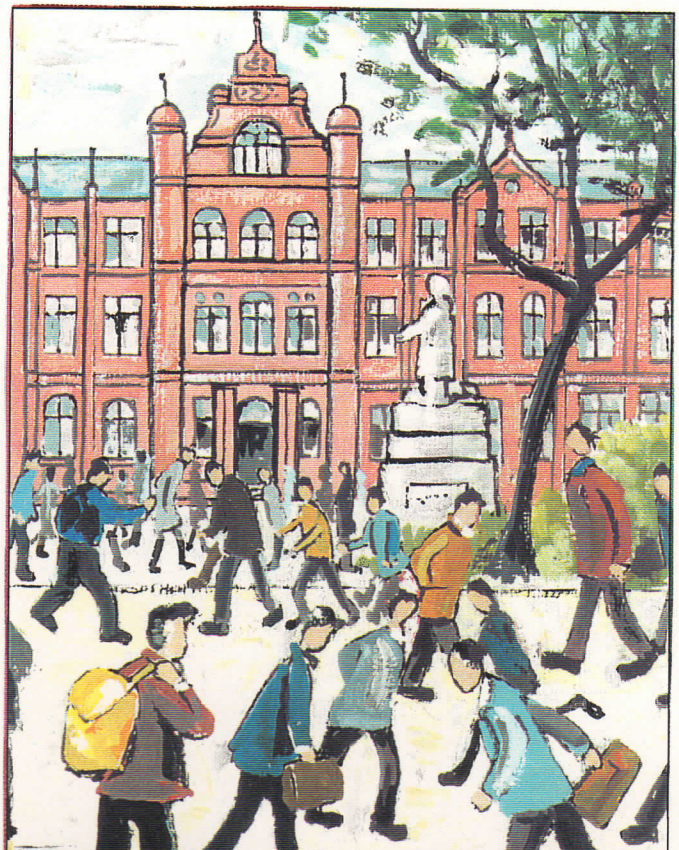
G. v. Bochmann

OPERATIONAL RESEARCH SOCIETY

First European Conference on Intelligent Management Systems in Operations

Edited by Khairy Kobbacy, Sunil Vadera, and Nathan Proudlove

University of Salford, U.K.
25-26 March 1997



NEW MODELS FOR APPLYING AUTOMATIC RECONFIGURATION IN NETWORKS AND DISTRIBUTED SYSTEMS

Petre Dini, Gregor v. Bochmann, Raouf Boutaba
Computer Science Research Institute of Montreal
1801, McGill College street, #800
Montreal, (Qc), H3A 2N4, CANADA
e-mail: {dini, bochmann, rboutaba}@crim.ca

ABSTRACT:

Automatic reconfiguration is a new challenge to manage complex distributed systems. Current approaches to specify such systems are not able to capture behavioral aspects to prevent the service degradation. This article introduces three elements which are relevant for the automation of configuration management: (i) refined standardized state change models with particular states imposed by the need of prevention (usage and operational states), or the need of creation, migration, and deletion of managed objects (administrative state), (ii) the notion of actual availability which allows us to identify predictive reconfiguration activities based on the lifecycle history of a managed object, and (iii) the notion of required environment for a managed object, distinguishing the functional, operational, and existential environment of a managed object. This characterization of the environment is defined according to the new state change models and environmental conditions proposed in this article. Examples of using our proposal in reconfiguration policies are presented.

1. INTRODUCTION

A management system is an application which consists of specialized managing objects playing different management roles, such as monitoring, fault detection, or reconfiguration. Automatic reconfiguration and automatic reconfiguration management require new mechanisms, techniques, and approaches to collect, interpret, and re-act in an appropriate manner to reconfiguration needs. The main purpose of this article is threefold: (1) to refine the existing state change models, (2) to propose a dynamic quantitative evaluation of a operational behavior of a managed object, and (3) to identify environmental constraints related to the administrative and operational state change models. The goal is to offer accurate and relevant information, which can be used within management policies which are implemented as the functional behavior of managing objects. This allows an active and automatic management of distributed systems. A management system can apply reactive policies, as a consequence of an event occurring within a managed object, or pro-active policies, commonly using prediction mechanisms based on the behavioral history of a managed object. Reactive and pro-active policies are used to prevent the degradation of the QoS (Quality of Service) or to ensure a graceful degradation of QoS. The result of these policies is an enhancement of the configuration of a system that can be performed by either (1) the selection of the best servers with respect to the clients' requests, or (2) the creation, migration, isolation, and deletion of managed objects. Our proposal can be summarized as follows.

State change models: Different classes of managed objects have a variety of state attributes. A change of at least one state attribute value determines a state change for the concerned component. Automation and distribution need some refinements of existing state change models, in order to combine state changes and alarms. For example, if the operational state is enabled, different alarms can occur. According to their relevance, we classify them in three categories, i.e. warning, critical or outstanding. Each event category enriches the semantics of the operational state, giving a more accurate operational view on a component. For those managed objects portraying a maximum capability, the capability range is thresholded between idle to busy, to accurately capture the loading. Consequently, several counter-based events concerning the usage state are evaluated with respect to two thresholds, which lead to a warning or critical usage state. We define the relation is-better-than, which creates an ordering between system components offering identical or similar services.

actual availability: Different alarms, e.g. critical alarms, could predict a future degradation. Based on the last notification (state change or alarm), the management system can prevent such a degradation. We

propose the notion of *actual availability* and *a formula* to compute it. This allows us to identify conditions for predictive reconfiguration activities, based on the lifecycle history of a managed object. A combination of new state models and the actual availability allows us to define the notion of *health* of a managed object, as a unified measure on the quality of the managed object behavior. Values of the actual availability, which is defined as the fraction of time when a component has been enabled since its initialization, can be stored and interpreted within time series models. These models are based on historical actual availability data, and allow to a management system to extrapolate future behavior or identify behavioural event patterns.

Required environment: Based on notifications (state changes or alarms) or measurements (actual availability), a management system can predict a local degradation. Consequently, in order to prevent this degradation or to enhance the QoS, a manager can apply reconfiguration actions, such as the substitution, migration, isolation, or re-initialization of managed objects. These actions may include the creation, deletion, or re-installation of managed objects. All these actions are subject of environmental constraints, i.e. existential or functional constraints. The set of objects which must be available before the creation of an object constitutes the existential environment of this object. Another set may act as a pre-condition of the enabled operational state. Each reconfiguration management activity must verify that the constraints of the required environment are satisfied. Consequently, we introduce a package containing specific attributes related to the required environment.

The structure of the paper is as follows. We focus in Section 2 on state changes and notifications. Section 3 defines different kinds of required environments, according to the state change models. In Section 4, the actual availability of a managed object is defined. Section 5 presents several management policies using our proposal. The conclusion summarizes our proposal.

2. STATE CHANGE MODELS

Internet [1], OSI Management [2], and TINA [3] have proposed distinct state change models. Management aspects are separately represented in Internet and OSI Management. In the Internet approach the value space is neither finite nor standardized, while in OSI Management, state change models are standardized, but not sufficiently refined to express service degradation. Additionally, the administrative state value space {*locked*, *unlocked*, and *shutting-down}* allows to automatically reconfigure clients and servers, but does not offer accurate information on configuration aspects. In TINA, management changes are embedded in functional changes, i.e. there is no separation between functional and management behaviors.

2.1. Operational state change model

Operational state changes (enabled/disabled) are internally decided by a system resource. In many cases, the state *enabled* does not accurately represent the real state of a component. In order to capture the real situation of the operational state, we refine the state enabled. If a component is enabled, different alarms having various degrees of severity could be sent as notifications. We classify alarms into three severity levels: *warning alarms*, *critical alarms*, and *outstanding alarms*. The behavioral model of a system component concerning these alarms is shown in Figure 1.

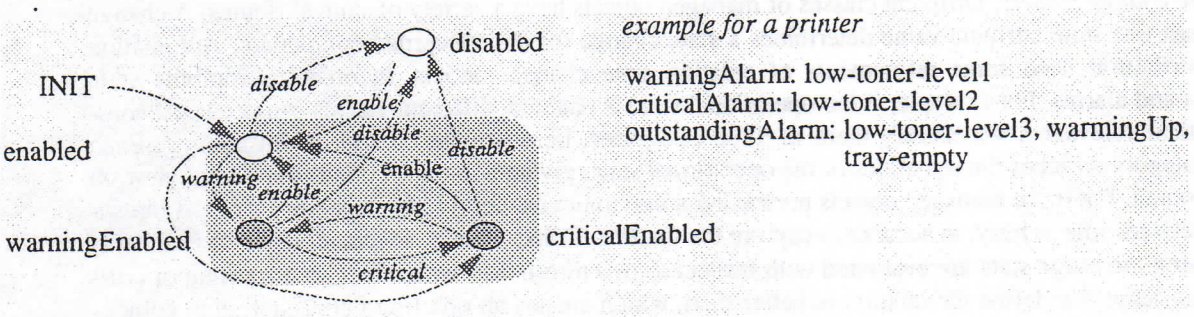


Figure 1. State transitions diagram of the operational state change model

2.2. Usage state change model

Usage state changes are internally decided by a system resource. *Idle* and *busy* states are well-defined with respect to the behavior of the appropriate managed object. In many cases, the state *active* does not accurately represent the real charge of a component, which is a relevant aspect in automatic reconfiguration. Among probable causes of alarms we mention *threshold crossed* and *storage-capacity problem* which refer to the usage capacity. Consequently, according to changes of its load, and with respect to *threshold1* and *threshold2*, the component is either in the *warningActive* state, or *criticalActive* state, as shown in Figure 2. If a new user is served at the limit of the *maximum capacity*, or the maximum capacity decreases, the usage state becomes *busy*.

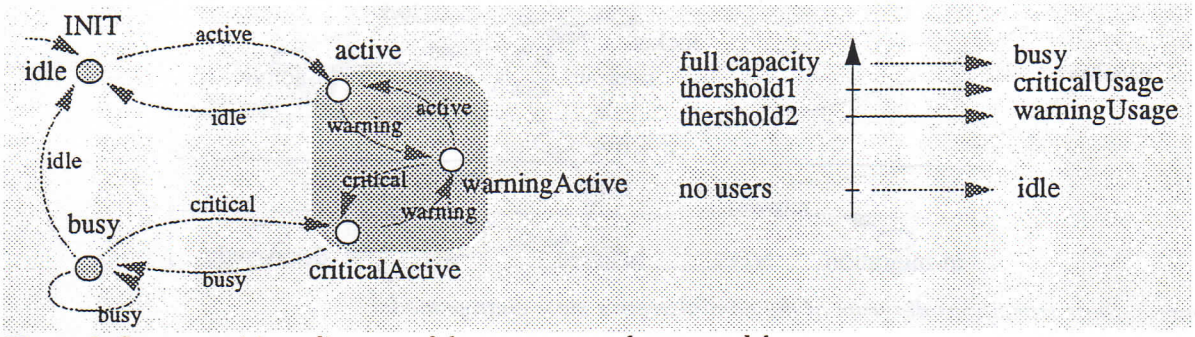


Figure 2. State transitions diagram of the usage state change model

2.3. Administrative state change model

The administrative state reflects the component state with respect to management decisions. For managing in a manual manner, the OSI proposal to describe the administrative state offers complete information. Either a component is *unlocked*, which means that its services can be used by any system component, or it is *shutting-down*, when only the current clients still access these services. For reconfiguration purposes, the state *locked* suffices to ensure the liberty of administrative actions taken by a system operator. However, procedures to create, initialize, insert, remove, migrate, or isolate a system component are part of the system operator's expertise. In distributed systems which are managed by many system operators, the operators' expertise frequently in conflict because a component may be locked with different goals. Since the current administrative state change model does not cover the complexity required by the automatic reconfiguration, we introduce three distinct phases concerning the administrative state:

phase 1: object creation, insertion, and initialization;
phase 2: current administration of an exiting object, as described by standards, and
phase 3: object removing, destruction, or migration.

In the phase 1, a managed object is prepared to administratively become *available*. We have identified two states, *isolated* and *initialized*, which precede the state where object's services are available. As presented later, each managed object type has its required environment attributes. A managed object is first *created,* and *added* to the system (or domain) as an isolated object, being in the state *isolated*. In this phase, the required environment refers to the existential context. For example, a kernel needs memory space, a type of processor, and an external memory, while a file needs memory. We consider that the managed object is inserted in the system, but still isolated. Only its existential environmental requirements are satisfied. It is registered within the appropriate MIB (Management Information Base), but not in the service repository, because its services are not yet available.

If a managed object is already within a domain (i.e. its insertion has been performed), and its services have to be used, the state becomes *initialized.* During this state, the operational environmental requirements are satisfied. Three activities are in particular performed by the management system in this state: (1) management parameters are initialized, i.e. the operational state, usage state, and actual availability are originally set, (2) objects conforming to the required types forming the operational environment are identified and either reserved or connected to the given object, and (3) the new instance is registered within the service repository of the concerned management domain, and if it is the first instance of a new

type, i.e. no other instances of the same type exist within the concerned management domain, a special managing object (commonly an extended trader) updates the service repository.

An object may be available, when all required cooperation relations corresponding to its required environments are created. Cooperation relations represent object interactions through object interfaces [4].

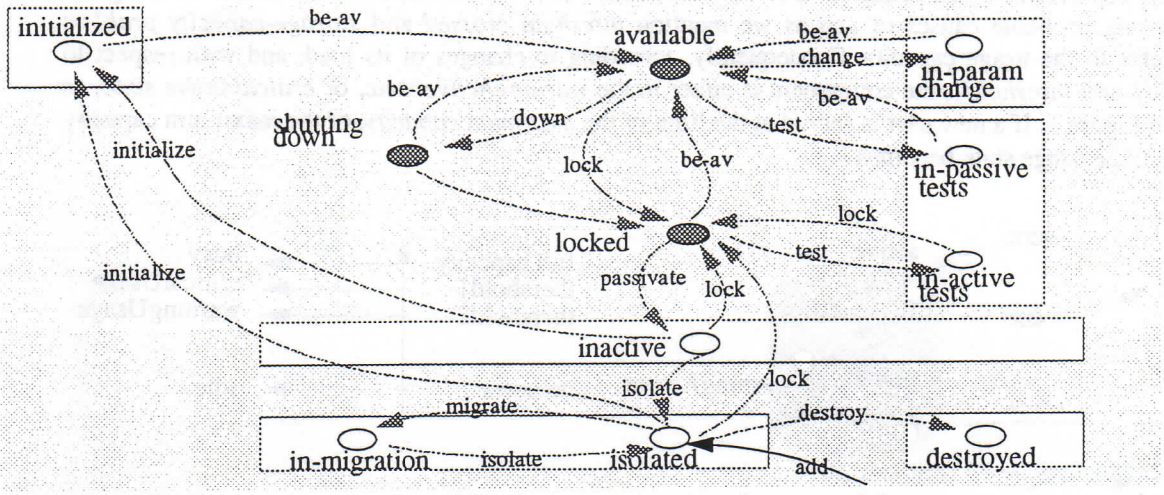


Figure 3. State transitions diagram of the administrative state change model

In the phase 2, when all of these actions have been performed, the managed object administratively becomes *available*. If no other reconfiguration actions occur, this is a stable state, allowing to the potential clients to use services of the concerned managed object, according to its usage and operational states. The model presents two sub-states, called *in-passive-tests*, and *in-param-change*. We recall, that the object model we have considered presents multiple interfaces. The test interface is independent of the functional interface. Consequently, several passive tests, can be applied, without disturbing the normal services. For example, the evolution of the actual availability can be periodically tested. This state differentiates those managed objects that are subject of particular management activities, such of test activities. In the state *in-passive-tests*, the reconfigurator must create and monitor particular cooperation relations concerning the tested and testing object. These cooperation relations are commonly initiated by the managing system. The state *in-param-changes* has been introduced to clearly distinguish a state where the properties are in the process of change. As an example, a node could be in this state during the reconfiguration activity of removing one of its telecommunication interfaces. Although the node is still operationally *enabled*, its usage state is *active* or possible *criticalActive* after this reconfiguration activity, the number of interfaces changes. As a conclusion, the group of these three states, i.e., *available, in-passive-tests,* and *in-param-changes* represents the lifecycle period where there are no administrative restrictions to access services of the concerned managed object. Services are accessible strictly according to operational and usage states of a managed object.

From the state *available*, a managed object can transit to either the *shutting-down*, or *locked* states. In our model, the semantics of these states is similar to that adopted by the OSI Management approach. The administrative state becomes *shutting-down* when no other new users are administratively allowed to access the services of the concerned managed object. Notice that there is no relation to a refusal due to the usage state, if this state is *busy*. The state *locked* is reached from *available*, or *shutting-down*. The first case is commonly used when the managed object must be actively tested, or for reconfiguration purposes implying configuration changes within the system (see the next group of state values).

The state *locked* may serve either to apply several active tests, or purely the use of the concerned managed object is prohibited. But differently to the state *in-passive-tests*, the managed object can not offer its own services when it is in the *in-active-tests* state. If the test results are processed by special test analysers viewed as managed objects, additional cooperation relations may be required, according to the required environment of test analysers. From *shutting-down* or *locked*, a managed object may become *available*. In all these three states, cooperation relations concerning the required environment of the related man-

aged object are created and activated.

The last group of states (phase 3) concerns specific aspects useful for reconfiguration in distributed systems. In the state *inactive*, the component related to the managed object is not available. All of its cooperation relations, except the obligatory cooperation relations, are normal terminated, i.e. due to reconfiguration actions, and services are not enabled. This is not the case of *locked*, where all previous cooperation relations exist. While *locked* can be reached for different administrative goals, such as the enhancement of system stability due to many alarms issued by the concerned managed object, the state *inactive* represent a pre-state of a reconfiguration activity. Commonly, a managed object reaching this state has necessarily had the *shutting-down* state, up to its usage state becomes *idle*. The main goal is to reconfigure the managed object (isolate, remove, or migrate). Some external reconfiguration constraints may revoke the initial decision, i.e. an alarm which announced an imminent failure of a node is cancelled by the fault manager. Also, a reconfigurator which has inactivated a managed object for migrating it elsewhere, may reconsider its decision; it can re-*initialize* this object, according to Phase 1.

The remaining three states are called *isolated, in-migration*, and *destroyed*. When a managed object is *inactive*, all its functional cooperation relations are terminated, except the obligatory cooperation relations established with objects belonging to the existential required environment. Let us assume that the managed object must be removed from the system. Consequently, the state *destroyed* is reached. In this case, all obligatory cooperation relations corresponding to the operational required environment are terminated. When the object is effectively destroyed, i.e., completely deleted from the system, even its obligatory existential cooperation relations are terminated. A garbage collector effectively eliminates the managed object from the system. From the isolated state, a managed object can enter the *in-migration* state. This state is necessary to eventually prepare the migration services. A management system can decide to remove or migrate a managed object, according to high-level reconfiguration decisions. When a migration from a domain to another is decided, the state becomes *in-migration*. This state allows to transient domains to recognize the status of a temporary managed object. The reconfiguration can succeed or fail; consequently, the managed object becomes isolated, either in the original domain (failure), or in the target domain (success). The state reached from *in-migration* is *isolated.*

The state management function ensures state transitions, as presented by the state change model diagram. Once in a state, particular reconfiguration services fulfil additional activities concerning the state of cooperation relations.

3. REQUIRED ENVIRONMENT

Each managed object has certain properties defined by its type, according to the relation *is-instance-of.* At the instantiation, each type property is initialized according to the type definition. This definition must clearly specify the types and the cardinality of the set of managed objects forming the required environment of each instance. However, it is possible that at the instantiation of a managed object, some objects of its required environment do not exist. This has the following consequences:

1. If missing required objects belong to the *test environment*, the new object can not be tested;
2. If missing required objects belongs to the *management environment*, the new instance can not be automatically managed;
3. If missing required objects belongs to the *existential environment*, the new instance can not be created;
4. If missing required objects belongs to the *operational environment*, the new instance can be created, but it can not be in the operational state enabled.

Consequently, the existential environment represents hard constraints which are pre-conditions of the creation or the existence (after the creation of the concerned object), whereas the operational environment are hard pre-conditions for the provision of services by the object in question. Obviously, if all these environmental requests are satisfied, the new object can be created, tested, managed, and it provides its own services. If, after the creation of an object, an interaction with one of its environmental objects can not be maintained, e.g. an environmental object is destroyed, locked, or disabled, the operational or administrative states of the previous object follow our models, according to the definition of environmental types. If we consider the administrative state change model, the environments required for each administrative state are presented in Figure 4. We observe that there are administrative states where only certain kinds of re-

quired environment are necessary. For example, if the administrative state is isolated, only the requirements for the existential environment remain.

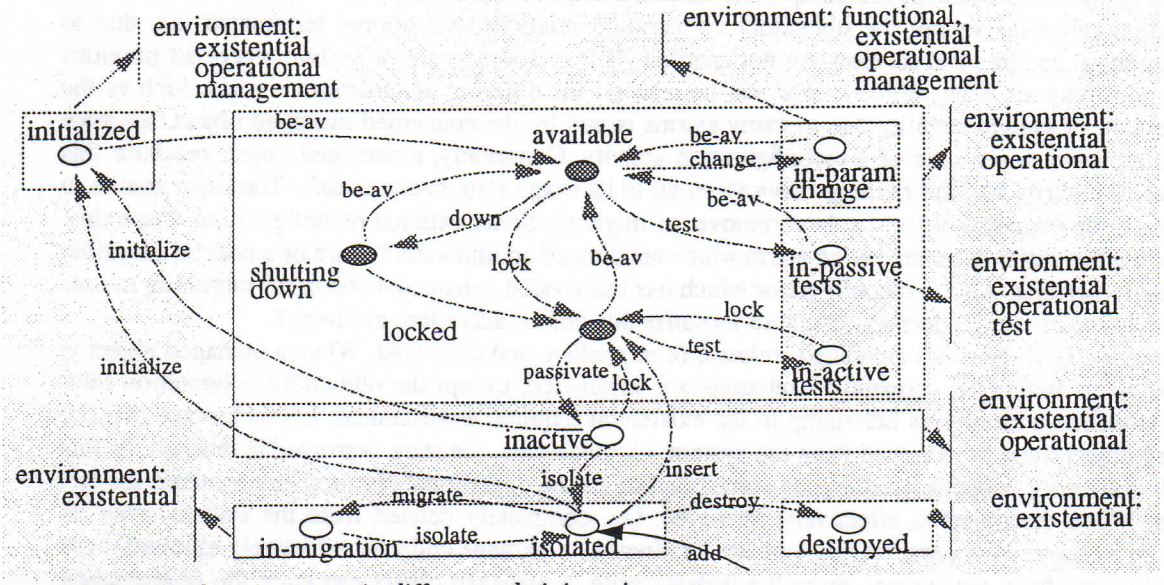


Figure 4. Required environment in different administrative states

Note:
According to these environment types, the interaction between this new instance and any object is called test, management, existential, and operational, respectively. It is not sufficient that the objects of the object's environment exist, but also, the cooperation relations between this object and its environmental objects must be accordingly created. Also, the state of these cooperation relations must allow the requested services. Also, objects playing roles within required environments of a concerned managed object may have different operational, usage or administrative states. Hence, it is not guaranteed that the environmental needs will be effectively satisfied. This leads to consider the cooperation relations and environmental objects together. Several conditions must be accomplished in order to create the environmental satisfaction of a managed object, and to maintain it, as presented in [5].

4. ACTUAL AVAILABILITY

The actual availability of a system component at time t, written a(t), represents the availability of the component's services up to the given time. Its value, defined by formula (I), represents the fraction of time this component has been in the operational state enabled after an event, called start-period, occurred. As shown by formula (I), the actual availability value is a continuous function of time, defined as a quotient between the amount of time where the resource has been in the enabled state (commonly called operational time) and the observation period (between the time t_0 and the time t of the end of the observation period).

$T = [t_0, \infty]$, t_0 is the timestamp where the measuring period begins (I)

a: $T \rightarrow [0, 1.0]$

$$a(t) = \frac{1}{t - t_0} \int_{t_0}^{t} \lambda(t)\, d\, t, \quad \text{where } \lambda(t) = \begin{cases} 1 & \text{if operational state at time t is } \textit{enabled}, \text{ and} \\ 0 & \text{otherwise} \end{cases}$$

For management purposes, this formula must be evaluated at any time $t \in T$. However, polling responses and notifications are issued at specific times $t_i \in T$. We assume that each state change event is represented by <event type, timestamp>, where timestamp is the time when the state change occurred (notification), or a polling request has been issued (management command).

$$a(t) = \frac{1}{t - t_0} \quad (a(t_i) \times (t_i - t_0) + \lambda(t^{(-)}) \times (t - t_i)) \qquad \text{(II)}$$

where t_i is the time of the last state change before the time t, and
$t^{(-)}$ means just before t.

The actual availability is calculated each time an *state change event occurs*, or *on-demand* at the initiative of managing objects, i.e. between two state changes, using the formula II above. This formula allows us to compute the actual availability a(t), by knowing the actual availability $a(t_i)$ at the time t_i of the last computation. Based on formula (II), a management system will calculate the actual availability value each time a state change event occurs or a polling action is performed. Whereas this case is natural, a preventive computation by polling can be performed by the management system before a new state change event occurs. This case is frequently used with two goals. First, for the *preventive control*, a management system updates the actual availability regardless whether a state change occurred. Second, if a *critical customer* requests services offered by a system resource, the management system must evaluate the actual availability of the provider in order to choose the best solution. To accurately adapt reconfiguration policies to the real state of a component, other derived parameters may be considered [6][8].

5. MANAGEMENT POLICIES

5.1. Monitoring policies

Monitoring policies may independently use criteria based either on the actual availability, operational state, or usage state. Let us consider the polling operation performed by a manager with a variable frequency to collect the real state of a managed object. This frequency is increased when the concerned component behaves abnormally, in order to capture new information in order to prevent a degradation [7]. A simple monitoring policy based on the actual availability can be expressed by the policy P1:

P1: if $a(t) < a_0$
then
pollingFrequency = f (a(t))
else
pollingFrequency = f_0

where a_0 is a threshold of the actual availability defined by the management system for those system components playing critical roles for particular applications, f_0 is a basic polling frequency, and f(a(t)) is an updated polling frequency according to the actual availability. When the actual availability decreases, the polling frequency is increased to capture possible degradation of QoS offered by the appropriate component. Policy P2 presents another combination between the notions defined in this paper.

P2: if operationalState = warningEnabled,
and
usageState = criticalActive
then
pollingFrequency = max {f (a(t)), g(usState)} [7]

5.2. Selecting the most available server

Commonly, establishing cooperation relations implies many kinds of constraints, expressing the requested QoS or the current state of the system resources which must interact. In distributed systems, identical or similar services can be offered by many resources. Mainly, QoS issues concerning static properties of potential cooperating resources, offered as interface constraints by their appropriate managed objects, have been presented in [8]. In the following, we emphasize QoS constraints related to the real performance of cooperating objects. The problem is, how to select the most available server for a client, based on the actual measures and models proposed in this article. We define several relations for comparing different system components, from the management point of view, with respect to their current availabilities, operational and usage state values. We write "*C1* $\succ$ *C2*" (read *C1 is-better-than C2)* to indicate that C1 is in

some sense better than C2. Clearly, we can order the operational and usage state values as follows:

enabled ≻ *warningEnabled* ≻ *criticalEnabled* ≻ *disabled;*

idle ≻ *active* ≻ *warningActive* ≻ *criticalActive* ≻ *busy.*

We define the *health* of a component C with respect to its weighted current availability and its operational state, as $h_C(t) = < opState, \bar{a}(t)>_C$. We then can define several decision policies, for comparing system components, based on their health and usage state as follows:

The management policy *operational-state-first,* can be defined as:

$h_{C1}(t) \succ h_{C2}(t)$ iff $((opState_{C1} \succ opState_{C2}) \vee ((\bar{a}_{C1}(t) \geq \bar{a}_{C2}(t)) \wedge (opState_{C1} = opState_{C2}))$,

while the management policy *current-availability-first* considers

$h_{C1}(t) \succ h_{C2}(t)$ iff $(\bar{a}_{C1}(t) > \bar{a}_{C2}(t)) \vee ((\bar{a}_{C1}(t) = \bar{a}_{C2}(t)) \wedge (opState_{C1} \succ opState_{C2}))$.

The QoS is directly dependent of the loading for many types of servers. Consequently, we use the tuple $<health, usState>_C$, to represent the QoS offered by a given component C. The following management policies may be used to compare the QoS of different components.

The *health-first* policy considers that:

$C1 \succ C2$ iff $((h_{C1}(t) \succ h_{C2}(t)) \vee ((h_{C1}(t) = h_{C2}(t)) \wedge (usState_{C1} \succ usState_{C2}))$,

while the *usState-first* policy considers that:

$C1 \succ C2$ iff $(usState_{C1} \succ usState_{C2}) \vee ((usState_{C1} = usState_{C2}) \wedge (a_{C1}(t) \succ a_{C2}(t)))$

6. CONCLUSIONS

In our proposal, we have introduced the definitions of three elements which facilitate automatic reconfiguration management: (i) new state change models for managed objects, (ii) the actual availability, and (iii) the required environment for a managed object, distinguishing the functional, operational, and existential environment of a managed object, as well as its management and test environments. Based on these proposals, we have defined several monitoring and server selection policies, commonly used to collect data and reconfigure distributed systems. Ongoing works use this approach to formalize reconfiguration activities in distributed systems.

6. REFERENCES

[1] Feit, S. 1995. SNMP: *A Guide to Network Management, McGraw-Hill Inc.*, 1995.

[2] ISO/IEC JTC 1/SC 21 N 6356, *State Management Function* (Final Text of DIS 10164-2)

[3] TINA-C Deliverable. *Information Modelling Concepts*. TB_EAC.001_1.2_94, Version 2.0, April 3, 1995.

[4] Dini, P., Bochmann, v. G. Specifying Lifecycles of Object Interactions for Reconfiguration Management in Distributed Systems. *The Second IEEE Systems Management Workshop*, Toronto, Ontario, Canada, June 19-21, 1996, pp. 82-91.

[5] Dini, P. Migrating Wireless Services: A Management Solution for Automatic Reconfiguration. *The International Conference on Telecommunications " Bringing East&West Through Communications"* ICT97, Melbourne, Australia, 2-4 April 1997.

[6] Dini, P., Bochmann, v. G., Boutaba, R. Performance Evaluation for Distributed System Components, *The Second IEEE Systems Management Workshop*, Toronto, Ontario, Canada, June 19-21, 1996, pp. 20-19.

[7] Dini, P., Bochmann, v.G., Koch, T., Kraemer, B. Agent Based Management of Distributed Systems with Variable Polling Frequency Policies. *IFIP/IEEE International Symposium on Integrated Network Management,* San Diego, CA, USA, May 12-16 1997.

[8] Dini, P., Das, A., Bochmann, v. G. Applying Parallel Algorithms for Managing Distributed Systems, *The 11th International Conference on Systems Engineering (ICSE'96), Special Session on Networks and Distributed Systems*, University of Nevada, Las Vegas, 9-11 July 1996.