Performance Analysis of the Simplicity Project: a Layered Queueing Network Approach

Francesco Lo Presti
Dipartimento di Informatica, Università dell'Aquila
Via Vetoio, 67010 Coppito (AQ), Italy
lopresti@di.univaq.it

***Abstract*— As technology develops, a broader and broader range of heterogenous ICT devices and network-based services are made available to users. As for today, this often results in an enormous burden of complexity on users, service providers and service operators.**

To overcome these limitations, the IST Simplicity project aims to ease the user interaction with devices and the use of services and functionalities. The Simplicity architecture revolves around a system of brokers which provide a layer able to decouple user needs and user devices, as well as service deployment and fruition, from the underlying networking and service support technologies.

In this paper, we address the performance of the Simplicity system. By using the Layered Queueing Network formalism, we propose and study a model of the Symplicity architecture. The analysis allows us to explore in a simple and efficient way the system performance under different setting and to explore several alternatives.

Key words: Ubiquitous Services, Service Personalization, Performance Analysis, LQN model.

I. Introduction

As technology develops, a broader and broader range of ICT devices and network-based services are made available to users. As for today, to use these services, the user has to access them through heterogenous technologies and protocols, must use different devices, must properly configure them, must be authenticated and charged in different way by the different services. The result is an enormous burden of complexity for users, service providers and network operators, complexity which can become a serious obstacle to the development and deployment of new services.

To overcome these limitations, the IST-Simplicity project [1], [2], [4] aims to ease the user interaction with devices and the use of services and functionalities. The project goal is to design and deploy a brokerage level able to decouple user needs and user devices, as well as service deployment and fruition, from the underlying networking and service support technologies. Within Simplicity, users are characterized by a personalized profile to be used for different services/transactions, eventually based on different classes of terminal. Such profile allows an automatic, transparent personalization and configuration of terminals/devices, and provides a simple and uniform way to be recognized, authenticated, located and charged.

In Simplicity the user profile is stored in a so called Simplicity Device (SD). Users personalize terminals and services by the simple act of plugging the SD in the chosen terminal (see Figure 1). The Simplicity system encompasses the Simplicity Device and a Brokerage Framework. The Brokerage Framework will use policy-based technologies (e.g., policies for mobility support, QoS, security, SW downloads) to orchestrate and adapt network capabilities, taking into account user preferences and terminal characteristics.

The Brokerage Framework comprises a Terminal Broker module, which is primarily used to allow the interaction of the SD with both the terminal and the network, and a Network Broker module. Since the SD has the goal to allow a uniform and personalized user view of services, there must be a way to describe, and advertise such services, to allow the user to browse and select them. Subsequently, there is the need to coordinate services and share/allocate the available resources. The Network Broker is responsible to perform the aforementioned tasks, by providing a platform for service deployment, advertisement, personalization, etc.

The architecture of the Terminal and Network Broker is based on the concept of subsystems. Subsystem means, generically, a software system that is able to perform some task, or to collect some input from external sources. Several subsystems connect to a specific subsystem called Mediator, to which they send asynchronous messages called events. The Mediator then dispatches these events, according to a set of policies. This architecture is flexible enough to support all the functionalities required in the system. Moreover, this approach enables flexible addition and deletion of subsystems, without affecting the rest of the system.

Performance evaluation of the Simplicity Architecture is a key ingredient to assess the effectiveness and efficiency of the proposed high level architecture. By combining different performance evaluation methodologies, namely, measurements, analytical modeling and simulations, we plan to verify the conformance of the architecture to users and system performance requirements; at the same time, we can identify the presence of potential bottlenecks to be addressed.

As a first step, in this paper, we propose and analyze an analytical model of the Terminal Broker and User Terminal. The formalism used for the model is the Layered Queueing Network (LQN) model [3], [5], an extension of the well-known Queueing Network model, developed especially for modelling concurrent and/or distributed software systems. The model allowed us to the terminal performance under different setting. In particular: the impact of CPU utilization on Simplicity operation response time; the relative benefit of giving CPU priority to the Mediator or to all Simplicity subsystems; impact of network loss on performance. Starting from this model, moreover, we have also explored the impact of different implementation alternatives on Simplicity performance. In particular, with reference to the mediator, we studied: the impact of events rate on the mediator performance; the trade-off in using different level of priority to speed-up dispatching of sensible events; the impact of a multithreaded implementation. Given the space limitation, we will provide the entire model, but only part of the performance evaluation results.

The rest of the paper is organized as follows. Section II

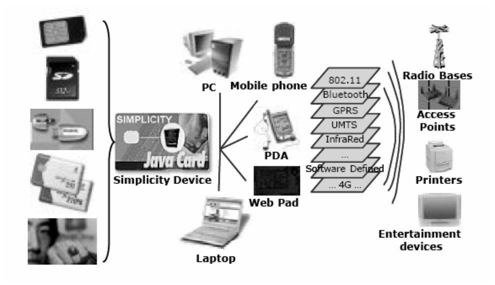

Fig. 1. The Simplicity Concept.

introduces the LQN formalism. Section III describes the Simplicity architecture. Details of the Terminal Broker are given in Section IV. Section V presents the LQN model for the Terminal Broker. Performance results are presents in Section VI. Section VII concludes the paper.

II. LQN Modelling

LQN was developed as an extension of the well-known Queueing Network (QN) model [3], [5]. With respect to traditional QN, in LQN a server, to which customer requests arrives and queue for service, may become a client to other servers from which it requires nested services while serving its own clients. An LQN model is represented as an acyclic graph whose nodes (named also tasks) are software entities and hardware devices, and whose arcs denote service requests. The nodes with outgoing and no incoming arcs play the role of pure clients. The intermediate nodes with incoming and outgoing arcs play both the role of client and of server, and usually represent software components.

A software or hardware server node can be either a single-server or a multi-server. A LQN task may offer more than one kind of service. An entry has its own execution time and demands for other services (given as model parameters). Each server has an implicit message queue, where the incoming requests are waiting their turn to be served. Servers with more then one entry still have a single input queue, where requests for different entries wait together. The default scheduling policy of the queue is FIFO, but other policies are also supported.

III. System Architecture

In this section we briefly review the Simplicity Architecture (details can be found in [1]).

A. Simplicity Device

The key role of the Simplicity Device is to store in a secure and safe way users profiles, preferences and policies to allow dynamic and automatic discovery and registration of terminal and network capabilities. The SD requires memory for storage of information. Such memory can be located internally, on the SD, or externally, on the devices in the environment. To this purpose, the SD can store a set of pointers to network locations

B. Terminal Broker

The user terminal/device is supplied with a so-called Terminal Broker (TB). The TB is the entity that interfaces the user to the network. A first function of the TB is to provide the user with a mean to read/write/modify the personalization information stored in the SD. A second function of the TB is to enable the SD to perform discovery of terminal capability and service adaptation to the ambient environment. The TB can configure applications and network settings and can dynamically download plug-ins and applications. Last, the TB is the entity that allows user preferences and policies, stored in the SD, to drive service adaptation to networking technologies and capabilities.

C. Network Broker

The Network Broker (NB) is responsible to provide: i) a platform for service deployment, advertisement, personalization, ii) service adaptation capabilities to the considered context (location, time, etc.); iii) an orchestration of events and the handling of simultaneous access of several users to the same resources, services, and locations. The NB provides a network based platform. The platform takes into account user profiles, terminal capabilities as well as the network side and handles the user request in an optimized and personalized way. In order to achieve the above objectives, the NB needs to communicate with the TB in order to retrieve information about the user profile, his/her context and the device capabilities. As far as the network side is concerned, the NB uses policy-based technologies (e.g., policies for mobility support, QoS, security, software downloads). These modules perform the orchestration of the available resources, the adaptation of network capabilities and the management of the different access technologies and networking alternatives. The NB also interacts with NBs of adjacent networks in order to optimize end-to-end service across network domains.

IV. Architecture of the User Terminal System

In this section we describe in more detail the User Terminal System Architecture. Figure 2 shows an architectural overview of the User Terminal System. Terminal broker functionalities are encapsulated in separate subsystems. Interaction between these subsystems is enabled by a Mediator which relays messages between them. This approach enables flexible addition and deletion of subsystems, without affecting the rest of the system.

Adaptor subsystems allow the introduction of legacy entities. These subsystems represent an interface to the legacy service. All information flows between Simplicity and the Legacy Entity pass via the adaptor subsystem, which acts like an ordinary Simplicity subsystem.

The Mediator is a Broker-internal message-relaying component. Messages are sent by one subsystem and received by one or more subsystems. Senders and receivers do not have to have any knowledge about each other. Rather, the Mediator uses a set of rules to determine which messages are to be forwarded to which receivers. The forwarding rule appropriate for a particular message is selected depending on the type of the message and the type of the sender. Every Broker contains a single Mediator which is responsible for all messages relaying within the Broker. The Mediator itself is not proactive; it simply reacts to Messages that have been sent by the various subsystems.

External communication is not relayed through the Mediator. If a subsystem needs to communicate with a subsystem on another Broker (e.g. in order to provide a service), it communicates through the Simplicity Broker Communication subsystem.

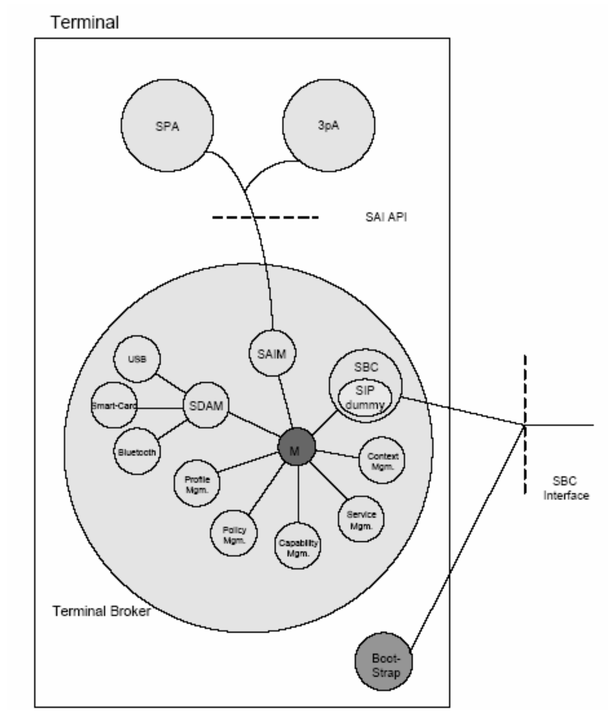

Fig. 2. Architecture Overview of the User Terminal System.

A. Subsystems

1) Simplicity Device Access Manager (SD-AM): The Simplicity Device Access Manager (SD-AM) is the subsystem in charge of access to the Simplicity Device. As different kinds of cards may be used, this subsystem must present a unified interface to the Mediator, while its internal structure may vary, depending on the kinds of cards in use. The SD-AM offers an interface able to support the various functionalities required in the SD. In particular, it must offer an API to store and retrieve user and terminal profiles, and executable code that is used to start up the Simplicity system.

2) Simplicity Application Interface Manager (SAIM) and the Simplicity Personal Assistant (SPA): The Simplicity Application Interface Manager (SAIM) subsystem allows a 3rd party application to run on top of the Terminal Broker, and to use the functionalities of Simplicity, but without being aware of the detailed mechanisms of the Simplicity brokerage framework. This subsystem offers 3rd party applications a standardized interface, the Simplicity Application Interface (SAI). SAI interface is also used by the Simplicity Personal Assistant, a software entity that acts proactively and assists users in their tasks. The SPA is involved in user authentication and users preference management.

3) Management Subsystems: A number of Management subsystems are attached to the Mediator. These include the Policy Management Subsystem, which is responsible for collecting policies and executing them, using a policy-engine; the Capability Management Subsystem, which deals with terminal capabilities (hardware, software, applications); the Profile Management Subsystem, which is responsible for collecting requests to download or upload profile information from other subsystems.

4) Simplicity Broker Communication (SBC): The Simplicity Broker Communication (SBC) Subsystem provides transparent access to a remote broker, while hiding the details about the specific transport protocol used to establish and manage the connection between brokers. A broker discovery procedure is first performed, and then, when the SBC has discovered available network brokers on the network, interbroker communications are established. The SIP controller module is responsible for discovering a Network Broker, and establishing a communication with it. A preliminary connection is required to perform any event exchange with the remote broker, and, in this context, SIP protocol is employed as a signalling protocol.

V. Terminal System Model

In this section we derive a LQN model for studying the performance of the Simplicity Terminal broker. Our goal is to provide a simple yet flexible approach to study and explore the system performance under different alternatives.

We consider a Terminal hosting the SD. We assume a terminal comprises different hardware devices: a CPU, a Disk, a Network connection. We assume the terminal runs different types of software. First of all, the terminal runs the Simplicity terminal broker software which allows the user to access Simplicity services. Moreover, the terminal also runs some native applications and a set of other terminal related tasks which compete for terminal resources with the Simplicity software. Finally there will be also some (third-party) applications which will be invoked by the user by using the Simplicity system. We assume the user periodically runs some native applications and/or access Simplicity services through which he accesses some applications, e.g., a browser, a multimedia downloading, etc. To access Simplicity Services, we assume the user goes through a sequence of steps: getting a list of services, calling and eventually subscribing to a service, personalizing, accessing and using the service. After such a sequence of operation the user stays idle for a given period, after which repeats the operations above.

Given the user behavior just described, and taking into account the operation sequence diagrams (which we cannot show in this paper for space limitation), we can now derive the LQN model for the terminal broker. The complete model, we displayed in Figure 3. For sake of space and given the similarities of the different operations, we will not illustrate the complete derivation of the LQN model, but we will focus on the analysis of the most representative case, the “get list of service” function.

In Figure 3, we can distinguish the circles denoting the hardware devices: CPU and Disk of the terminal; Disk_SD representing the storage SD; the Network and the CPU_R and Disk_R of the network broker. In the top row we can distinguish the reference tasks, i.e., those tasks which only generate requests: the Other_Task and Native which represents the other applications/services running on the terminal and the User which represent the Simplicity user. The User accesses the Simplicity services via the SPA invoking the different entries which correspond to the different types of activities involving the user.

We now provide the derivation of the LQN submodel modelling to the GetList function. The GetList operation is invoked by the user to get the list of available service from the Network

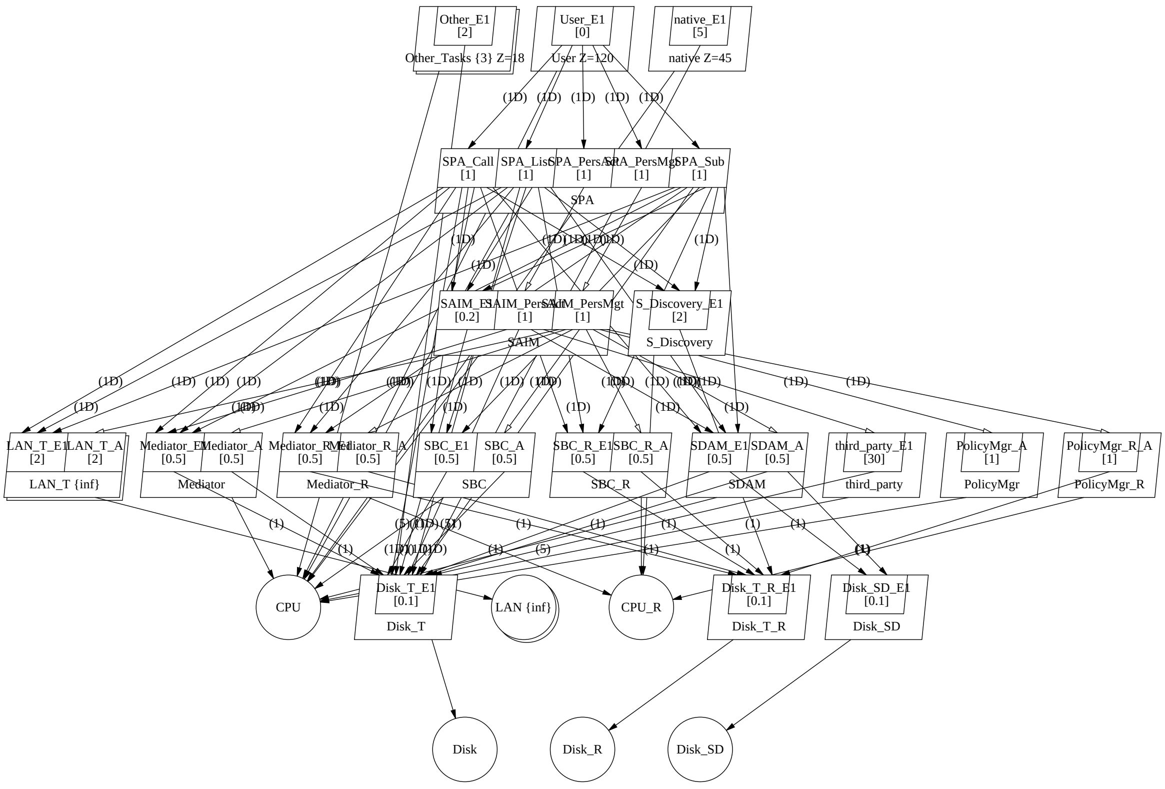

Fig. 3. LQN Model of the Terminal Broker.

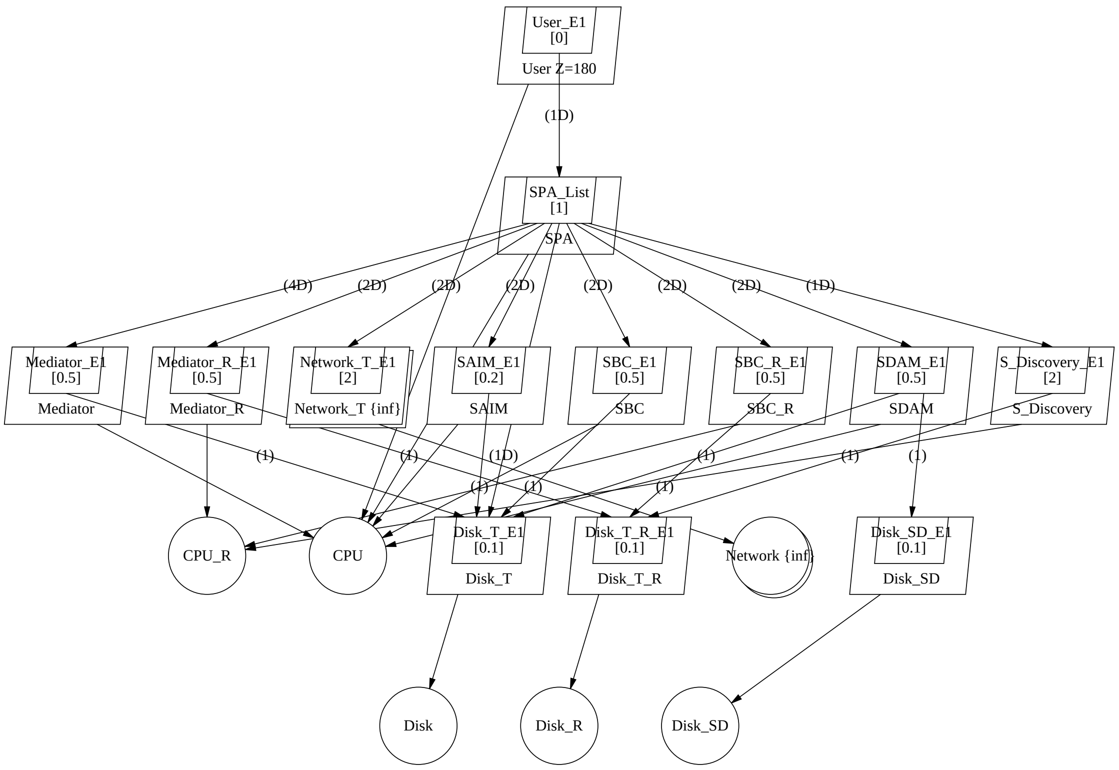

Fig. 4. LQN submodel for the get_list function.

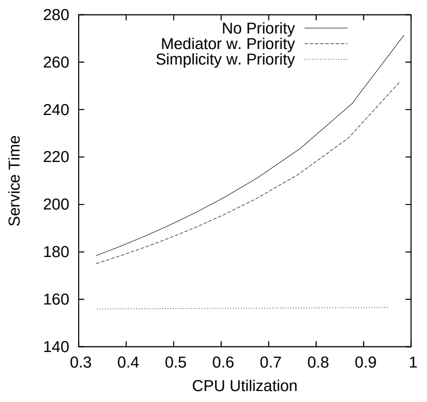

Fig. 5. Service Time (measured in ms) vs CPU Utilization.

Broker. The user sends a message to the SPA which forward the message to the SAIM which relays it to the Mediator. The Mediator pushes the message to the SBC which sends it to the Network Broker via the network. The message is then received by the remote SBC which will forward it to the Network Broker mediator and finally to the subsystem handling GetList requests (actually the Network Broker operations are more complex and involves more subsystems. For sake of simplicity, here we assume a single subsystem does it all). The answer message with the service list travels back all the way to the SPA which then recover the user profile information from the SD (via the SDAM) and presents the list to the user.

From the modeling point of view it is important to note that the interactions among all subsystems is all via asynchronous messages which represents a non-blocking type of interaction. Nevertheless, we cannot not use the "asynchronous" mechanism of the LQN framework which is a pure "send-no-reply" type of interaction. Here we resort to a simple approximation, which boils down to model the chain of non blocking calls with a sequence of synchronous call made by the SPA task to all the tasks in the chain. The resulting LQN submodel is drawn in Figure 4, where we see the SPA_List entry invoking the SAIM, the SBCs, the Mediators and the Network.

We stress that this model, despite the above approximation well represents the actual system behavior as long as - as is in our case - the service time from the point of the SPA and User is of interest.

VI. Performance Analysis

In the previous section we described the terminal broker model. In this section we use this model to study the terminal performance under different setting. Given space limitation, here we will restrict our study to: 1. the impact of CPU utilization on Simplicity operation response time; 2. the relative benefit of giving CPU priority to the Mediator or to all Simplicity subsystems.

As performance index, we will consider the Simplicity service time, measured as the average time from the instant the user issues the "GetList" command to the instant the chosen Application starts. This is obtained directly from the service time of the User_E1 entry. We consider the following three different scenarios: 1. no priority. 2. Mediatior priority. The Mediator has higher priority than all other task over the terminal CPU; 3. Simplicity priority. All Simplicity tasks have priority over other terminal tasks. In the analysis, the CPU utilization has been varied by varying the think time of the Other_Task tasks. In all the following examples, the Simplicity software accounts for only 5% of the CPU utilization.

The results are reported in Figure 5. As expected, without priority mechanism the CPU utilization has significant impact on Service Time. This suggests that the load induced to the CPU by other application can severely impact Simplicity performance. The situation improves by allowing Simplicity tasks have priority. In this case, the service time does not depend on the CPU utilization. In case we allow only the mediator to run with higher priority, we see that the improvement over the non-priority case is only marginal. This analysis suggests that wherever possible, Simplicity should run with priority over other terminal tasks.

We remark that, nevertheless, the results below are to be considered preliminary since we do not have yet workload data to parameterize the model. This data will be available in the near future as the system will be implemented.

VII. Conclusions

In this paper we have addressed the performance of the IST Simplicity project. Focussing on the User Terminal, we have provided an LQN model of the Simplicy architecture and presented preliminary results.

We are further developing the model and consider several architectural and implementation alternatives. We plan to complement this work by simulation and measurements which will provide the means to parameterize and validate this performance study.

Acknowledgments

This work has been partially founded by the EU in the framework of the IST project Simplicity.

References

[1] N. Blefari Melazzi and al., "The Simplicity Project: easing the burden of using complex and heterogeneous ICT devices and services. Part I: Overall Architecture", *in Proc. of IST Mobile and Wireless Summit 2004*, Lyon, France, June 2004.

[2] N. Blefari Melazzi and al., "The Simplicity Porject: easing the burden of using complex and heterogeneous ICT devices and services. Part II: State of Art of Related Technologies", *Proc. of IST Mobile and Wireless Summit 2004*, Lyon, France, June 2004.

[3] G. Franks, A. Hubbard, S. Majumdar, D. Petriu, J. Rolia, C.M. Woodside, "A toolset for Performance Engineering and Software Design of Client-Server Systems", *in Performance Evaluation*, Vol. 24, pp. 117-135, November 1995.

[4] R. Seidl and al., "The Simplicity Project: Personalized and Simplified communications spaces for mobile users", *Proc. of IST Mobile and Wireless Summit 2004*, Lyon, France, June 2004.

[5] C.M. Woodside, J.E. Nelson, D.C. Petriu and S.Majumdar, "The Stochastic Rendevous Network Model for Performance of Synchronous Client-Server-like Distributed Software", *in IEEE Transaction on Computers*, Vol. 44, pp. 20-34, January 1995.