Layered Dependability Modeling of an Air Traffic Control System

Olivia Das, C. Murray Woodside
Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada
email: odas@sce.carleton.ca, cmw@sce.carleton.ca

Abstract

Quality attributes, such as performance and dependability of a software-intensive system are constrained by its software architecture. The combined performance and dependability (called performability) effects of an architecture can be evaluated by constructing a performability model that considers the failure/repair behavior and performance attributes of its components, interactions among the components and the fault tolerant approaches adopted. This paper constructs and analyzes a Dependable Layered Queueing Network (Dependable-LQN) performability model for a large-scale Air Traffic Control system. It demonstrates the capability of the appraoch, for evaluating the performability of a large-scale software architecture.

1. Introduction

The Dependable Layered Queueing Network (Dependable-LQN) model is a performability model for fault-tolerant distributed applications with a layered software architecture and a separate architecture for failure detection and reconfiguration. It considers the failures (repairs) of its application and management components, management connections and the application's layered failure dependencies together with the application performance. It combines Fault-Tolerant Layered Queueing Networks (FTLQN) and the Model for Availability Management Architecture (MAMA) [1]. FTLQN in turn extends the Layered Queueing Network model [2] (a pure performance model) with dependability related components and attributes. This paper describes its application to an Air Traffic Control (ATC) system.

An ATC system [3, 4, 5] is a large-scale complex distributed system which demands high availability and high performance. Its software architecture plays a key role in achieving its high quality requirements; we consider the architecture described in [3]. This is an *en route* system which controls aircraft from soon after takeoff until shortly before landing. Its end users are the air traffic controllers. It has a layered software architecture with one subsystem depending on another for services and it uses a separate management architecture for automatic failure detection and recovery. It is an important case study for us because Dependable-LQN model perfectly fits the choice for modeling such a complicated and large system.

The goal of this paper is to demonstrate the use of the Dependable-LQN model on a substantial system with strong requirements for performance and dependability. It also describes the use and scalability of a tool [6] that takes the layered software description and the management component interactions as input and solves the model analytically to generate results.

2. The Dependable-LQN Model

2.1. First part: FTLQN Model

Figure 1 illustrates an example of an FTLQN model with an example of a layered console application. There are seven tasks (concurrent operating system processes, represented as rectangles), "Console", "Application Server", "Console Server", "Log Server", "Database-A", "Database-B" (backup of "Database-A") and "Data Server". Each task runs on a processor represented by an ellipse. The task "Application Server" creates a report which involves reading from the database, requesting some kind of application data from the "Data Server" and then logging the report to the "Log Server". Tasks have one or more entries which are service handlers embedded in them ("Data Server" has "Get Application Data" and "Get Console Data"). A *service request* (represented by a rounded rectangle) has a single target server if there is no server redundancy, or it may have a set of redundant target servers with a policy on how to use them.

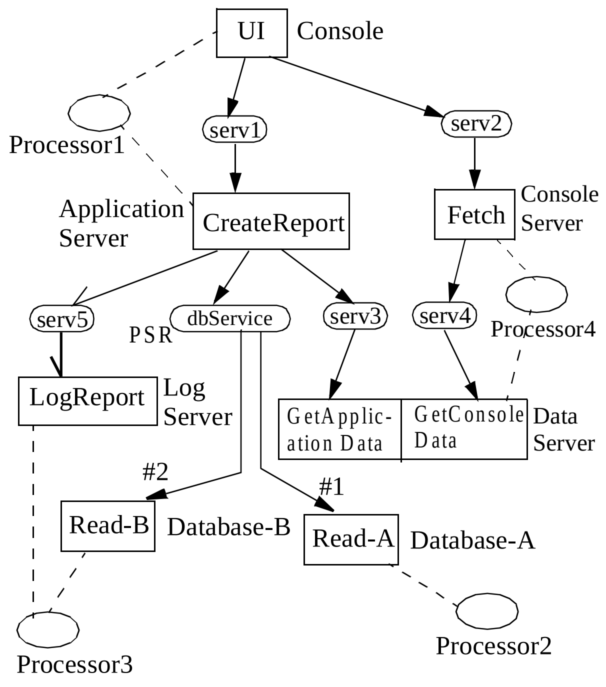


Figure 1. An FTLQN model

The different redundancy policies supported for a *service request* are:

- Primary-Standby Redundancy (PSR) with load sent only to the primary target
- Active Redundancy (AR) with load replicated and sent to all working targets
- Primary-Standby Active Redundancy (PSAR) with load replicated and sent to all the working targets where a designated primary does exist. This redundancy policy has been introduced in this paper. It is useful in cases where a client sends a service request to all the redundant servers (in Layer *i*) for close synchronization, however only the primary server of Layer *i* would send the service request to Layer *i*+1 in order to decrease the number of replicated requests to Layer *i*+1.
- Load-Balanced Redundancy (LBR) with load equally divided among all working targets

In Figure 1, “dbService” is a service requested by the entry “Create Report” for reading from the database. It has PSR policy where the priority of the target servers are labelled “#n” on the arcs going out from the service to the server(s). In this model, all the service requests are synchronous where the sender is blocked until the receiver replies. The model is restricted to being acyclic in order to avoid cycles of mutual waiting that may lead to deadlock.

Asynchronous service requests can also be accommodated where a client does not block after sending a request to the server. In Figure 1, “serv5” represents an asynchronous service request. In contrast to a synchronous service request where the failure of a client directly depends on the failure of the servers, an asynchronous service request does not add any failure dependencies. In order to add additional failure dependencies that cannot be represented by service-based dependencies, another abstraction called *depends* relationship (of types “AND” or “OR”) can exist between any two entries (illustrated in Section 3).

The performance parameters for the FTLQN model are the mean total demand for execution on the processor by each entry and the mean number of requests for each interaction. The availability related parameters for this model are the probabilities of being in failed state for each component (either a task or a processor) of the application. The performance measures are usually associated with the tasks that only originate requests (e.g. “Console” task), also called *reference tasks*.

The FTLQN model shows the policy for redundant servers but the decision about where to forward the request is made by the fault management sub-system (not visible in this model) based on its knowledge of the state of the application components. This resolution of requests gives different operational configurations of the application. The probabilities of the operational configurations now depend on the fault management architecture, management subsystem failures, and as well as on the application.

The FTLQN model can be solved to compute steady-state measures, e.g. mean throughput of the system in presence of failures. The general strategy of the analysis is to compute the performance for each reachable configuration that has different choices of alternative targets for requests and combine it with the probability of each configuration occurring, to find the measures. For example, in Figure 1 if all the tasks are operational, then the configuration is the system as shown, but with Database-B, and its service requests (labelled #2) removed as they are not used. This configuration is a pure Layered Queueing Network

(LQN) performance model [2]. Each LQN model can be solved for different performance measures by the LQNS tool[2], based on extended queueing approximations. This strategy is similar to the Dynamic Queueing Network approach given in [7, 8] for queueing network models.

The next section describes the management components, their connections and how they are related to the application components.

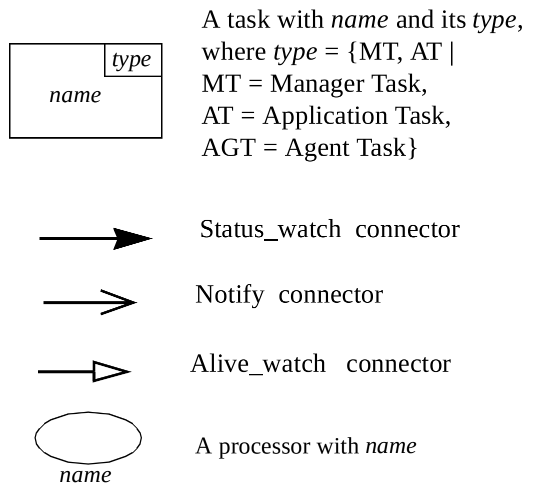
2.2. Second part: MAMA Model

MAMA model has four classes of components: application tasks, agent tasks, manager tasks and the processors hosting them. They are connected using three different classes of connector:

- Alive-watch connector: This connector is used between two components in cases where the destination component would like to be able to detect whether the source component is alive or not. This may be achieved by periodic polls or "i-am-alive" messages. Usually, the source of this connector are the manageable application components.
- Status-watch connector: In cases where a destination component would like to know about the liveness of the source component and also wants to collect failure data about other components in the system that has been gathered by the source component, this connector is used. An example would be a connector from an agent task to a manager task.
- Notify connector: This connector is used for cases where the source component would like to send or forward reconfiguration commands to its sub-ordinate destination component (for example, a manager sending commands to an agent or an agent forwarding a command to an application task) or conveying management information to its peer (for example, a manager sending information about the domain it manages to another manager).

Cycles may occur in a MAMA model; we assume that the flow of information is managed in a way so as not to cycle or ping-pong. It is also assumed that if a task watches a remote task, then it also watches the processor executing the remote task in order to differentiate between a task failure and a processor failure.

Figure 2 shows the graphical notation used in this work for MAMA components and connectors.



type
name
A task with *name* and its *type*, where *type* = {MT, AT | MT = Manager Task, AT = Application Task, AGT = Agent Task}
Status_watch connector
Notify connector
Alive_watch connector
A processor with *name*
name

Figure 2 Notation used here for the MAMA model

For this model, the failure probabilities can be provided for all management components and the connectors between them.

3. Dependable-LQN Model of an ATC En Route System

An airspace controlled by a single ATC facility is administratively divided into sectors. For example, US airspace is serviced by 20 facilities each with a maximum of 118 sectors per facility. Each facility receives aircraft surveillance and weather data from radars and communicates with other external subsystems such as other en route facilities. Inside each facility, air traffic services are divided among four subsystems: Surveillance Processing Service (that receives radar data and correlates it with individual aircraft tracks) is provided by Radar subsystem directly connected to radars, Flight Plan processing and Conflict Alert services is provided by Central subsystem, Display service (which displays aircraft position information obtained from radars and allows inputs from air traffic controllers to modify flight plan data or change display format) is hosted by the Console subsystem, Monitoring service (which provides the monitoring and control service for other ATC services ensuring their availability policies) is hosted by the Monitor and Control subsystem. There are up to four consoles allocated for handling each sector. Fault-tolerance is achieved by software server groups. For example, there are up to four Display Management primary/standby active redundant servers per sector, three primary/standby active redundant Surveillance Processing servers, two primary/standby Flight Plan Management servers.

All the three redundant Surveillance Processing servers receive the raw radar data from the radars in parallel, however, only the primary would send the processed radar data to the Display Management servers running in the consoles. A PSAR redundancy policy have to be used to model this case.

Figure 3 shows some parts of a Dependable-LQN model based loosely on the description in [3]. Each bubble represents a process group with replication; the service nodes and redundant servers are not shown. The communications with replicas is made transparent by a process-to-process session manager (P2PSM) in each host (not shown here).

Some failure dependencies are implicit in the server request-reply dependencies. Others can be made explicit with a *depends* relationship, for instance that "display radar data" depends on "process radar data", even though the communications is asynchronous. This would be an *OR depends* relationship on the three radar processing replicas, since any one is sufficient.

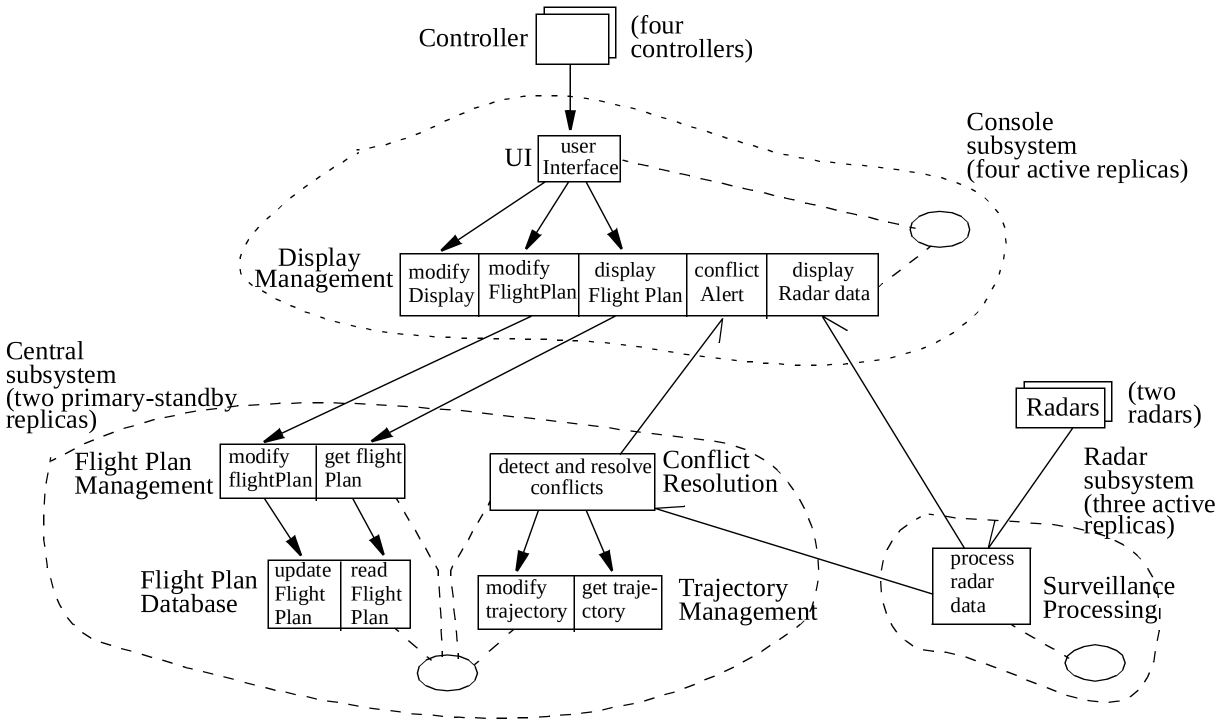


Figure 3 A Dependable-LQN model for an ATC en route system. Redundant server groups are not shown here.

The fault management architecture depends on a *group availability management* server (gSAM) on each processor which monitors all the software servers in its own processor and also monitors the other processors in its software server group. In MAMA terms, the gSAM servers maintain an *alive-watch*. This is supported by three redundant name servers which maintain a list of primary host processors for each server group. When a failure of a software server occurs in its processor, a gSAM server notifies other gSAM servers in its own group. Similarly, the failure of a processor is detected by the other gSAM servers in a group. Whenever a failure is detected, the gSAM server notifies the name servers which in turn notify the relevant P2PSM's so that they can retarget their service requests.

Figure 4 shows a portion of the MAMA model for Figure 3, including one replica of the Monitor and Control subsystem. The redundant servers and the interactions among the gSAM servers in a group are not shown here.

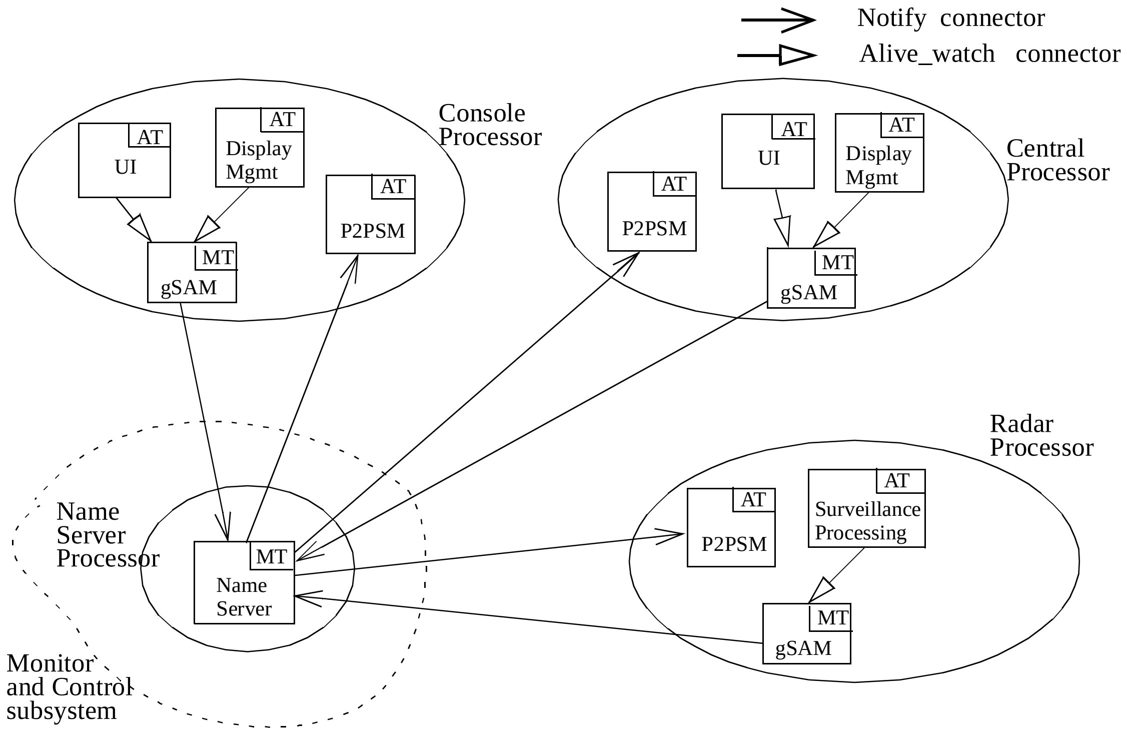


Figure 4. Portion of MAMA model for Figure 3. Redundant server groups and interactions among the gSAM servers in a group are not shown here.

A model for a sector has around 13 processors and 41 tasks (including the P2PSM and gSAM tasks). This is comparable to the other models that have been solved with the Dependable-LQNS tool [6]. The analysis results will be described at the workshop.

4. Conclusion

This paper described a Dependable-LQN model for evaluating performability of a large-scale Air Traffic Control system. The value of the work lies in showing how the Dependable-LQN model can be used to evaluate whether an architecture meets its performability goals or not and also to demonstrate the scalability of the model solving tool. Our future research includes considering the detection and recovery delays in addition to the management architectural limitations and its failures into the analysis.

5. References

[1] O. Das and C. M. Woodside, "Modeling the coverage and effectiveness of fault-management architectures in layered distributed systems", IEEE International Conference on Dependable Systems and Networks (DSN'2002), June 2002, pp. 745-754.

[2] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, and M. Woodside, "Performance Analysis of Distributed Server Systems", in the Sixth International Conference on Software Quality (6ICSQ), Ottawa, Ontario, 1996, pp. 15-26.

[3] F. Cristian, B. Dancey and J. Dehn, "Fault-Tolerance in Air Traffic Control Systems", ACM Transactions on Computer Systems, 14(3), August 1996, pp.265-286.

[4] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice, Addison-Wesley, 1998.

[5] A. S. Debelack, J. D. Dehn, L. L. Muchinsky and D. M. Smith, "Next generation air traffic control automation", IBM Systems Journal, 34(1), 1995, pp. 63-77.

[6] O. Das and C. M. Woodside, "Dependable LQNS: A

performability modeling tool for layered systems", submitted for IEEE International Conference on Dependable Systems and Networks (DSN'2003).

[7] B. R. Haverkort, I. G. Niemegeers and P. Veldhuyzen van Zanten, "DYQNTOOL: A performability modelling tool based on the Dynamic Queueing Network concept", in Proc. of the 5th Int. Conf. on Computer Perf. Eval.: Modelling Techniques and Tools, G. Balbo, G. Serazzi, editors, North-Holland, 1992, pp. 181-195.

[8] B. R. Haverkort, "Performability modelling using DYQNTOOL+", International Journal of Reliability, Quality and Safety Engineering., 1995, pp. 383-404.