View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Infoscience - École polytechnique fédérale de Lausanne

Modular Communication Infrastructure Design with Quality of Service

Pawel Wojciechowski and Péter Urbán

Distributed Systems Laboratory
School of Computer and Communication Sciences
Swiss Federal Institute of Technology (EPFL)
Lausanne, Switzerland

Extended Abstract

January, 2002

Introduction

We are interested in the design of programming abstractions and infrastructure (or middleware) for building global network applications. We envisage that these applications will be executed on different types of machines, including mobile computers which may have only intermittent access to the network, and palmtops which offer limited user interfaces. The users will be moving frequently and communicating with other users or systems at different locations using different interface devices and communication standards. For instance, a portable device might use the IEEE 802.11 wireless LAN while in the university campus, switching to GSM after leaving the campus, and again to IEEE 802.11 or Bluetooth at home. Some parts of the application(s) may also migrate between computers during their normal execution, e.g. we may want to migrate a running application from / to a laptop computer in order to continue working in a disconnected mode, or a new user interface and location-dependent data could be downloaded dynamically to a mobile device when a physical location changes.

In order to support communication between distributed (possibly migrating) parts of a distributed application, the communication infrastructure is required. However, different applications may have different requirements as for tolerance to failures, response time, security and local memory usage for buffering messages, etc. Therefore the infrastructure should be application oriented and executed as (a layer of) middleware customized for a given application. In the Crystall project [1], we are interested in implementations of the communication infrastructure using a modular approach, which is based on implementing properties required by a user application as separate protocols, and then combining selected protocols of the infrastructure using a software framework [2]. This approach helps to clarify the dependencies among different properties, and makes it possible to construct systems that are customized to the specific needs of the application or underlying network environment. The proposal described below is an off-spring of the Crystall project that is related to the *quality of service (QoS)* aspects of the modular design of network protocols.

Timeliness and Fault Tolerance Guarantees

Many distributed applications have some timing constraints and require from the underlying communication infrastructure (or middleware) the QoS properties which are related to *timeliness*. For example, urgent messages which cannot be delivered to a mobile device in a short time are useless and should be discarded after some time. On the other hand events that can not be distributed to subscribers now (e.g. since the subscribers are not connected or have failed) should be buffered and sent again later when the subscribers will recover or reconnect (possibly at new locations). Another QoS properties are related to *fault tolerance*. One way of making an application tolerant to partial failures (of machines and the network) is to replicate its services on different machines using *group communication* algorithms. There are many different QoS properties which an application using the group communication middleware may require, such as the order of messages (e.g. Atomic Broadcast [3]) and the way to deal with the network partitions.

In the modular design of the communication subsystem, we assume that each protocol module of the communication infrastructure may provide some QoS properties for an upper-level module(s), and require certain guarantees from the lower-level module(s) of the protocol graph. For instance the *failure detector*, which is a basic building module of many distributed algorithms provides some information on which distributed processes have crashed, usually a list of processes that are *suspected* as crashed. This list in not always up-to-date or correct and may change over time: a failure detector may take a long time to start suspecting a process that has crashed, and it may erroneously suspect a process that has not crashed (this can happen due to message losses and delays in network communication or message processing). Chen et al. [4] study the quality of service (QoS) of failure detectors. By QoS, they mean a specification that quantifies two properties of failure detectors: (1) how fast the failure detector detects actual failures, and (2) how well it avoids false detections. In order to guarantee the QoS properties, any practical implementation of a failure detector in a distributed system must depend on assumptions about timeliness in the underlying network, e.g. what is the probabilistic transmission time.

The quantitative values of timeliness can only be based on some assumptions about the execution environment and the network. Similarly, the QoS properties related to fault-tolerance strongly depend on the failure model supported by the system environment. For instance, we may assume that if a machine crashes then all processes on this machine are lost, or they can be recovered upon site recovery; in the latter case the undelivered messages could be delivered to the processes. Unfortunately, global computation assumes the use of global network infrastructure -- in this case it is not possible to predict the timeliness values easily, nor make assumptions about the failure model. For example, in a wide-area network messages can be lost, communication delays are unpredictable, crash of a remote machine and message loss can be indistinguishable (e.g. due to a simple fact that some machines are hidden behind firewalls and the whole network is divided into a large number of administrative domains under no single management). The wide-area network is close to the specification of asynchronous systems, in which there is no timing assumption whatever. On the other hand, local area networks are mostly reliable, with a single management, and message communication delays are usually predictable.

Nevertheless, some knowledge about the application could help to estimate the assumptions about the underlying environment that are necessary to validate if the QoS properties requested by a given module can by satisfied. This would help to check if the configuration of modules used for a given application is correct. For example, servers replicated in the cluster of servers will always communicate via a local-area network, where delays can be predicted. As for wide-area networks, we can assume that message delays and message losses follow some probability distribution; this distribution depends, e.g. on the distance and real time in different geographical locations, etc.

The communication infrastructure built from the modules should have easily-predictable behaviour. It should be possible to verify if the assumptions about the QoS match the requirements of the application, and if they are still valid if some of the modules have been replaced or modified. Unfortunately, the traditional programming abstractions for implementing network protocols do not allow for specifying the guarantees as for reliability and the quality of service. Moreover, the distributed algorithms that are used for implementing the communication infrastructures are hard to understand, analyse and verify in a straightforward way. Thus, the problems are: (1) how to describe the (requested and provided) QoS properties of the modules, and (2) how to verify correctness of the composition of individual QoS properties provided by the modules.

Static Validation of QoS Requirements

We claim that the network protocol "architect" whose task would be to build the communication infrastructure with a quality of service that is required by an application should be able to compose the middleware layer without deep knowledge of the protocol implementation details. Therefore, it should be possible to express the quality of service guarantees of a given protocol in the declaration of the module that implements this protocol. The compiler which takes the module composition as an input should be able to verify if the infrastructure is correct in terms of the expected quality of service of the whole composition. In practice, this would require to verify if any subset of modules used to build the subsystem offer quality of service that is required by other modules that depend on them. The parameterised QoS guarantees of the service implemented by the modules should be specified in the module QoS declaration (and we trust the module implementators that they are correct). At the compilation or system start up time, when the modules are composed with other modules, the actual guarantees would be computed, given a range of values (or relations) that form assumptions about the timelines guaranties or fault-tolerance properties of the lower-level modules of the subsystem architecture. Below we consider modules with the QoS that is related only to quantitative timeliness properties.

The project is to design the extension of the module composition framework which allows to specify and (automatically) verify some properties of network protocols related to time. It should be possible to analyse statically at compile time (i.e. without doing any measurements) if a distributed algorithm built from a collection of protocol modules (possibly communicating with many remote sites) actually satisfies timeliness properties requested by the application, given certain (quantitative) assumptions about the network communication. It should be possible to express these properties and the input assumptions in a programming language, and verify the partial correctness at compilation time.

As a proof of concept, we could implement such features as a mild extension to a small concurrent programming language, such as Pict or Nomadic Pict [5] or other language based on process calculi. The advantage of choosing such a language is that the intermediate code generated by a compiler is usually small, and corresponds to a calculus with a clearly defined operational semantics; therefore it should be easy to perform static analysis of the code. For example, in Pict all computation is expressed by terms of communication on logical channels. The compiler performs static analysis of the intermediate code in order to eliminate unnecessary communication and fragments of code which are never executed. This analysis can be seen as a global search in the space of possible execution traces (of course, we mean only those traces that do not require any interaction with the external environment); this analysis produces a lot of useful statistics about the program's possible execution at runtime and it seems plausible to be able to compute also some of the quality of services guarantees (given the specified input values).

Nomadic Pict extends Pict with constructs for distributed programming that include mobile agents and primitives for network communication. The primitives for network and agent communication are translated into calls to standard network protocols (TCP / IP), and the local communication on logical channels which is similar to Pict. In Nomadic Pict, we usually compile the whole distributed program (which includes the definition of servers and clients) as one piece of code (though it is not, of course, necessary to do so) and spawn the servers and clients on remote machines using migration. Therefore, we can imagine that the compiler would be able to analyse the whole distributed program and "compute" any time dependencies that would be useful to estimate the timeliness properties of the algorithm. We suppose that such an approach could be also applied for some more traditional languages.

Related Work

[2] describes a framework that, similarly to our proposal, aims at providing QoS guarantees in distributed systems, and suggests to do so by implementing systems using composition of microprotocols. The difference is that the authors focus on the composition of microprotocols, whereas we would like to focus on the analysis of the guarantees (mostly timeliness guarantees) that result from the composition of microprotocols with known properties. The key concepts of our proposal are: (1) the programming language abstractions for the QoS, and (2) the design of the compiler's module which can perform static analysis of distributed programs with the QoS properties expressed using these abstractions. Whereas [2] focuses on distributed object systems, we feel that the object-oriented aspect of systems design is rather orthogonal to their QoS characteristics. Both [1] and our proposal consider QoS related to fault tolerance.

References

[1] Semantics of Protocol Modules Composition and Interaction. Pawel T. Wojciechowski, Sergio Mena, André Schiper. To appear in the Proceedings of Coordination'02 (The 5th International Conference on Coordination Models and Languages), 2002.

[2] Providing QoS Customization in Distributed Object Systems, Jun He, Matti A. Hiltunen, Mohan Rajagopalan, and Richard D. Schlichting. Proceedings of Middleware 2001 (The International Conference on Distributed Systems Platforms), Springer LNCS 2218, 2001.

[3] A Modular Approach to Fault-Tolerant Broadcasts and Related Problems. Vassos Hadzilacos and Sam Toueg. Technical Report TR94-1425, Department of Computer Science, Cornell University, 1994.

[4] On the Quality of Service of Failure Detectors. Wei Chen and Sam Toueg and

Marcos Kawazoe Aguilera. In the Proceedings of DSN 2000 (The International Conference on Dependable Systems and Networks), 2000.

[5] Nomadic Pict: Language and Infrastructure Design for Mobile Agents. Pawel T. Wojciechowski and Peter Sewell. This appeared in the Proceedings of ASA/MA'99 (First International Symposium on Agent Systems and Applications/Third International Symposium on Mobile Agents), October 1999.