Advances in Designing QoS Architectures – the 2nd Generation Ahead

Jan de Meer

GMD-FOKUS, Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany,
email: jdm@fokus.gmd.de

Armin Eberlein

University of Calgary, Department of Electrical & Computer Engineering,
2500 University Drive NW, Calgary, Alberta, T2N 1N4, Canada,
email: eberlein@enel.ucalgary.ca

Abstract.

The 1st generation of QoS architectures was communication- and network-oriented. The forthcoming 2nd generation will be application-oriented. The new architectural concepts of adaptation and reflection are expected to mediate between the 'dynamics' of applications and the 'statics' of resources. Dynamic capabilities enable users to roam or to change QoS requirements according to constraints from their environment. The concept of adaptation either adjusts the balance of resources to maintain graceful degradation, recover service quality, or alternatively inform the end-user of the need to alter a new level of service. The principle of reflection as it is used for the design of QoS platforms enables a system to inspect and to adapt its own internal behaviour. Three known architectural design principles, namely openness, reflection and adaptation, are presented first. Then a novel approach is introduced that claims to glue the dynamics of applications with the statics of resources by adopting all three principles. In that case, QoS-awareness is understood as an up-to-date extension of the principle of openness meeting the requirements of dynamic applications and the requirements of the more static capabilities of resources.

1. Introduction

The 1st generation of QoS-aware architectures solved the problem of managing different services provided by communication-oriented platforms. Theses services include protocols and mechanisms to handle streams, to negotiate, to reserve and to free resources, e.g. buffers, channels, CPU-capacity, specific protocols, etc. to control admission requests for transmissions, to distinguish between different service classes, and to shape traffic according to the characteristics of the network's services. Examples of the 1st generation QoS architectures include CORBA[11], TINA[6], Lancaster-A[4], Heidelberg-A[13][15], Tenet-A[1], and Omega-A[10]. Reviews and comparisons of these architectures can be found in [12] and [1].

The 2nd generation of QoS-aware architectures addresses the problem of flexibility with respect to variable constraints from the business-process driven environment of an application. Support of application-aware flexibility include changing configuration capabilities on the fly (i.e. the service creation model), and keeping control of contracted QoS variables and values (i.e. the service control model). Whereas reflective technology seems to be better suited for configuration purposes, adaptive technology appears to be more applicable for QoS control. Reflective techniques allow reification and altering of system internal states and aspects. Adaptation techniques in general allow any process that is observable and controllable to adopt QoS constraints dynamically. Examples of 2nd generation QoS architectures that are under development are: Lancaster's Reflective Middleware, Illinois' 2K, Cambrigde's Tempest, Columbia's Netscript, as mentioned by Kounavis in [8] or, the Berlin apercu-A[9], as it is presented below in this paper.

During the last decade of developing advanced communication technology and systems there has always been the need to provide architectural approaches that allow easy changes of properties and behaviour of a system. Over time the need increases to 'open' system architectures towards new technologies and flexibility mainly required by mobile applications. This is because applications evolved from local environments to distributed ones, and now involve highly sophisticated services, such as continuous multimedia data flows, mobile behaviour, configuration flexibility etc.

By adopting an improved concept of openness for the design of advanced QoS architectures, we intend to provide answers to questions like:

1) What are the implications of application-oriented QoS on architectures?

2) Can adaptation and reflection be unified into a common model of openness?
3) What is the conceptual difference between reflection and adaptation?

2. The Principle of Openness

In the early days of distributed system design, openness has meant the unrestricted interconnection of heterogeneous stand-alone systems. Openness has been achieved by defining a complex protocol interface in a hierarchical fashion that comprises all aspects of communication from the physical layer to the application layer. This approach is the well-known OSI protocol stack[16] and has been the conceptual model for all types of networks, including the Internet. However, the suggested strong hierarchy of the OSI-model became weakened because it has not been open enough for the various needs of bridging distance in communication networks such as LANs, WANs, MANs. This kind of openness was achieved by inventing 'vertical' interfaces, so-called services that separate protocols in a stack. Hence the fixed order of protocols of the OSI model was replaced by a weak order. Weak ordering allows reshuffling of protocols and the invention of empty places in the stack. The latter helps avoid unnecessary communication.

When development speed of technology became a factor for the design of communication systems, networks had 'to be opened' to allow the exchange of technology without affecting the execution of applications in the same system. In order to achieve this kind of openness, an interface to system-wide network management was suggested. Additionally, applications became more powerful and required more flexibility of the communication services that support them. Next to basic functionality of a service, quality of service became more and more crucial for applications. Similar, as the network management systems enabled awareness to technological changes, the QoS management system enables openness to novel features of applications such as mobility, continuity, non-deterministic behaviour, etc. In order to achieve this fourth and most complex kind of openness, the concept of reflection and adaptation were invented, but from different points of view. Reflection is a more programming-oriented concept and adaptation is an analytical design concept.

To conclude one can say, that architectural design has evolved from pure communication networks to application managing platforms. This development is represented by the four identified aspects of openness. There exists one typical architectural element related to each aspect that in turn 'opens' the architecture of a system:

- Connectivity among systems is achieved by protocols
- heterogeneity of network architecture is achieved by services
- separation of network technology from communication platforms concerns is achieved by network management
- flexibility of applications is achieved by reflective and adaptive capabilities.

3. The Principle of Reflection

Costa et al [5] define a reflective system as "one that is able to inspect and to adapt its own internal behaviour". The emphasis here is placed on the evolution of system behaviour. How can this be achieved? Currently, there is no definitive answer possible, since even the notion of reflection is not yet uniquely defined. Nevertheless, there is agreement on two aspects of reflection. The first one comprises the capability of making system aspects or states explicit that characterize its internal behaviour. The second aspect addresses capabilities of how to alter these internal states according to changed application requirements. Whereas the first aspect is called reification, the second one is termed absorption. Designers who claim to apply these aspects of reflection provide system architectures in which a clear separation between resource management, event management (communications), and property or policy management is anticipated (see Figure 1). Properties that can be modified by users, such as QoS, are called meta-data in the approach of reflection. The adaptation is triggered by the meta-data that contain the values of QoS. The meta-data of, e.g. a component, is structured into information of component encapsulation and environment, their compositional capabilities and into information about used or provided resources.

In order to improve connectivity – another aspect of reflection - Blair et al [3] propose 'receptacles' that allow dynamic bindings among components. With respect to the three-jar partitioning of reflective architectures, as sketched in Figure 1, bindings are suited to property management. To some extend, the receptacles are

comparable to CORBA connectors that reify connections between components in order to dynamically modify connections rather than components. On the meta-level there are rules and policies to enforce modification constraints.

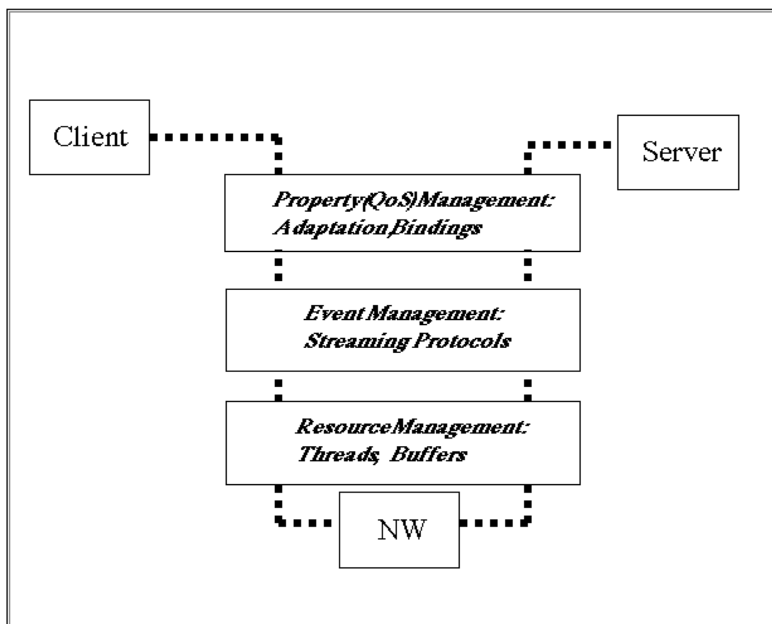

Figure 1: Reflective QoS Architecture

4. The Principle of Adaptation

According to Waddington et al [14] adaptation is used "either to adjust the balance of resources to maintain graceful degradation, recover service quality, or alternatively inform the end-user of the need to alter a new level of service". From this definition one can derive that an adaptation mechanism must be capable of

- gaining information about the current state of the system,
- discriminating system states in order to find out how "to adjust the balance of resources" or "to inform the end-user",
- adjusting resources or informing end-users.

Those mechanisms are not new. One can find them already in today's networks as signaling paths implemented in protocols and routers, i.e. flow control or admission control mechanisms. However, the integration of the application to be designed into a unified system architecture capable of controlling its service quality in real-time is a novel design aspect. The underlying control theoretical approach provides active agents, i.e. sensors that gain information, and actuators that adjust resources or inform third parties. The discriminators, sometimes also called monitors, comprise the system policy and are hence capable of making decisions on the quality of service or on the use of resources.

The architectural principle that is already applied by systems to implement flow-control mechanisms is feed-back. But the architectural principle applied by admission control mechanisms is feed-forwarding. Both principles include links that connect an architectural 'point of observation (PoO)' with a 'point of actuation (PoA)'. In the case of feed-back the PoA steers the source of the controlled flow. In the case of feed-forward the PoA triggers some remedy activities preferably at a network's edge. Here, admission control does not prevent applications from trying to misuse resources or a service. Hence, the forward controlling policy must provide remedy operations that are applied after the misuse has been detected. In contrast, backward controlling policies operate on previously specified target values of one or more system variables being critical with respect to the quality provided. Thus, misuse can occur. Since backward and forward control policies serve different purposes, an architecture that provides QoS-awareness must support both kinds of policies.

Control mechanisms are applied at different levels of abstraction because of their different purposes and because of a system property that is inherent to control loops, called 'direction of effect'. Direction of effect describes a cause-effect relationship. The cause for controlling occurs first at the point of observation and then triggers an action at the point of actuation. Architecturally, we can identify horizontal and vertical control mechanisms. Whereas admission control mechanisms are said to be vertically oriented, flow control and tuning mechanisms are both said to be horizontally oriented. A control mechanism has a vertical effect if it is located at a service boundary between layered protocols in the stack, i.e. at network edges. It has a horizontal effect if the control mechanism is arranged along the information flow

of a protocol. More details about these architectural considerations can be found in [9].

A QoS-aware architecture of a system must be designed such that it is both, testable and observable. Furthermore, QoS-aware systems must also be controllable from the outside via their built-in points of actuation. From a modelling point of view observability and controllability can be expressed in terms of continuous functions. These functions represent flows whose behaviour change over time. In a continuous system model the internal state is represented by a continuous state function *x(t)*[1]. The egress flow, i.e. the flow at the edge of a network or the flow at a component's interface, is represented in the model by the continuous service function *y(t)*[1]. The tuning capabilities are defined by the steering function *u(t)*[1]. The architectural relationship between state and service functions is outlined in figure 2.

A system or a component *S* is said to be observable, if the service *y(t)* comprises all those internal system states, that are represented by *x(t)*, which means *S* is testable (observable) by inspecting *y(t)*. *S* is said to be completely observable, if the state *x(t)* can uniquely be determined from *y(t)* in a finite period of time[7].

A system or a component *S* is said to be controllable, if each of the internal states of *x(t)* can be triggered by a given steering function *u(t),* which means *x(t)* is adjustable by modifying *u(t). S* is said to be completely controllable, if *u(t)* can translate any state $x(t_0)$ into any other state $x(t_1)$[7].

Due to design constraints, some of the internal valuations of the system's or component's state *x(t)* may not be determinable by observing *y(t).* In that case a so-called observer must be constructed that estimates the state value of a continuous variable from observing *y(t).* The condition that such an observer can be designed is called the observability of the system.

[1] Notice, for the functions *x, y, u:* $\Re \rightarrow \Re$ the notations *x(t), y(t), u(t)* denote to signals (streams) that are variant over time.

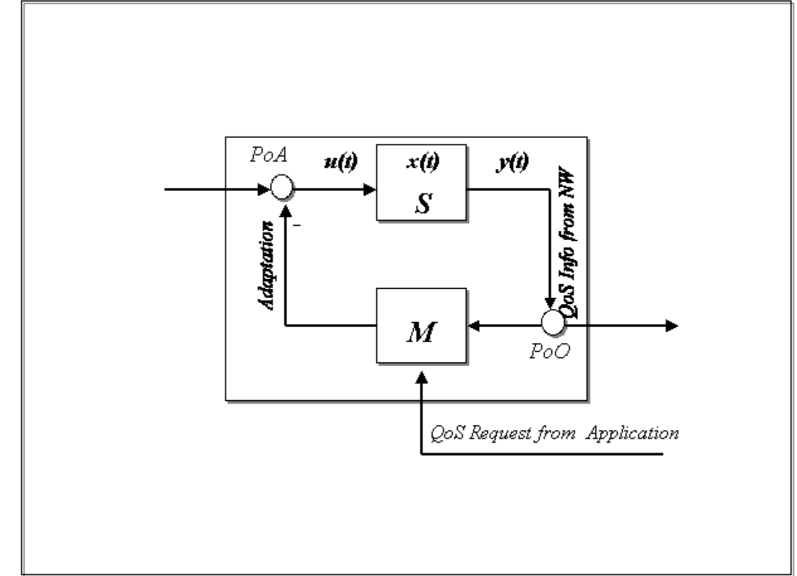

Figure 2: Adaptive QoS Model

The resulting architecture of the composite system is outlined in Figure 2. It is gained from the super-positioning of both the serving component *S* and the controlling component *M*. The state function representing the serving component is the ratio between the output *y(t)* of component *S* and the steering signal *u(t)* of component *M*. If the composite behaviour of the super-positioned system *SM* is *–1*, the system becomes unstable because the denominator of the quotient of $y(t)/u(t)=S/(1+SM)$[2] is zero. Notice, only for $SM \geq 1$ the composite system is stable[7].

[2] The quotient *S/(1+SM)* computes from the transition equations of the components *S* and *M*, i.e. *y(t)=u(t)S* respectively *u(t)=-y(t)M.*

These simple considerations show that adaptive systems containing looped components may not necessarily operate as expected if the behaviour of both components *S* and *M* are considered separately. Composed into an overall system the components *S* and *M* are dependent on each other. Dependability in adaptive systems is the interference of the behaviour of the *S*- and the *M*- components that are architecturally composed into a loop. In order to avoid instabilities resulting from behavioural interference, the analytical power of control theory is applied to calculate predictive policies in order to gain stable and fast tracking adaptive protocols.

5. Architectural Advances

The above considerations show that the two principles of reflection and adaptation are somehow complementary. Reflection technology supports on-the-fly configuration of services and components in a very convenient way. Adaptation technology supports observation and control of critical QoS parameters during run-time. Both capabilities are required to achieve awareness of moves of objects or changes of goals as they happen in an application domain.

One of the most important feature of a 2nd generation QoS-architecture is thus the openness of its internal structures. Openness provides the necessary visibility of internal behaviour. Visibility is required for the various activities that enable QoS management functions, like controlling, filtering, shaping, monitoring etc. More precisely, openness is subdivided into observability and controllability as presented previously. Whereas observability defines the constraints between the internal state variables and the outputs, controllability defines the relationship between the input and the internal state of a component or a system. To take these constraints into account a system designer has to know the state variables that are critical to QoS.

Control requires decision procedures. Decision procedures interpret observations and generate actuation signals for the adaptation of resources or application sources. Thus, decision procedures are policy-driven which may change according to traffic or service conditions in a system. Therefore, the following design objectives can be derived:

- to identify openness constraints for a known QoS policy in terms of observability and controllability functions.
- to identify system variables that are critical to the QoS policy to be achieved and how they relate to input and output of the considered part of the system.
- to architecturally separate control functions from service functions, i.e. to define a clear interface between the control plane and the service or network plane.

In Figure 3 the structure of the Berlin apercu Architecture (apercu-A) is outlined for which the three constraints mentioned above have been considered. apercu-A is a middleware approach that plays the role of a mediator between the application layer and the networking layer. It accepts QoS requests from applications and gathers automatically state information on the resources from network edges and application component interfaces. The sensors are plugged into the interfaces of components dealing with transportation. These resources are said to be open with respect to controlling QoS (compare with openness introduced in section 2). These components are found at the edges of the network which are e.g. routers, shapers etc. Actuators are plugged into application interfaces or into resource management interfaces found at components placed at the networking jar. However, the plugging strategy of sensors and actuators depends on the adaptation policy applied. Not only the behaviour of the application could be adapted to the observed traffic condition, the resources could also be rearranged in such a way that the same quality of service can be maintained even after the observation of contention.

The first class of agents comprises the metering and monitoring agents. The second class contains the actuation agents that are plugged into the resources and services in order to steer their behaviour. The third class of agents of apercu-A is the class of discriminator agents. Discriminators may reside not only in the middleware but also in the various domains of the application and of the network edges. Thus discrimination is a typical distributed process that assists the adopted QoS policy in making decisions. During execution, the decision making process refers also to the negotiated values, thresholds, limits, etc, that are captured by the QoS contract to be achieved. The apercu middleware is capable of providing the infrastructure for the QoS management, i.e. the signalling between sensors, actuators and discriminators, the access to QoS contracts from any location, and the roaming of decision-making agents, i.e. the discriminators. apercu-A achieves a clear distinction between QoS control and service functionality.

The implementation elements of sensors, actuators and discriminators (monitors) of Figure 3 relate to the concepts of the adaptive QoS model of Figure 2. In the example shown, sensors measure the output flow of a network and forward these

measurements to the discriminator *M* which consists of three components distributed over the application domain and the apercu middleware domain. The discriminator decides cooperatively which action must be triggered either in the network or in the application domain according to the QoS contract. The points of actuation are distributed as well over different domains, i.e. in the case of Figure 3 over the application and networking domains. The distribution depends as well on the adaptation policy applied.

Although the apercu middleware is designed to be open to mobile users and configurable applications, its architecture is not a reflective one. Its architectural concepts are consequently derived from the control theoretical model as depicted in Figure 2. Adaptation as applied in reflective architectures is less restrictive than adaptation in adaptive architectures. Thus, reflective architectures provide greater flexibility but at the same time, they are less predictive because the approach of reflection is less strict than the theory of control.

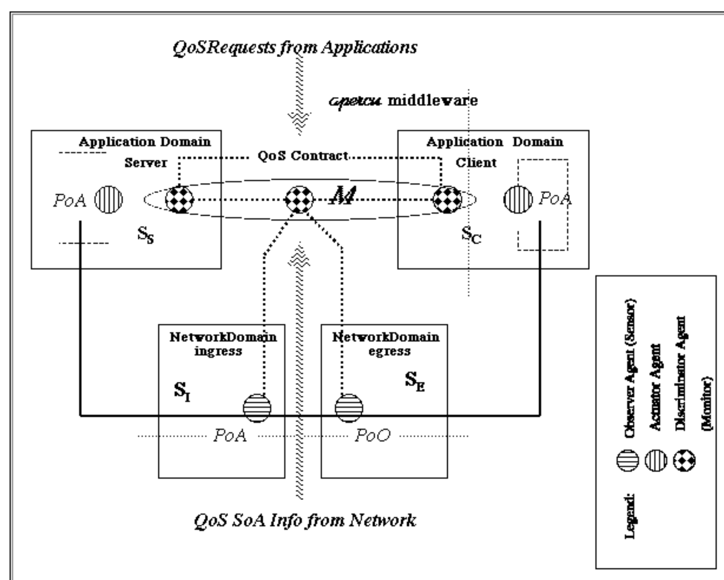

Figure 3: apercu Adaptive QoS Architecture

6. Conclusions

If reflection is intended to support the dynamic rearrangement of components in a system, reflection can be considered as an enabling technology for adaptation. Adaptation requires reflection because of the desired system controllability. Controllability aims at the internal state of a system, e.g. the actual arrangement of resources, which must be modifiable by the external steering function *u(t)* (c.f. 'principles of adaptation' above). The rearrangement of resources is triggered by the actuation signals generated by one of the discriminator agents that reside in the Berlin apercu middleware. Discrimination is the process of decision-making. Decisions are taken according to a model that comprises the characteristic values of the critical variables given in a QoS contract, and a policy for the prediction of the next actuation signal, i.e. the next step of control to be entered. Since decision-making depends on the current internal state of the system, this state must be observable and must be fed to the discriminator agent.

The apercu middleware is a platform that provides an infrastructure that enables the observation and, if necessary, the adaptation of QoS constraints of applications. The platform provides easy-to-plug mechanisms for all types of QoS agents, i.e. sensors, actuators, and discriminators on the meta-level (see principle of reflection). Since the meta-level is organized separately from the service provision level, modifications of measurement, steering and decision policies can easily be applied. Hence, QoS characteristics and policies are able to evolve over time.

Finally, the questions

1) What are the implications of application-oriented QoS on architectures?
2) Can adaptation and reflection be unified into a common model?
3) What is the conceptual difference between reflection and adaptation?

initially raised in this paper can now be answered as follows:

1) Application-oriented QoS requires end-to-end mechanisms, which span one or more networks. Architecturally, it must be possible to insert points of observation and actuation at the network edges. Furthermore control policies must be able to be connected to the observation and actuation points.
2) Platforms that are QoS-aware require agent technology for the dynamic capabilities of sensing, actuation, and discrimination.

3) Adaptation and reflection are complementary[3]. Whereas adaptation provides mechanisms to control a system, reflection provides mechanisms to make systems flexible with respect to application constraints.
4) Both principles, adaptation and reflection, are based on the assumption that systems are no longer fixed in their behaviour by their design. For the 2nd generation self-modification is an accepted concept for the design of QoS-aware platforms.

[3] The authors of this paper argue for a complementary way of consideration. It helps to avoid overlapping development of architectural approaches and QoS mechanisms. Of course other diverging considerations are possible as well.

References

[1] Aurrecoechea, C., Campbell, A.T. and L. Hauw: A Survey of QoS Architectures, ACM/Springer Verlag Multimedia Systems Journal, Special Issue on QoS Architecture, Vol. 6 No. 3, pg. 138-151, May 1998.

[2] Banerjae A., Ferrari D., Mah B.A., Moran M., Verma D.C. and Zhang H. (1996): The Tenet Real-Time Protocol Suite: Design, Implementation and Experiences, IEEE/ACM Transactions on Networking, 4 (1), pp. 1-10.

[3] G. Blair, M. Clarke, F. Costa, G. Coulson, H. Duran, N. Parlavantzas, Lancaster University: The Evolution of Open ORB, presentation at the RM2000 Workshop, 7-8 April 2k, Middleware IBM Palisades.

[4] Campbell A., Coulson G. and Hutchison D. (1994): A Quality of Service Architecture, ACM Computer Communication Review, 24 (2), pp. 6-27.

[5] F.M. Costa, G.S. Blair, G. Coulson: Experiments with Reflective Middleware, Lancaster University, Internal Report MPG-98-11.

[6] E.Koerner (1998): Methods and Elements for the Construction of Collaborated Services in TINA, These de doctorat, University de Liege, ISSN 0075-93333

[7] B.C.Kuo (1995): Automatic Control Systems, Prentice Hall, ISBN 0-13-304759-8

[8] M. Kounavis, COMET Group Columbia University: Presentation of slides at RM2000, Middleware IBM Palisades, April 7-8, 2k.

[9] J.B. deMeer (1999): On the construction of Reflexive System Architectures, RM2000, Middleware IBM Palisades, April 7-8, 2k.

[10] Nahrstedt K. and Smith J.M. (1996): Design, Implementation and Experiences of the OMEGA End-Point Architecture, IEEE Journal on Selected Areas in Communications, 14 (7), pp. 1263-1279.

[11] Quality of Service (QoS) OMG Green Paper, Working Draft, Version 0.4a, OMG, June 1997, ftp://ftp.omg.org/pub/docs/ormsc/97-06-04.pdf

[12] A. Sloane, University of Wolverhampton, D. Lawrence Middlesex University (Editors): Multimedia Internet Broadcasting, book to appear in early 2001, Springer ISBN 1-85233-283-2.

[13] Vogt C., Wolf L.C., Herrtwich R.G. and Wittig H. (1998): HeiRAT - Quality of Service Management for Distributed Multimedia Systems, to appear in ACM Multimedia Systems Journal - Special Issue on QoS Systems.

[14] D. Waddington, D, Hutchison: A General Model for QoS Adaptation, 1998 6th IWQoS Proceedings Napa Valley California May 18-20, pp. 275-278.

[15] Wolf L.C. and Herrtwich R.G. (1994): The System Architecture of the Heidelberg Transport System, ACM Operating Systems Review, 28 (2), pp. 51-64.

[16] ISO/IEC JTC 1/SC 21 N8228 revised: Revision of the OSI Basic Reference Model ISO/IEC 7498-1, February 1994