are occupied throughout the interaction period.

Synchronization Figure 1 depicts the synchronization of two structured actions `a` with interaction time `2`, priority `1`, and two requested resources of type `R`. Synchronization partners become in general enabled at a different point in time. Thus, one action `a` becomes enabled earlier than the other one and has to wait for its synchronization partner before requesting resources. 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.

Independence Independent structured actions, i.e. parallel composed structured actions without synchronization, allow us to model full parallelism, reduced parallelism, and full sequentiality. Figure 2 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}`, `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] For time being, we restrict ourselves to the case of `fifo` (first in first out).

2.2 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. Nondeterminism results from choices of alternative, subsequent behavior. Those choices are either internal or external, that means fully independent of the environment or influenced by its behavior, respectively.

For the case of internal choices, we decided to use a *probabilistic choice operator*. It 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`. External choices that are classically modeled by disabling expressions, can be 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 time `t`, 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 time `t` 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.

2.3 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. timeout operators.

The LotoTis syntax is given in Table 1. The new LotoTis constructs are marked with `->`. 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 process. Therefore, resources can only be allocated by actions declared on the same level as the resources themselves. Every structured action is explicitly

```
   specification:
     specification-symbol specification-identifier formal-parameter-list
     global-type-definitions
->   [time-decl][resource-symbol resource-decls][external-action-decls]
     behavior definition-block endspecification-symbol
-> time-decl:       time-symbol time-domain in-symbol
-> time-domain:     discrete | dense
-> resource-decls: resource-identifier '[' resource-number
                    [',' resource-discipline] ']' [',' resource-decls]
-> action-decls:    action-identifier '(' interaction-time [',' priority
                    [',' resource-request [',' monitoring-signal]]] ')'
                    [',' action-decls]
-> external-action-decls: external-symbol action-decls in-symbol
   process-definition:
     process-symbol process-identifier formal-parameter-list define-symbol
->   [resource-symbol resource-decls]
     definition-block endprocess-symbol
   behavior-expression: ...
->   | hide-symbol action-decls in-symbol
->   | behavior-expression ''[['' probability '']]'' behavior-expression
->   | behavior-expression ''[['' time '']>'' behavior-expression
->   | behavior-expression ''[['' time ''>>'' behavior-expression
     | ...
```

Table 1. The Syntax of LOTOTIS

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.

LOTOTIS operators are 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. An overview about the LOTOTIS operators is given in Table 2.

We distinguish between basic LOTOTIS and full LOTOTIS. Basic LOTOTIS has no data part. Full LOTOTIS is basic LOTOTIS extended with data dependencies. It incorporates algebraic data type 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 additional features of full LOTOTIS are explained in Table 3. They are similar to those features of full LOTOS. A good introduction and guidelines for their use can be found in [2].

Operator	Comment
External: **external** $g_1(..), .., g_n(..)$ **in** $\mathcal{B}$	The external operator declares external gates $g_1, \ldots, g_n$ of a LOTOTIS specification with their parameters.
Hide: **hide** $a_1(..), .., a_n(..)$ **in** $\mathcal{B}$	The hide operator declares internal, hidden structured actions $a_1, .., a_n$ 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: $\mathcal{B}_1 \|[g_1, .., g_n]\| \mathcal{B}_2$	$\mathcal{B}_1$ 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	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 process: **process** $P[g_1,..,g_n]$ $(x_1 : t_1,..,x_m : t_m)$ .. **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**$(x,..)$	Upon successful termination, a list of data values is offered to the subsequent behavior.
Parameterized sequential composition: **exit**$(x,..) \gg$ **accept**$x_1 : t_1,..$ inB	The exit values are passed to the subsequent behavior.
Local value definition: **let** $x : t = v,..$ inB	Variable x of type t is bounded to value v in $\mathcal{B}$.
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

2.4 The Formal Semantics of LOTOTIS

Due to the true concurrent behavior of LOTOTIS, standard LOTOS semantics is definitely inappropriate for defining LOTOTIS. Instead, we use an intermediate specification language which has instantaneous actions and a concept of time prefixing. Thus, time-consuming, structured LOTOTIS actions can be represented as sequences of instantaneous actions with time consumption in between.

The LOTOTIS semantics is defined in two steps:

$$\text{LOTOTIS} \xrightarrow{trans} \text{GENIUS} \xrightarrow{TDS} SLTS.$$

The intermediate specification language is called GENIUS. Roughly speaking, GENIUS is LOTOS with time, priorities, probabilities, monitoring, and time prefix, probabilistic choice, and timeout operators. GENIUS is an upward compatible extension of LOTOS in the sense that it extends LOTOS with additional features while preserving the original LOTOS semantics. LOTOTIS is GENIUS with structured actions and resources. LOTOTIS is transformed to GENIUS in order to define the LOTOTIS semantics. The transformation allows us to consider LOTOTIS to be an upward compatible extension of LOTOS.

The transformation function *trans* maps every syntactically correct LOTOTIS specification L to a GENIUS specification $trans(L)$. Besides the mapping of structured LOTOTIS actions, the transformation from LOTOTIS to GENIUS is used for the explicit definition of the LOTOTIS resource management. This is defined in terms of additional processes and data types contained in $trans(L)$. The formal semantics of GENIUS is defined by an operational semantics. The

transition derivation system *TDS* maps *trans*(L) to a class of structured labeled transitions systems *SLTS*(*trans*(L)) representing the behavior of L.

In order to give an insight into the LOTOTIS semantics definition, Fig. 3 and Fig. 4 present parts of the LOTOTIS to GENIUS transformation and inference rules for GENIUS operators, respectively. The complete LOTOTIS semantics definition is contained in [11].

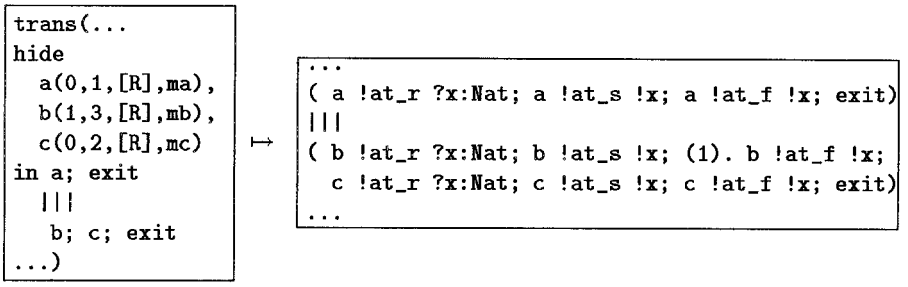

Fig. 3. Mapping of LOTOTIS Actions to GENIUS Actions

The mapping of structured LOTOTIS actions to instantaneous GENIUS actions is exemplarily given in Fig. 3. A structured LOTOTIS action is defined by a sequence of requesting, starting, and finishing GENIUS action. At the requesting action `a !at_r ?x:Nat` the action `a` waits for all its synchronization partners. During the synchronization within this GENIUS action, there is assigned an unique identification number `x` for the tuple of actions. This number is known by all synchronization partners, so that they are the only one that are able to synchronize in the respective starting, requesting, and finishing actions. After the resource request in `a !at_r ?x:Nat`, the tuple of actions willing to synchronize becomes pending until successful resource allocation. The time point of synchronization in `a !at_s !x` marks the successful resource allocation and initiates the start of the interaction period. The duration of the interaction period is defined by means of the time prefix operator. If the structured action is instantaneous (its interaction time is 0), no time prefix is used. The end of the interaction time is marked by a finishing action `a !at_f !x`, which immediately causes all allocated resources to be released. You may wonder, why there is no direct reflection of the other action parameters. In fact, the information on requested resources and monitoring signals is transferred to another process called `Collector` being one of the additional processes for the resource management.

Exemplarily for the transition derivation system of GENIUS we present the axioms for the GENIUS time prefix operator and the inferences for the GENIUS timeout operators. The inferences uses the set $\mathbf{max}\mathcal{B}$ containing all GENIUS actions with maximal priority that are enabled in $\mathcal{B}$[4]. Thus, the priority order

[4] A GENIUS specification defines a priority for every action. The special action χ represents the passage of time and has lowest priority.

of actions is extended to a priority order on behavior expressions.

Soft Timeout

$$\frac{\mathcal{B}_1 \,|t \xrightarrow{g} \mathcal{B}'_1 \,|t \text{ and } name(g) \notin \{g_1, \ldots, g_n, \delta, \chi\} \text{and } g \in \mathbf{max}\mathcal{B}_1}{\mathcal{B}_1[g_1, \ldots, g_n[l]{>}\mathcal{B}_2 \,|t \xrightarrow{g} \mathcal{B}'_1[g_1, \ldots, g_n[l]{>}\mathcal{B}_2 \,|t} \quad \text{(*evolving*)}$$

$$\frac{\mathcal{B}_1 \,|t \xrightarrow{g} \mathcal{B}'_1 \,|t \text{ and } name(g) \in \{g_1, \ldots, g_n, \delta\} \text{ and } g \in \mathbf{max}\mathcal{B}_1}{\mathcal{B}_1[g_1, \ldots, g_n[l]{>}\mathcal{B}_2 \,|t \xrightarrow{g} \mathcal{B}'_1 \,|t} \quad \text{(*saving*)}$$

$$\frac{\mathcal{B}_1 \,|t \xrightarrow{\chi[l']} \mathcal{B}'_1 \,|t \text{ and } \chi \in \mathbf{max}\mathcal{B}_1 \text{ and } l' < l}{\mathcal{B}_1[g_1, \ldots, g_n[l]{>}\mathcal{B}_2 \,|t \xrightarrow{\chi[l']} \mathcal{B}'_1[g_1, \ldots, g_n[l]{>}\mathcal{B}_2 \,|t+l'} \quad \text{(*passage of time*)}$$

$$\frac{\mathcal{B}_1 \,|t \xrightarrow{\chi[l']} \mathcal{B}'_1 \,|t \text{ and } \chi \in \mathbf{max}\mathcal{B}_1 \text{ and } l' = l}{\mathcal{B}_1[g_1, \ldots, g_n[l]{>}\mathcal{B}_2 \,|t \xrightarrow{\chi[l']} \mathcal{B}_2 \,|t+l'} \quad \text{(*timeout*)}$$

Hard Timeout

$$\frac{\mathcal{B}_1[[l]{>}\mathcal{B}_2 \,|t \xrightarrow{g} \mathcal{B}'_{|t'} \text{ and } g \in \mathbf{max}(\mathcal{B}_1[[l]{>}\mathcal{B}_2)}{\mathcal{B}_1[[l \gg \mathcal{B}_2 \,|t \xrightarrow{g} \mathcal{B}'_{|t'}}$$

Fig. 4. The GENIUS Timeout Operators

Likewise to an untimed disable operator, the soft timeout operator represents situations, where the left behavior expression may be disrupted by some exceptional circumstances. However, these exceptions can disable the behavior expression on the left only after time l, i.e. only at a well-defined moment of time. In addition, the left behavior expression cannot be disrupted any more if it executes one of the saving actions $g_1, \ldots, g_n$ or if it terminates. A LOTOTIS soft timeout operator is transformed to a GENIUS soft timeout operator where the saving actions are the respective starts of interaction periods of those LOTOTIS structured actions that are contained in the left hand behavior expression. The hard timeout operator models hard deadlines. Whenever the left behavior expression has not terminated until the hard deadline expires, the timeout will occur and will disrupt the left behavior expression. The hard timeout operator is a special case of the soft timeout operator, since no saving actions exist for the behavior expression on the left. A LOTOTIS hard timeout operator is transformed one-to-one to a GENIUS hard timeout operator.

2.5 The refinement relation between LOTOS and 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 external LOTOS actions with the occurrences of corresponding external LOTOTIS interaction periods. In other words, we only compare the observable behavior of both specifications. Hence, the refinement relation between LOTOTIS and LOTOS can be seen as a weak refinement, which abstracts from internal details of the specifications under study. Obviously, the refined behavior is a subset of the original behavior.

	LOTOS construct	LOTOTIS construct
Structuring actions	External gate a $\longmapsto$ of a LOTOS specification with behavior expression $\mathcal{B}$	**time** ... **in** **resource** R[...],... **in** **external** a(t,p,r,m) **in** $\mathcal{B}$
	hidea in $\longmapsto$	**time** ... **in** $\mathcal{B}$ **resource** R[...],... **in** **hide** a(t,p,r,m) **in** $\mathcal{B}$
Quantifying disablings	$\mathcal{B}_1$ [> $\mathcal{B}_2$ $\longmapsto$	$\mathcal{B}_1$ [[t]> $\mathcal{B}_2$ or $\mathcal{B}_1$ [[t>> $\mathcal{B}_2$
Quantifying choices	$\mathcal{B}_1$ [] $\mathcal{B}_2$ $\longmapsto$	$\mathcal{B}_1$ [[p]] $\mathcal{B}_2$

Table 4. Performance Refinement from LOTOS to LOTOTIS

Three refinement rules from LOTOS to LOTOTIS exist (Table 4). Structuring external and/or internal actions comprises the transformation of defining action parameters for a given action — defining its interaction time, its priority, its resources, and/or its monitoring signal. It assumes that the time domain and the used resources are properly declared. Quantifying disablings is the transformation of defining a time parameter for a disabling operator. This reduces the possibilities, when the disruption can occur. The third transformation is the parameterization of choice operators that weights the alternatives of the choice expression with probabilities for their occurrences. The following theorem can be proven.

Theorem 1. *Structuring actions, quantifying disablings, and quantifying choices in a* LOTOS *specification yield* LOTOTIS *specifications, which are refinements of the original* LOTOS *specification.*

Furthermore, if we define the underlying LOTOS specification of a LOTOTIS specification to be the specification that results from omitting all additional

LOTOTIS features (by application of the inverse transformation rules of Table 4), the following lemma holds.

Lemma 2. *Every* LOTOTIS *specification is a refinement of its underlying* LOTOS *specification.*

For a formal proof of the refinement relation as well as of the subsequent LOTOTIS properties please refer to [11].

2.6 Further Properties of LOTOTIS

Upward compatibility of LOTOTIS with LOTOS comprises of two properties:

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.

The proof for upward compatibility is mainly based on the fact that the intermediate specification language GENIUS that is used for the LOTOTIS semantics definition, is upward compatible with LOTOS.

Another important aspect of formal specifications is that of their finiteness. Finiteness is often an essential precondition for the application of verification methods. The refinement relation between LOTOTIS and LOTOS allows us to proof the following theorem.

Theorem 3. *Every guarded* LOTOTIS *specification, whose underlying* LOTOS *specification is finite, is finite.*

Therefore, the finiteness conditions for LOTOS yield finiteness conditions for LOTOTIS. The following lemma can be formulated [3].

Lemma 4. *A guarded* LOTOTIS *specification is finite if the following conditions are fulfilled by its underlying* LOTOS *specification.*

1. *It is guarded,*
2. *it does not contain relabeling,*
3. *enabling is never involved within recursive calls in a process and the processes composed sequentially are finite,*
4. *if enabling is present within a recursive call, then at least its left argument is finite and does not contain the recursive call,*
5. *the disabling operator is only an outermost operator and its arguments are finite,*
6. *if disabling is involved in a recursive call then its left argument is finite, and*
7. *there exists no recursive calls within parallel compositions.*

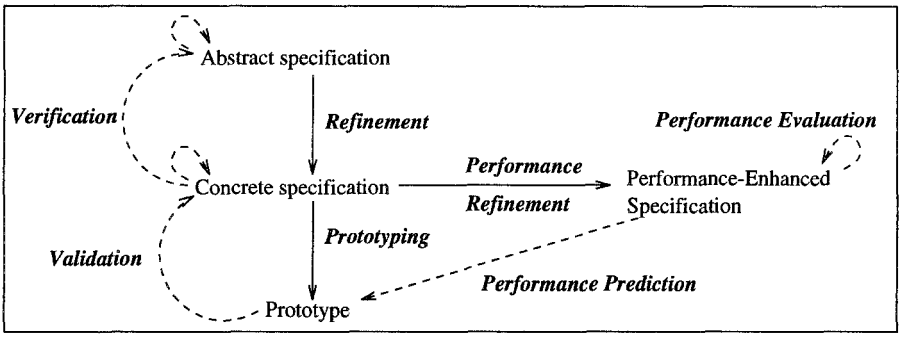

Fig. 5. The Performance-Oriented Prototyping Process

2.7 Predicting the Performance of LOTOS Prototypes

This section presents a methodology for the standard conform development of prototypes of distributed systems whose performance can be predicted in dependence of different execution environments (Fig. 5).

How to develop functionally correct distributed systems from an (untimed) formal service specification down to a system implementation is a well studied area. The development process for distributed systems starts with an abstract specification which reflects the main functionalities (services) offered to the user. Refinement techniques were developed to support the design of formal specifications from abstract to more concrete specifications [2]. A concrete specification describes the mechanisms for realization of the system functionalities. Verification methods are used to proof the functional correctness of these specifications or to proof the functional coincidence between abstract and concrete specifications by means of equivalence or contained-in relations [7]. Afterwards, prototypes are implemented for a first investigation of the system behavior in real environments. Prototype derivation techniques support the (semi)automatic development of prototypes from a concrete, implementation-oriented formal specification [6]. Validation techniques can be used to (semi)formally prove that prototypes (and subsequent implementations) meet their specifications [4].

However, it is still common that the timing behavior and the performance is investigated only when a prototype or a first implementation of the system exist. In the case of inconvenient performance, the whole development process is restarted, what results in long periods of system development, late system delivery, and high costs. Moreover, there will be no guarantee that the newly designed system will have better performance than the first one. Therefore, we suggest to use the technique of performance refinement that has been introduced above for predicting the performance of prototypes of distributed systems. The methodology is based on the LOTOS/LOTOTIS framework. It starts with an abstract functional LOTOS specification of the distributed system. This specifi-

cation can be refined in order to get a concrete, implementation-oriented LOTOS specification of the system. Afterwards, the concrete specification is enhanced with the performance characteristics of the system tasks in a given execution environment. Let us give two examples for the appropriate parameterization of a specification. Basic real-time parameters of system tasks such as duration of inter-process communication or the access to the storage has to be measured before incorporating them into the specification. Secondly, real resource requirements of system tasks are directly reflected by the resource parameter of structured actions. General rules of how to incorporate performance characteristics of execution environments into formal specification are given in [13]. By applying the performance refinement rules we get a LOTOTIS specification from which we can derive performance measures. We derive simulation models from LOTOTIS specifications. For this purpose, we use the close relationship between transition systems and discrete event systems. A simulation tool is currently under development, where we use our experiences from the TIS simulation tool [13].

Obviously, this is only a first step in order to support the performance-oriented development of distributed systems based on formal specifications. A lot of work has still to be done.

3 Conclusions

This paper resembled the concepts of quantified time, quantified nondeterminism, quantified parallelism, and monitoring to be 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.

To the author's best knowledge, only two LOTOS extensions [8], [9] are comparable to the one presented in this paper. The first, however, is not a proper extension of LOTOS since it excludes the disabling and enabling operator of LOTOS. The second contains besides a time and a priority/weight concept a concept of random experiments to express stochastic behavior. In that respect, it is 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. The absence of random variables in LOTOTIS is however lightened by the possibility to express dynamically changing system characteristics by the use of the LOTOTIS data part. In addition, we can see no serious problems when incorporating random variables into LOTOTIS. This would require the extension of the LOTOTIS semantics with aspects from probabilistic theory as it is similar done in [9].

Finally, we presented an approach to predict the performance of distributed systems already during the system design phase. Based on a LOTOS specification of the functional behavior, a performance-enhanced LOTOTIS specification is

derived. Simulation models of the LOTOTIS specification yield the performance estimates of interest. In case of low performance, the distributed system can be re-designed before any prototyping or implementation efforts are spent.

References

1. P. Dembinski. Queueing models for ESTELLE. In *Proc. of the 5th Intern. Conf. on Formal Description Techniques*, pages 73–86, 1993.
2. K.J. Turner (editor). *Using Formal Description Techniques.* John Wiley & Sons, Chichester, 1993.
3. A. Fantechi, S. Gnesi, and G. Mazzarini. How much expressive are LOTOS behaviour expressions? In *Paricipant's Proc. of the Third Intern. Conf. Formal Description Techniques*, pages 9–24, 1990.
4. G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall Software Series. Prentice-Hall, Englewood Cliffs, New Jersey, 1991.
5. ISO. Information processing systems - open system interconnection - LOTOS - a formal description technique based on the temporal ordering of observational behaviour. ISO/IEC 8807, 1988.
6. Guy Leduc. A framework based on implementation relations for implementing lotos specifications. *Computer Networks and ISDN Systems*, (25):23–41, 1992.
7. F.J. Lin, P.M. Chu, and M.T. Liu. Protocol verification using reachability analysis. *Computer Communication Review*, 17(5):126–135, 1987.
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. 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.
10. H. Rudin. Time in formal protocol specification. In *Proc. of the GI/NTG Conf. on Communication in Distributed Systems, Karlsruhe*, pages 575–587, 1985.
11. 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).
12. 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.
13. M. Walch. A framework for performance analysis of parallel protocol execution. In *Participant's Proc. of the IFIP Intern. Conf. on Information Networks and Data Communication*, Madeira Island, Portugal, 1994.
14. 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.