30

Quality of Service Management in Distributed Systems using Dynamic Routation

Leonard J.N. Franken[a], Peter Janssen[a],
Boudewijn R.H.M. Haverkort[b] and Gidi van Liempd[a]

[a] PTT Research, P.O. Box 15000, 9700 CD Groningen, the Netherlands
[b] University of Twente, P.O. Box 217, 7500 AE Enschede, the Netherlands

With the advent of multimedia in computing and communication systems, a new set of requirements has been imposed. Because multimedia applications demand a guaranteed Quality of Service (QoS), the sharing of resources, by means of allocating applications and their communications on computing and communication resources, is an important issue. In this paper, we address the issues of communication routing and process allocation in an integrated way, the so-called *routations*. It is only in such an integrated way that *end-to-end Quality of Service* requirements of multimedia end-users can be fulfilled.

For the determination of a routation, we propose to use the A^* algorithm, an intelligent tree-search algorithm, in combination with a heuristic, $H3''$. The A^* algorithm always finds an optimal solution if one exists, whereas the heuristic algorithm comes up with (sub)optimal solutions. The heuristic, however, accomplishes the same task as the A^* algorithm with far reduced computational effort. With the combination, denoted as $A^*/H3''(x)$, the A^* algorithm will, when time is available, find an optimal solution, and in case of limited time the heuristic will calculate a suboptimal solution. This combination has been found suitable for a real time multimedia environment by a thorough statistical analysis after testing the heuristics on 1750 randomly generated test cases.

Finally, we present an experimental multimedia, ANSAware-based, distributed system in which the routations take place using the heuristic we developed. To demonstrate the capabilities of the routation algorithms the *xallocator* has been implemented. The *xallocator* guarantees the QoS using heuristic routation algorithms and dynamic reconfiguration whenever a change in the distributed environment occurs.

Keyword Codes: C.2.4; D.4.4; D.4.5; G.1.6; I.2.0; I.2.8
Keywords: Quality of Service; Performance of Systems; Communications Management; Reliability; Optimization; AI General; Problem Solving, Control Methods and Search

K. Raymond et al. (eds.), *Open Distributed Processing*

1. INTRODUCTION

In large current-day distributed systems, such as e.g., multimedia telecommunication systems like videophones, the sharing of resources, by means of allocating a wide variety of applications and their communications on a pool of computing and communication resources, is an important issue for Quality of Service guarantees. Because computing and communications are both important for current-day discrete and continuous end-to-end applications, we advocate an integrated approach towards the allocation of processes and the routing of communications as well. In this paper we present this integrated approach by the use of a so-called *routation algorithm* that can be used in the context of the *performability manager* introduced by Franken *et al.* [1, 2].

The performability manager, a distributed system component, guarantees and maintains the quality of service (QoS) requirements of an application in a distributed system [3]. On the basis of a model-based procedure, the performability manager decides which configuration is put into effect, thereby using dynamic reconfiguration facilities of the underlying supporting computing platform. The ultimate goal in using a model-based reconfiguration approach is to be sure that the reconfigurations performed do result in a desired change of the end-user perceived QoS. This goal, however, is in general too ambitious to be reached in an environment in which (mild) real-time requirements are present as well. It is simply infeasible to evaluate all possible alternative configurations. Therefore, we split the generation process of possible alternative configurations in two steps. We first generate a number of reasonable alternatives, based on static system and application properties and constraints, i.e., in this step we do not address issues related to queueing. The intend is, however, to create the alternatives as reasonably as possible. Then, in a second and more time consuming step, we automatically create queueing models for the candidates selected in the first step, evaluate them, and decide then which alternative configuration has to be put into effect. This second step has already partly been addrssed in [2].

This important issue of routing and allocation has been dealt with before [4, 3, 5, 6, 7]. However, we address the issues of routing and process allocation in an *integrated* way. This is a desirable approach as it is often unpractical to separate these highly connected issues. For the configuration creation we present routation algorithms based on an intelligent tree search algorithm, the A^* algorithm, known from artificial intelligence theory [8] as well as some heuristics.

This paper is further organized as follows. In Section 2 we present and evaluate (test) the A^* algorithm and the derived heuristics. In Section 3, we discuss a prototype implementation of a routation component in an ANSAware-based distributed environment using a videophone application. Section 4 concludes the paper.

2. COMBINED ROUTING AND ALLOCATION: ROUTATION

We start this section with the explanation of the basic terminology for routation algorithms in Section 2.1. After that, we discuss the algorithms we used and tested their performance on the 1750 test cases in Sections 2.2 and 2.3 respectively.

2.1. Notation and terminology

When determining a valid routation for a distributed environment, properties and constraints of the elements of the distributed environment are to be used. The elements are the application components, dataflows, devices, communication links and communication paths. In Figure 1, we graphically present a small distributed environment. The rectangles full of lines represent the application components, and the arrows between them the logical interaction between application components. Application components are executed on devices, which are interconnected via a transport network.

The properties and constraints of the elements of the distributed environment provide the algorithms with sufficient information for the search to a routation. When trying to find an optimal routation, an objective function is needed. We define these here as follows[1]:

property A property is a statement about a characteristic of an element, which can be identified by examining the element.

constraint A constraint is a statement about an element that restricts the routation of one set of elements (e.g. applications) onto another set of elements (e.g. computing and communication resources)). If a certain routation violates a constraint, that routation is an invalid mapping.

objective function An objective function assigns a value to each routation in such a way that a more preferred routation is assigned a higher (or lower) value. Objective functions thus rank routations.

To each element in Figure 1 a *descriptor* is attached which contains the name of the element, the properties of the element (given in the left half of the descriptor) and the possible constraints on it (given in the right half of the descriptor). The following properties and constraints are used (the subscript x denotes the element to the which property/constraint belongs):

Properties: et_x: execution time, er_x: execution rate, l_x: load, s_x: size (memory required), pr_x: precedence relations, v_x: volume, pol_x: processing overhead local communications, por_x: processing overhead remote communications, t_x: type (certain device type required), m_x: memory available, pc_x: processing capacity, fr_x: failure rate, rr_x: repair rate, c_x: cost, tr_x: transmission rate, td_x: transit delay, cl_x: number of communication links.

Constraints: d_x: device (specific device required), rt_x: response time required, rel_x: reliability required, av_x: availability required, c_x: max cost allowed, cl_x: number of communication links required, lb_x: lower bound load, ub_x: upper bound load, m'_x: available memory on a device, ac_x: upper bound on number of application components, df_x: upper bound on number of dataflows.

The objective of the optimization algorithms is to maximize the probability that a failure does not occur while the devices and communication paths are active. For every device the probability that it does not fail when it is processing the application components assigned to it is computed. For example, the time that an application component ac is using a device d is defined by $et_{AC}(ac,d) = \frac{et_{AC}(ac)er_{AC}(ac)}{pc_D(d)}$. For device d under routation $\mathcal{RA}$ this probability $R_d(\mathcal{RA})$ is equal to (with $x_{ac,d}$ equal to 1 if application component ac is assigned to device d, and 0 otherwise):

$$R_d(\mathcal{RA}) = \exp\left(-\sum_{ac=1}^{|AC|} x_{ac,d} et_{AC}(ac,d) fr_D(d)\right) \tag{1}$$

[1] In the literature, many different terms are used to indicate characteristics of an element (e.g., parameter [9, 10]), limitations on assignments (e.g., constraint [5, 11] or requirement [12]) and functions that need to be optimized (e.g., cost function [4, 13] or performance goal [4]).

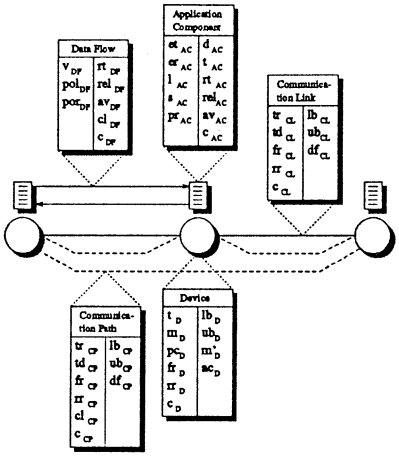


Figure 1: View on distributed environments with respect to routations

For the processing overhead for (local and remote) communications, expressed by R_{por}, and communication paths transporting data, expressed by R_{cp}, a similar expression can be derived [6, 7, 14, 15]. Since for an infinite time horizon these probability tend to zero, they are calculated for a certain time period. Using these probabilities, $R(\mathcal{RA})$ represents the probability that a failure does not occur during the period of length t in which the devices and communication paths are processing or transmitting the application components and dataflows assigned to them by routation $\mathcal{RA}$. The total reliability $R(\mathcal{RA})$ can then be calculated as follows (under the assumption these probabilities are independent and the failure times are exponentailly distributed):

$$\begin{aligned} R(\mathcal{RA}) &= (\prod_{d=1}^{|D|} R_d(\mathcal{RA}))(\prod_{d=1}^{|D|} R_{por}(\mathcal{RA}))(\prod_{cp=1}^{|CP|} R_{cp}(\mathcal{RA})), \\ &= \exp(-F_R(\mathcal{RA})). \end{aligned} \tag{2}$$

Maximizing the probability $R(\mathcal{RA})$ is equivalent to minimizing the function $F_R(\mathcal{RA})$. The function $F_R(\mathcal{RA})$ is the objective function our algorithms try to minimize in order to find an optimal routation.

2.2. The algorithms and their performance

For an efficient and effective solution of the routation problem we present a smart combination of a heuristic and the A^* algorithm. We first preseent the A^* algorithm, then a smart heuristic and then a combination of these two.

The A^* algorithm The A^* algorithm searches for an optimal routation by representing it as the (intelligent) search for a minimal-cost path in a tree [8]. Each node in the search

tree represents a different, *partial* (not all application components and dataflows are yet assigned), routation. An edge along the path represents the assignment of an application component or dataflow.

In the partial routation the first components and the first dataflows are assigned. This implies that the set of applications and dataflows are ordered. We assume that not every mapping exists, but that only those exist in which all dataflows that are assigned are used by already assigned application components.

The root node represents an empty routation, i.e. no application components nor dataflows assigned, all leaf nodes represent complete routations, i.e. all application components and dataflows have been assigned. The basic cycle of operations of the A^* algorithm starts with the selection of a node from a set of not yet expanded nodes, called the *OPEN* set. Initially, only the root node is in *OPEN*.

If the node selected is a leaf node, we have found a complete routation. Under a certain condition on the selection mechanism (see below), we can guarantee that the first leaf node encountered represents an optimal routation, at which point the A^* algorithm may terminate.

If the selected node is not a leaf node, A^* will *expand* the node, i.e., generate all children of that node. A child node represents the partial routation of the parent node, with an extra application component or dataflow assigned. We take care to generate only child nodes that do not violate constraints. The children are then inserted into the *OPEN* set, and A^* will select the next node to expand.

Choosing the node to expand is done using a heuristic evaluation (objective) function $f(n)$ (our $F_R(\mathcal{RA})$), which is calculated for every node n upon generation. The node in *OPEN* that has currently the lowest value for $f(n)$ is called the *most promising node*, and this is the node that A^* selects for expansion next. Note that the nodes in the set *OPEN* do not have to be all of the same level, and that the most promising node is not always one of the nodes of the deepest level in *OPEN*. This implies that backtracking is allowed. In our application, $f(n)$ represents an estimate of the cost of a complete optimal routation.[6, 14]

The heuristic $H3''$ algorithm The $H3''$ algorithm is based on two other algorithms, the $H1''$ and the $H2''$ algorithm. $H3''$ first determines two routations, one in the way algorithm $H1''$ does, the other in the way $H2''$ does. From these two routations the best is chosen. These algorithms themselves are derived from the $H1'$ and the $H2'$ algorithm which again are derived from $H1$ and the $H2$. In this section we start with the $H1$ and the $H2$ algorithm, which are based on algorithm 4 of Shatz [6] and describe the succesive improvement made whichs lead to the $H3''$ algorithm. The details of this improved versions are presented by Franken *et al.* [14].

The $H1$ routation algorithm first sorts the application components in decreasing order of total volume of communication with other application components. The second heuristic routation algorithm, $H2$, sorts the application components in decreasing order of execution time per second. Hereby, we treat both communication and processing-oriented applications in a fair way. Algorithms $H1$, $H2$ sort the application components and assign the application components and the dataflows one by one to the devices and communication paths. Backtracking is not allowed, so once a decision has been made, it cannot be turned back. At each decision point that pair of device and set of communication paths is chosen that has the minimum increase in the objective function and that does not violate any constraint. But it could be that in this way a device d is already occupied, by a number of application components that do not *have* to be assigned to d, at the moment an application component

ac having a *device constraint* $d_{AC}(ac) = d$ will be assigned. In such a case, the algorithms can not determine a valid routation although one might exist.

For application components having a device constraint we can use *advance reservations*. Adding these reservations to algorithms $H1$, $H2$ and $H3$ results in the enhanced heuristic algorithms $H1'$, $H2'$ and $H3'$ for which many of the device-constraint related problems in the basic algorithms can be circumvented. Running the enhanced heuristic algorithms on a number of cases showed that using reservations for application components having a device constraint resulted indeed in a higher number of solved cases. But in cases that an application component ac has a *device type constraint* $t_{AC}(ac)$ and it was assigned as one of the last application components, it could be that all devices of the required type or types were already occupied by other application components, not depending on these devices. Using reservations will not solve this completely, since an application component could still be assigned to more than one device.

A solution is the use of a *possible set* for each application component. This set contains all devices to which an application component can still be assigned. First for each application component the possible set is determined, using all constraints on the application components. Then, at each decision point all possible sets are checked and reservations are made for those application components having a possible set containing just one device. After each decision, all possible sets are adjusted according to the assignment made. This is another form of *constraint propagation* [16, 17]. Adding possible sets to algorithms $H1'$, $H2'$ results in the re-enhanced heuristic algorithms $H1''$, $H2''$ and $H3''$.

The combined algorithm: $A^*/H3''(x)$ The A^* algorithm determines optimal routations, but it has an exponential worst case time complexity and sometimes it runs out of memory. Algorithm $H3''$ has a polynomial time complexity, but it determines (sub)optimal routations and it does not always find a routation, although one exists. If a (sub)optimal routation does not have to be determined as fast as possible, but an algorithm is allowed to spend a certain time searching for an optimal routation, a combination of algorithms A^* and $H3''$, denoted as $A^*/H3''(x)$, can be used to find a routation. If the A^* algorithm finds a routation within x seconds, this routation will be the result of algorithm $A^*/H3''(x)$. If after x seconds, however, the A^* algorithm does not find a routation, it is stopped and a routation will be determined by algorithm $H3''$.

2.3. Results

The algorithms have been tested on 1750 cases. These cases were divided into seven classes of 250 cases, each class having a specific number of application components (4, 6 and 8) and a specific number of devices (3, 4 and 5). The properties and constraints were, within a range of allowed values, completely randomly generated for each case. Giving every application component and dataflow every possible constraint would result in a high number of cases for which a routation does not exist. Therefore we used probabilities for the existence of every ⟨application component , constraint⟩-pair and ⟨dataflow , constraint⟩-pair during the generation of the cases. For the same reason the cost constraints c_{AC} and c_{DF} were not used. In [14] the ranges of allowed values for all properties and constraints, together with the mentioned probabilities, are given. For the comparison of the algorithms we use the following criteria ; the *number* of found routations, the *quality* of the determind routations,

and the *time* needed to determine routations. For the comparison of the quality we use the relative error $d(H)$. Thie relative error for algorithm H is equal to

$$d(H) = \begin{cases} \dfrac{R(\mathcal{RA}_{A^*}) - R(\mathcal{RA}_H)}{R(\mathcal{RA}_{A^*}) - R(\mathcal{RA}_{\overline{A^*}})}, & \text{if } R(\mathcal{RA}_{A^*}) \neq R(\mathcal{RA}_{\overline{A^*}}), \\ 0, & \text{if } R(\mathcal{RA}_{A^*}) = R(\mathcal{RA}_{A^*}), \end{cases} \quad (3)$$

where A^* is an adjusted version of the A^* algorithm that determines the valid routation with the *highest* instead of the lowest value for the objective function. The relative error $d(H)$ ranges from 0 to 1, and is equal to 0 if algorithm H determined an optimal routation, and equal to 1 if algorithm H determined the worst case routation.

The number of routations found by the algorithms A^*, $H3''$ and $A^*/H3''(x)$ for the various classes of cases are given in Table 1. For $A^*/H3''(x)$ we vary x over to 1, 2, 5, 10, and 15.

Table 1: Number of found routations by A^*, $H3''$ and $A^*/H3''(x)$

			$A^*/H3''(x)$				
	A^*	$H3''$	$x=1$	$x=2$	$x=5$	$x=10$	$x=15$
$\|AC\|=4, \|D\|=3$	194	188	193	193	193	193	193
$\|AC\|=6, \|D\|=3$	135	111	125	127	129	130	130
$\|AC\|=8, \|D\|=3$	45	36	38	39	40	41	41
$\|AC\|=6, \|D\|=4$	180	166	173	175	177	178	180
$\|AC\|=8, \|D\|=4$	132	107	108	109	112	116	117
$\|AC\|=6, \|D\|=5$	194	189	189	190	192	194	194
$\|AC\|=8, \|D\|=5$	162	155	155	155	155	156	158
Total	1042	952	981	988	998	1008	1013

The relative errors $d(H)$ of the routations found by algorithm $H3''$ and $A^*/H3''(x)$ are given in Table 2 as well as the percentage of the found routations that have a relative error $d(H)$ of at most 10%.

Table 2: Average $d(H)$ and percentage $d(H) \leq 0.1$ of algorithms $H3''$ and $A^*/H3''(x)$

			$A^*/H3''(x)$									
	$H3''$		$x=1$		$x=2$		$x=5$		$x=10$		$x=15$	
	$d(H)$	≤ 0.1	$d(H)$	≤ 0.1	$d(H)$	≤ 0.1	$d(H)$	≤ 0.1	$d(H)$	≤ 0.1	$d(H)$	≤ 0.1
$\|AC\|=4, \|D\|=3$	0.0168	96%	0.0023	99%	0.0005	100%	0.0002	100%	0.0000	100%	0.0000	100%
$\|AC\|=6, \|D\|=3$	0.0602	79%	0.0144	95%	0.0082	97%	0.0045	98%	0.0029	99%	0.0027	99%
$\|AC\|=8, \|D\|=3$	0.0963	64%	0.0613	80%	0.0592	81%	0.0505	84%	0.0229	91%	0.0176	94%
$\|AC\|=6, \|D\|=4$	0.0704	78%	0.0459	87%	0.0320	90%	0.0200	93%	0.0106	96%	0.0054	98%
$\|AC\|=8, \|D\|=4$	0.1144	62%	0.1031	64%	0.0962	65%	0.0656	76%	0.0410	85%	0.0291	89%
$\|AC\|=6, \|D\|=5$	0.0469	83%	0.0364	87%	0.0320	88%	0.0222	91%	0.0097	97%	0.0055	98%
$\|AC\|=8, \|D\|=5$	0.0849	71%	0.0832	71%	0.0821	71%	0.0724	73%	0.0541	78%	0.0374	85%
Total	0.0594	80%	0.0408	86%	0.0352	88%	0.0261	91%	0.0160	94%	0.0107	96%

From these tables it follows that for larger distributed environments, the increase in the quality of the determined routations is less than for smaller distributed environments, when x increases. This could be expected, since the larger the distributed environments, the more

time the A^* algorithm needs to find a routation, i.e., the A^* algorithm exceeds in more cases its time limit. This means that in more cases the routation determined by algorithm $H3''$ is used, which determines suboptimal routations.

Table 3: Average time in seconds needed to find a routation by A^*, $H3''$ and $A^*/H3''(x)$

			$A^*/H3''(x)$				
	A^*	$H3''$	$x=1$	$x=2$	$x=5$	$x=10$	$x=15$
$\|AC\|=4, \|D\|=3$	1.193	0.029	0.129	0.160	0.214	0.271	0.322
$\|AC\|=6, \|D\|=3$	37.369	0.045	0.537	0.795	1.281	1.891	2.350
$\|AC\|=8, \|D\|=3$	120.010	0.056	0.887	1.519	3.167	5.236	6.836
$\|AC\|=6, \|D\|=4$	14.754	0.086	0.773	1.202	2.023	2.810	3.364
$\|AC\|=8, \|D\|=4$	276.780	0.113	1.086	1.987	4.149	6.734	8.617
$\|AC\|=6, \|D\|=5$	13.399	0.161	0.968	1.560	2.826	4.140	4.967
$\|AC\|=8, \|D\|=5$	128.626	0.200	1.185	2.132	4.761	8.551	11.678
Total	70.349	0.105	0.758	1.260	2.441	3.900	5.006

The times needed to find a routation by algorithms A^*, $H3''$ and $A^*/H3''(x)$ are given in Table 3 (in seconds). From this table it follows that the larger the distributed environments become, the more algorithm $A^*/H3''(x)$, on average, approaches or even exceeds its time limit allowed for algorithm A^*. If, on average, the time limit is exceeded, this indicates that the routation determined by algorithm $H3''$ is often the result. A careful choice of the time limit is therefore necessary in order to use the advantages of algorithm $A^*/H3''(x)$.

3. ROUTATION IN A MULTI-PARTY VIDEO-PHONE

To demonstrate some of the capabilities of the heuristic routation algorithm $H3''$ the *xallocator* has been implemented in an experimental distributed system providing a multi-party videophone service. The experimental distributed system is described in Section 3.1 and the *xallocator* is described in Section 3.2.

3.1. The distributed environment of the multi-party video-phone

The multimedia application used in the demonstration is a *multi-party video-phone*. This application has been implemented using *ANSAware*. Video recorded at one computer system, using a video camera and a special video card, is sent to two other computer systems, which perform a similar task. For this test we equipped three computer systems. At each computer all three video streams are displayed.

The application consists of a number of application components: three video-phones, three duplicators, a stream manager and a stream factory. These application components and the dataflows between them are shown in Figure 2.

At each computer to which a video camera is attached, a video-phone resides. Each video-phone sends the video signal it records to a duplicator, which sends this data to the other two video-phones. The stream manager and the stream factory are used for managing

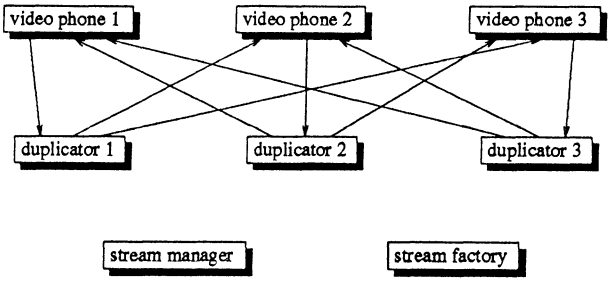


Figure 2: The application components and the dataflows of the multi-party video-phone

the video streams between the video-phones and the duplicators, but have, after starting up the application, no interaction with the other application components.

The properties of the application components are shown in Table 4. These properties are theoretically determined approximations of the real properties; actual monitoring of the application components has not been performed. The three video-phones have equal properties, just like the three duplicators.

Table 4: Properties of the application components

	et_{AC} (ms)	er_{AC}(/s)	l_{AC}(/s)	s_{AC}(Kbytes)
video-phone	20	20	0.4	3072
duplicator	10	20	0.2	1024
stream manager	1	1	0.001	1024
stream factory	1	1	0.001	1024

Except for the video-phones, the application components originally do not have any constraints. The video-phones all have a device constraint d_{AC}, since they interact with the user of a specific device. Reconfiguring the distributed environment according to a new routation means migrating one or more application components and/or dataflows. Since the only application components that can be migrated are the duplicators, we assign a device constraint d_{AC} to the stream manager and the stream factory. The dataflows all have identical properties and no constraints. Since each frame has a size of at most 8 Kbytes, the volume v_{DF} of each dataflow is equal to 160 Kbyte/s. The processing overhead por_{DF} for each dataflow is assumed to be 40 ms/s.

The distributed system used for the demonstration consists of a number of different Sun workstations connected by an Ethernet. The properties of and the constraints on these devices as used in the demonstration are given in Table 5. Note that the devices do not have a cost property c_D. The memory constraint m'_D is set to 1, meaning that all memory present is available.

The values presented in these table do not intend to represent the actual properties of and constraints on the devices. The devices are fully connected, i.e., a communication path exists between each pair of devices. The properties of and constraints on these communication paths allow all possible routings for the dataflows of the multimedia application.

Table 5: Properties of and constraints on the devices

	t_D	m_D (Kbytes)	pc_D	fr_D(/s)	ub_D	ac_D
sun012/015/039	Sparc10/Parallax	16384	5.0	5^{-5}	0.8	16
sun027/029	Sparc IPC	8192	1.0	5^{-5}	0.8	16
sun035	Sparc ELC	8192	0.75	5^{-5}	0.8	16

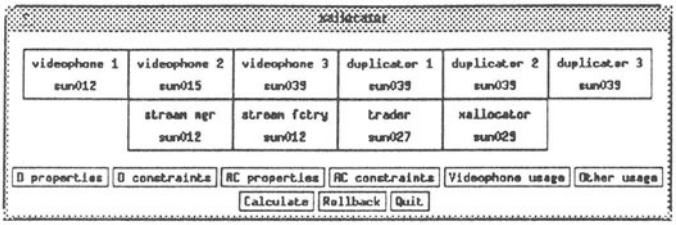
3.2. Overview of and experiences with xallocator

The *xallocator* is an application that allows the user to view the current allocation of the application components of the multimedia application, change the properties of and the constraints on the devices and simulate the presence of other applications in the distributed environment, by changing the usage of the devices. After one or more changes to the distributed environment have been made, a routation is determined using algorithm $H3''$. If the current routation is no longer valid or the just determined routation has a higher reliability, the distributed environment is reconfigured according to this new routation. In Figure 3, an example view of the main screen of *xallocator* is shown. This screen contains the current allocation of the application components and a number of buttons. A description, in terms of properties and constraints, of an application components or a device will be shown after selecting it. Note that the allocation of two other application components is shown as well: that of the trader and the *xallocator* itself.
After using *xallocator* some time, we noticed that the possible assignment of the duplicators to other devices after changing one or more values very well visualizes what a routation algorithm is capable off. The time needed for a migration is considerably larger than the times needed to determine a new routation. In all cases the time needed for a reconfiguration should be minimized and un-noticable by the users. However, there will always be a trade-off between serious QoS degradation and a short unavailability (because of a reconfiguration). Futhermore, despite the fact that, considering the changes in the properties and constraints, only one duplicator had to migrate, sometimes a new routation was determined for which all three duplicators had to migrate. The reason for this is that when there is more than one device that increases the objective the least, the algorithm chooses the first of these devices to assign an application component to.

One possible solution to this problem is to exchange the first device in the set of devices with the device to which application component ac is currently assigned, just before assigning application component ac when determining the new routation.

4. CONCLUSIONS AND FURTHER RESEARCH

In this paper we have addressed the issues of routing and allocation in distributed systems. In particular, we addressed these issues in an integrated way, the so-called routations, in order to ensure an optimal realization of the end-to-end quality of service. We then proposed an optimal routation algorithm based on the A^* algorithm and a heuristic, $H3''$, derived from the A^* algorithm. A smart combination of the A^* algorithm with the $H3''$ heuristic gives us a very good routation algorithm, called $A^*/H3''(x)$. This algorithm can be used in an



Pressing a button results in the following actions: **D properties**: An overview of the properties of the devices will be shown. Selecting one of these properties allows the user to modify this property; **D constraints**: An overview of the constraints on the devices will be shown. Selecting one of these constraints allows the user to modify this constraint; **AC properties**: An overview of the properties of the application components will be shown. These properties cannot be modified; **AC constraints**: An overview of the constraints on the application components will be shown. These constraints cannot be modified; **Video-phone usage**: An overview of the usage of the devices by the application components of the multi-party video-phone is shown; **Other usage**: An overview of the usage of the devices by the other, simulated, applications in the distributed environment is shown. Selecting one of these usages allows the user to modify this usage; **Calculate** A routation for the current settings of the distributed environment is determined. If no routation can be determined, or if the current routation is still valid and has a reliability equal to the reliability of the just determined new routation, this is reported. If a new routation is determined that is closer to optimal than the current routation or the current routation is no longer valid, the distributed environment is reconfigured according to this new routation. The dynamic reconfiguration is performed; **Rollback** The changes made to the distributed environment since the last determination of a valid routation are discarded. **Quit** Terminates *xallocator*.

Figure 3: An example view of the main screen of xallocator

environment with real-time requirements. This claim is proven to be true by a statistical analysis of 1750 randomly selected test cases. The lower the timing requirements, the more this algorithm operates as a normal A^* algorithm, i.e., the better it becomes. Also, the higher the timing requirements, the more it acts as a pure heuristic algorithm. In this way, this algorithm combines the best of both worlds, in an adaptable fashion.

We applied the $H3''$ algorithm on an ANSAware-based videophone application using the *xallocator*. It appeared that routation is useful but more work is needed on the algorithms as well as on the migration aspect of application components.

As topics for future research, we envisage the following. The current routation algorithms optimize only one objective function. Different applications, however, will have different QoS requirements. Therefore, research is needed to allow for the use of different objective functions for different applications during the creation of a single routation.

Another topic of interest is the use of incremental or adaptable routations. The aim of these is to allow for the (stepwise) addition or removal of new applications without re-routating, i.e., reallocating and rerouting, the existing environment, thereby still satisfying all QoS requirements.

With respect to the dependability and reliability aspects of the QoS requirements, research is needed towards strategies for the routation of replicated applications [18].

A final topic we would like to mention is research towards distributed or decentralised routation algorithms. The algorithms described in this paper are fully centralised. In order

to achieve acceptable performance in larger systems, decentralisation might be a good way to go.

References

[1] L.J.N. Franken and B.R.H.M. Haverkort. The Performability Manager. *IEEE Network: The Magazine of Computer Communications, Special Issue on Distributed Systems for Telecommunications*, 8(1):24–32, Januari 1994.

[2] L.J.N. Franken, R.H. Pijpers, and B.R. Haverkort. Modelling Aspects of Model Based Dynamic QoS Management by the Performability Manager. In G. Haring and G. Kotsis, editors, *Computer Performance Evaluation. Modelling Techniques and Tools. Proceedings of the 7th International Conference, Vienna, Austria*, pages 89–110. Lecture Notes in Computer Science, Springer-Verlag, Volume 794, May 1994.

[3] T.C.K. Chou and J.A. Abraham. Load Balancing in Distributed Systems. *IEEE Transactions on Software Engineering*, SE-8(4):401–412, July 1982.

[4] N.S. Bowen, C.N. Nikolaou, and A. Ghafoor. On the Assignment Problem of Arbitrary Process Systems to Hetrogeneous Distributed Computer Systems. *IEEE Transactions on Computers*, 41(3):257–273, March 1992.

[5] W.W. Chu, L.J. Holloway, M. Lan, and K. Efe. Task Allocation in Distributed Processing. *IEEE Computer*, 13(11):57–69, November 1980.

[6] S. M. Shatz, J. Wang, and M. Goto. Task Allocation for Maximizing Reliability of Distributed Computer Systems. *IEEE Transactions on Computer*, 41(9):1156–1168, December 1992.

[7] C.M. Woodside and G.G. Monforton. Fast Allocation of Processes in Distributed and Parallel Systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):164–174, February 1993.

[8] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1990.

[9] D.P. Anderson. Metascheduling for Continuous Media. *ACM Transactions on Computing*, 11(3):226–252, August 1993.

[10] K.G. Shin, C.M. Krishna, and Y. Lee. Optimal Dynamic Control of Resources in a Distributed System. *IEEE Transactions on Software Engineering*, 15(10):1188–1197, October 1989.

[11] K. Efe. Heuristic Models of Task Assignment Scheduling in Distributed Systems. *IEEE Computer*, 15(6):50–56, June 1982.

[12] J. P. Huang. Modeling of Software Partition for Distributed Real-Time Applications. *IEEE Transactions on Software Engineering*, 11(10):1113–1126, 1985.

[13] S.M. Shatz and J-P. Wang, editors. *Tutorial: Distributed Software Engineering.* IEEE Computer Society Press, 1989.

[14] L.J.N. Franken, P. Janssens, B.R.H.M. Haverkort, and E.P.M. Van Liempd. Dynamic Routation in Distributed Environments. Submitted for publication, 1994.

[15] J. Laprie and K. Kanoun. X-Ware Reliability and Availability Modeling. *IEEE Transactions on Software Engineering*, 18(2):130–147, February 1992.

[16] V. Kumar. Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine*, 13:32–44, Spring 1992.

[17] P. Meseguer. Constraint Satisfaction Problems: An Overview. *AI Communications*, 2(1):3–17, March 1989.

[18] L.J.M. Nieuwenhuis. *Fault Tolerance Through Program Transformation.* PhD thesis, University of Twente, 1990.