POLICY-BASED BANDWIDTH MANAGEMENT FOR TACTICAL NETWORKS WITH THE AGILE COMPUTING MIDDLEWARE

Niranjan Suri[1,2], Marco Carvalho[1], James Lott[1], Mauro Tortonesi[3], Jeffrey M. Bradshaw[1], Mauro Arguedas[1], and Maggie Breedy[1]

[1]Institute for Human & Machine Cognition
[2]Lancaster University
[3]University of Ferrara
{nsuri, mcarvalho,jlott,jbradshaw,marguedas,mbreedy}@ihmc.us; mtortonesi@ing.unife.it

ABSTRACT

Bandwidth allocation and enforcement in tactical networks is a challenging problem. The mobile ad-hoc wireless environment is bandwidth constrained and the bandwidth required by applications running at any given moment in time typically exceeds the bandwidth available. In addition, both network topology and availability of network resources vary rapidly in the mobile ad-hoc scenario. Therefore, it is important to properly realize allocation of bandwidth to the competing applications, monitoring of both available and assigned network resources, and the enforcement of constraints on channel usage.

IHMC's agile computing middleware and KAoS policy and domain services components provide a resource management platform that enables dynamic control of application bandwidth utilization in a transparent manner.

The agile computing middleware performs bandwidth monitoring and enforcement. It builds on top of the Mockets library for the realization of traditional client/server application-level communications. The middleware also integrates with the FlexFeed component to provide applications with publish-subscribe communications semantics and to support service-specific instream data manipulation.

The KAoS policy and domain services handle policy specification and distribution. KAoS policies can be used to specify bandwidth limits based on hosts, port numbers, and/or data flows.

INTRODUCTION

Communications in tactical military environments are often constrained by low bandwidth due to the wireless and ad-hoc nature of the networks. Applications often require more bandwidth than available. Therefore, the available bandwidth needs to be allocated to competing application needs based on the priority of the tasks being performed.

There is a need for a distributed resource management system that allocates bandwidth to distributed applications according to their priority and desired QoS levels. The system should also enforce bandwidth constraints in order to make sure that applications do not consume more bandwidth than assigned.

In addition, the resource management system should detect changes in network resource availability and trigger a reallocation of bandwidth when needed. In fact, the dynamic properties and behavior of mobile ad-hoc networks result in a considerable fluctuation of the available bandwidth on a continuous basis, which has a significant impact on application behavior.

IHMC's agile computing middleware [1] and KAoS policy and domain services components work together to address the challenges described above. The middleware provides two libraries – Mockets and FlexFeed – that handle the communication needs of applications. One of the key goals for the middleware has been to transparently handle bandwidth monitoring and enforcement without requiring significant changes to the applications.

Mockets is a comprehensive communications library that can be used by applications instead of other transport protocols such as TCP and UDP [2] [3]. Mockets has been optimized to operate on mobile ad-hoc networks and provides several enhancements and a better abstraction to improve the performance of applications.

FlexFeed is a publish-subscribe systems that handles data feeds from providers (such as sensors) to clients [4] [5]. FlexFeed handles hierarchical data distribution to optimize bandwidth utilization and dynamically transforms the data instream to support policy enforcement. Unlike Mockets,

FlexFeed is data-aware, which allows it to operate on the data in order to adapt to a dynamic environment. For example, FlexFeed can dynamically downsample images and video, change the compression ratio, or change the update rate in order to adapt to changing network conditions.

KAoS policy services can be used to express policies governing the runtime behavior of the system [6] [7]. These policies are a combination of authorization policies and obligation policies. The authorization policies allow bandwidth to be allocated to the desired applications (or to particular data flows). The obligation policies state minimum requirements in terms of either bandwidth or service levels to be expected from the application. A monitoring component checks the system for compliance with obligation policies and triggers a reallocation or a notification to a human operator if necessary.

This paper is organized as follows. Section two presents a scenario to motivate the need for bandwidth management and the capabilities of the middleware. Section three presents an overview of Mockets, FlexFeed, and KAoS. Section four describes the bandwidth allocation, enforcement, and monitoring mechanisms. Section five presentsa few additional implementation details. Some related work is discussed in section six, followed by conclusions.

MOTIVATING SCENARIO

The motivating scenario for this paper is based on a MOUT (Military Operations in Urban Terrain) exercise that was part of the Warrior's Edge initiative of the Horizontal Fusion Portfolio's Quantum Leap demonstrations.

The scenario involves a number of dismounted soldiers and robotic platforms operating together in an urban setting. All the nodes are connected via a mobile ad-hoc network, which carries network traffic for voice, video, as well as a number of applications. The robots are equipped with several sensors, including cameras that can transmit live motion video to any subscriber. Sensors are also embedded into the environment and integrated into the soldier equipment. Various servers in the environment provide services that are invoked by clients supporting soldiers as well as personnel monitoring the operation at a command center. Clients may request live motion video from any of the camera sensors. Requests have different priorities, based on the nature of the data, the person performing the request, as well as the role of the requestor. For example, the driver of a robot getting a video feed from a camera on the robot is a higher priority than other observers requesting video or other data. Moreover, there are more stringent requirements on minimum service levels (higher frame rate and resolution and lower latency for video delivery) for robot operators who must drive the robot effectively.

In this context, the goal of the research described in this paper is to realize dynamic policy-based management of bandwidth assignment between multiple competing demands. One of the design guidelines of the proposed solution is to separate policy management operations from policy enforcement and network conditions monitoring.

The agile computing middleware enforces policies transparently to the applications and users. It also keeps track of the allocated resources and monitors network conditions. In case the system is unable to comply with the policy requirements, it notifies a human operator or other system components about the failure condition so that appropriate steps can be taken to modify the policies.

OVERVIEW OF MAJOR COMPONENTS

Mockets

Mockets (i.e., mobile sockets) is a novel communication middleware, specifically designed to address the peculiar challenges of mobile ad-hoc networking [2] [3]. Mockets supports the mobility of communication endpoints, with the goal of facilitating user/terminal/code mobility. In addition, Mockets provides advanced communication semantics which permit the realization of adaptive applications capable to react to abrupt changes in network conditions. Finally, the Mockets middleware offers an application control and monitoring of the connection status and network conditions.

Mockets adopts the traditional client/server programming paradigm of sockets with additional features. For example, it provides a message-oriented communication API with advanced functionalities to manage endpoint mobility, modulate the quality of service, and monitor network conditions. Mockets also offers a second, stream-oriented API compatible with TCP Sockets to facilitate the task of porting legacy applications to the new middleware. However, applications using the stream-oriented API will not benefit from the advanced functionalities of Mockets.

One of the main goals of the Mockets middleware is the support for device mobility. Therefore, one of the design guidelines for the middleware is the notion of the mocket as a communication endpoint that can move from one host to another without interrupting the communication session. The migration of a mocket connection is triggered

by an explicit application level event/command and is completely transparent to the remote application. In addition, the Mockets middleware can interact with the underlying network layer to detect the migration of a host to a different network locality. In this case, Mockets performs the reconfiguration/rebinding of all the mockets present in the host by notifying the remote endpoints about the network layer address change. The ability of the Mockets middleware to perform migration of connection endpoints and network driven connection reconfiguration is fundamental to permit applications to continue their execution even in presence of device mobility. Without this feature, applications would be forced to shut down and reinstantiate all the opened network connections upon host migration.

Mockets also proposes a novel programming model that does not masquerade communication channel characteristics but instead exposes network conditions to applications. Researchers in several areas have identified this capability as crucial for the development of robust and scalable distributed applications that are able to adapt to the varying network resource availability [8] [9]. Mockets supports the development of adaptive applications by providing them with both a wide range of message delivery semantics and timely and precise information about current network resources availability.

Mockets offers several message delivery services with different communication semantics and allows applications to choose the best suited one depending on application logic, network conditions, and user preferences. For instance, applications can choose any combination of reliable/unreliable and sequenced/ unsequenced delivery and assign transmission priority and information lifetime on a per message basis. The Mockets library also supports the classification of messages into different group types to permit applications to perform group operations on specific type of messages. Typical operations include enforcing a maximum transmission bandwidth (as in this paper), assigning a specific lifetime or a transmission priority value, or canceling/replacing a group of enqueued messages. In addition, Mockets permits fine-tuning the performance of applications by setting the transmission priority and maximum lifetime properties of messages.

Finally, the Mockets middleware monitors network conditions and provides the gathered information up to the application level. In this way, applications can operate more informed decision on how to tailor services, according to both service logic and user preferences.

Applications can either directly interrogate the Mockets monitoring facility or request to be notified when a specific event occurs by registering callback functions. For instance, one of the events about which applications may be notified is peer unreachability. This condition is detected by a keep-alive mechanism that allows quick discovery of problems at the link and network layers.

Historically, transport protocols have always been designed to masquerade varying network conditions to applications. Although the adoption of the subscribe-notify paradigm is rather unusual in network programming, the need for a richer model of interaction between applications and transport protocols has begun to emerge in other recent proposals [10] [11].

FlexFeed

FlexFeed is a publish/subscribe communications framework for dynamic in-stream data processing in mobile ad-hoc network environments under policy and resource constraints [4] [5]. The framework uses mobile agents as data-aware processing elements to better customize multicast trees and allocate in-stream data processing capabilities in the network. In-stream data processing relies on taking advantage of the multi-hop nature of network paths in ad-hoc networks to appropriately allocate data processing elements for some optimization criteria.

In the agile computing middleware, FlexFeed is the API that application utilize for publish-subscribe oriented communications. In the context of data-aware publish-subscribe systems, it opportunistically allocates computational resources in the network while taking into consideration load, connectivity, and bandwidth availability in order to minimize overall costs for data processing and distribution of multiple concurrent data feeds.

In order to provide decentralized mechanisms for high level policy definition, deconfliction and distribution, the FlexFeed framework has been integrated with KAoS policy services as its default framework but other approaches could be straightforwardly adapted. Upon policy distribution, the FlexFeed framework is responsible for providing on-demand deployment and activation of policy enforcers.

In FlexFeed, policies can be used to regulate local (and global) resource utilization of concurrent data feeds, as well as to regulate the context-dependent release of information between nodes. FlexFeed supports the deployment of customized data filters at run-time that can

be used as data-aware policy enforcers for specialized data types. For example, details of images being transmitted by a particular camera sensor can be transparently downgraded for clients not authorized to access the full resolution video.

Policies can also be used to regulate and constrain the autonomous behavior of the framework, providing bounds for self-adjustments to operation tempo and to the proactive manipulation of resources. For example, policies can specify the conditions under which resources can be used or moved in order to restore communication loss.

While other aspects of policy management are performed by KAoS, the enforcement of policies is autonomously handled by FlexFeed. The framework will opportunistically allocate (and monitor) the necessary resources for policy enforcement. When resources available are insufficient for the policy requirements, the framework reports the issue to the policy infrastructure, requesting assistance.

KAoS

KAoS is a collection of componentized services compatible with several popular agent and robotic platforms as well as general-purpose grid computing, CORBA, AFRL's Joint Battlespace Infosphere, and Web Services environments. KAoS Domain Services provide the capability for groups of software components, people, resources, and other entities to be semantically described and structured into organizations of domains and subdomains to facilitate collaboration and external policy administration. KAoS Policy Services allow for the specification, management, conflict resolution, and enforcement of policies within domains.

The KAoS policy ontology distinguishes between *authorizations* (i.e., constraints that permit or forbid some action) and *obligations* (i.e., constraints that require some action to be performed when a state- or event-based trigger occurs, or else serve to waive such a requirement). Other policy constructs (e.g., delegation, role-based authorization) are built out of the basic primitives of domains plus these four policy types.

Ontologies are represented in OWL (Web Ontology Language), which enables reasoning about the controlled environment, policy relations and disclosure, policy conflict detection, and harmonization, as well as about domain structure and concepts exploiting description-logic-based subsumption and instance classification algorithms and, if necessary, controlled extensions to description logic (e.g., role-value maps). No rules are used in policy representation—rather conditions are expressed as property restrictions on actions associated with the policy ontologies.

Specialized ontologies may be loaded to fit a particular application. For this application, an ontology was created for Mockets, which defines the actions and properties associated with Mocket communication. As an example, the Mocket ontology defines an action *OpenMocketConnectionAction* with properties such as *hostName*, *port*, *bandwidthLimit*, *dataflow*, etc. In this way, context-specific policies can be defined for a given application.

The KAoS Policy Administration Tool (KPAT) is a GUI which allows users to view and modify registered domains, agents, and other entities, as well as to commit new policies, and to modify or delete them. KPAT hides much of the complexity of the OWL policy representation from users. When a user commits a change to ontology (e.g., a new or edited policy, changes to domain structure) the Jena (http://www.hpl.hp.com/semweb/) toolkit is used to dynamically build a OWL representation based on the values selected by the user. To assist non-specialists in defining sensible policies for a specific application, the KAoS Policy Administration Tool (KPAT) supports both a generic OWL policy editor, as well as policy templates tailored to specific applications.

KAoS provides a simplified interface for application-specific policy enforcement mechanisms to interact with the policy services, which also insulates them from the complexities of the OWL-based policy representation and reasoning. An application can perform various policy disclosure queries with KAoS to determine whether to allow an intercepted action to occur, asking questions such as “Is this action allowed?”, “What are the allowed values for this property of a given action?”, and “What are the obligations for the given action?”. Thus the application developer does not need to be concerned with the details of evaluating the applicability of policies, but can rather focus on the implementation of the enforcement capability itself. For example, the Mocket policy enforcement mechanism asks queries such as “What is the allowed value for *bandwidthLimit* for an *OpenMocketConnectionAction* to *hostA*, *port 80*?”

KAoS relies on efficient and logically decidable description logic based subsumption and classification methods. This approach allows the policy disclosure methods in KAoS to be optimized such that the response to a query from an enforcer is provided on average in less than 1 ms. Further details about the performance and evaluation of KAoS can be found in [12].

BANDWIDTH ALLOCATION, ENFORCEMENT, AND MONITORING

Figure 1 shows the interactions between the middleware components. One or more administrators may use the KAoS Policy Administration Tool (KPAT) to express policies about the system. These policies are stored in the distributed instances of the KAoS Directory Service. While this component is shown as one unit in the diagram, it may be replicated at several points in the network.

Each host contains a KAoS proxy component that caches relevant policies locally. The FlexFeed and Mockets components of the middleware interact only with the local KAoS proxy components to query policies. For example, if an application on a host attempts to use Mockets to open a connection to another application, the Mockets library queries the KAoS proxy to ensure that the communication is authorized. The KAoS proxy also pushes policy changes to Mockets and FlexFeed via a callback mechanism.

Applications operating in the environment use either Mockets or FlexFeed to handle their communications. Publish-subscribe applications such as video clients and video servers use FlexFeed. Applications that expect to use a TCP or UDP style communication metaphor use Mockets. Doing so allows the agile computing middleware to obtain information about the communication patterns in the environment – at the level of hosts, applications (clients or servers) on hosts, and the data flows between applications. The information obtained includes the current bandwidth utilization, connection attempts, connection failures, connection losses, and other statistics such as packet loss, retransmissions, and queue lengths.

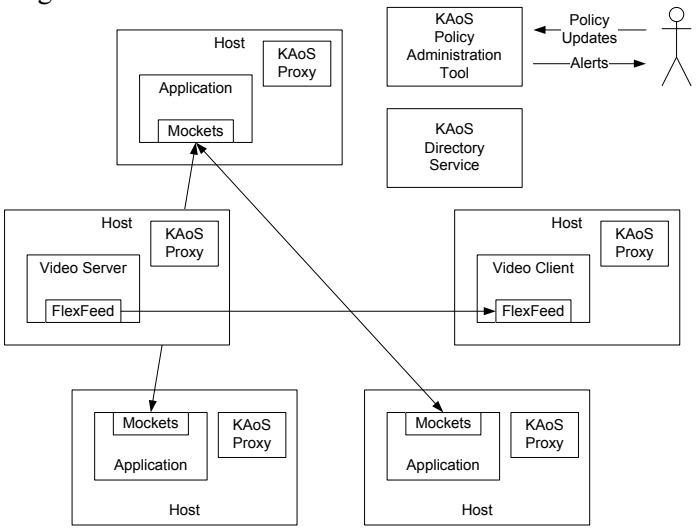

Figure 1: Middleware Components and Interactions

Another key requirement is transparent policy enforcement. Once applications start using Mockets and FlexFeed, no further changes are necessary to use KAoS policies to control application behavior. Mockets supports two different types of policies – an authorization or negative authorization policy that allows or denies connections between specified applications and an authorization policy that allows a specified amount of bandwidth to be used by a connection (or a type of data flow within a connection).

FlexFeed supports three different types of policies – an authorization or negative authorization policy allowing or denying an application to subscribe to data being provided by another application, and an obligation policy that specifies minimum service levels for subscriptions. FlexFeed supports additional policies that can constrain the nature of the data flowing across subscriptions, but those are discussed elsewhere [4] [5].

To support the type of scenario described earlier in the paper, several different types of policies are used. Authorization policies are first established to allow communications between applications that need to exchange information. Based on previously observed or expected behavior, initial bandwidth limits are specified for different applications. Finally, obligation policies are used to specify minimum service levels for operators of robotic platforms. We are working on mechanisms for automated adjustment of policy [13].

At runtime, Mockets and FlexFeed constrain bandwidth used by applications as specified in the initial policies. Moreover, a monitoring component periodically checks the obligation policies to make sure the system is still in compliance. Available bandwidth fluctuates continuously during runtime. Within reasonable limits, the middleware automatically manages the allocation as the network conditions change. However, drastic changes in the situation or network may result in the middleware being unable to comply with the demands being placed by the applications. In such cases, the monitoring component will detect that the system has failed to comply with one or more obligation policies. This component triggers a notification that is sent to the system manager or administrator (who could be a person or an agent). This entity can then change the initial allocation of bandwidth by changing the policies in place. Once the new policies are propagated to the KAoS Proxies, the FlexFeed and Mocket components are notified about the policy changes. This will result in the system behavior changing and (hopefully) complying with the obligation policies.

IMPLEMENTATION DETAILS

Mockets transparently enforces policies placed on connections. When an application attempts to connect to another application, Mockets passes the source IP, source port, destination IP, and destination port to the KAoS Proxy to check if a connection is allowed. The application accepting the connection also performs a similar check and will only accept connections that are allowed. If a connection is allowed, the KAoS Proxy will return the maximum bandwidth that is applicable to the connection. Any future changes to the allowed bandwidth are pushed to the Mockets component through a listener interface between Mockets and the KAoS Proxy, which allows the bandwidth limits to be changed dynamically at any point during the lifetime of the connection.

The transmitter component of Mockets then computes the maximum number of bytes that can be written per 100 ms. When the application sends data by calling the write() method in Mockets, the transmitter will ensure that the data sent out on the network is throttled to be below the specified limit. For example, if the bandwidth limit is 20 KB/sec, the application is allowed to send 2 KB every 100 ms. Therefore, Mockets will ensure that no more than 2 KB is transmitted within each 100 ms interval. If an application attempts to write more data, the application thread will be temporarily put to sleep until the application is allowed to send more data. The enforcement is completely transparent to the application.

Data flowing through the network must be tagged into different categories so that it can be differentiated by the middleware. The applications, the data providers, and the data consumers must also be known and visible to the middleware.

RELATED WORK

Many proposals from both industry and academia for QoS policy enforcement focus the adoption or extension of available standards developed by IETF [14] [15]. However, these frameworks do not provide any integration between the policy decision and enforcement entities and the applications. In fact, they do not notify applications in case of QoS policy changes, thus effectively preventing them to react properly by modulating their service level. Our approach goes beyond the one described above, as the integration of the agile computing middleware with KAoS allows both the decoupling of policy operations (creation/modification/ decision/enforcement) and protocol/service logic, and the notification of changes in the maximum available QoS to applications.

Other proposals such as [16] advocate the realization of policy-based QoS adaptation in the framework transparently to applications, by dynamically changing the communication protocols at runtime. However, this approach takes full control of adaptation strategies away from applications, and prevents them to modulate their service levels according to service logic and user preferences. The novel programming model proposed by Mockets and KAoS, which realizes a tighter coupling between applications and the underlying network, allows the development of more robust and scalable distributed applications that can better adapt to the dynamic and heterogeneous deployment scenarios.

CONCLUSIONS

This paper has described policy-based bandwidth management in the agile computing framework. Applications operate on top of the Mockets and FlexFeed communication libraries, which integrate with the middleware. They provide information to the middleware with respect to application communication needs and patterns. The KAoS policy and domain services framework is used to express policies regarding bandwidth allocation among competing application needs as well as minimum quality of service requirements for applications. These policies drive the bandwidth allocation in the middleware, which is enforced by the Mockets and FlexFeed libraries. Monitoring components allow the middleware to determine when obligation policies defining QoS requirements are not satisfied, which results in notifications to the administrator. In case of policy changes, the new bandwidth allocations are propagated back to the middleware, which changes the enforcement accordingly.

AKNOWLEDGEMENTS

This work is supported in part by the U.S. Army Research Laboratory under Cooperative Agreement W911NF-04-2-0013, by the U.S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0009, and by the Air Force Research Laboratory under Cooperative Agreement FA8750-06-2-0064.

REFERENCES

[1] Suri, N., Bradshaw, J.M., Carvalho, M., Cowin, T., Breedy, M., Groth, P., and Saavedra, R. Agile Computing: Bridging the Gap between Grid Computing and Ad-hoc Peer-to-Peer Resource Sharing. In P*roceedings of the 3rd*

IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003).

[2] N. Suri, M. Tortonesi, M. Arguedas, M. Breedy, M. Carvalho, R. Winkler, Mockets: A Comprehensive Application-Level Communications Library, in *Proceedings of 24th Military Communications Conference (MILCOM 2005)*, Atlantic City, NJ, USA, October 2005.

[3] M. Tortonesi, C. Stefanelli, N. Suri, M. Arguedas, M. Breedy, Mockets: A Novel Message-oriented Communication Middleware for the Wireless Internet, in *Proceedings of International Conference on Wireless Information Networks and Systems (WINSYS 2006)*, Setúbal, Portugal, August 2006.

[4] Carvalho, M. and Breedy, M. (2002) Supporting Flexible Data Feeds in Dynamic Sensor Grids Through Mobile Agents. In *Proceedings of the 6th International Conference in Mobile Agents* (MA 2002) Agents, Barcelona, Spain, October 2002.

[5] Carvalho, M., Suri, N., Arguedas, M. (2005) *Mobile Agent-based Communications Middleware for Data Streaming in the Battlefield*. To appear in Proceedings of MILCOM 2005, October 2005, Atlantic City, New Jersey.

[6] Bradshaw, J. M., Uszok, A., Jeffers, R., Suri, N., Hayes, P., Burstein, M. H., Acquisti, A., Benyo, B., Breedy, M. R., Carvalho, M., Diller, D., Johnson, M., Kulkarni, S., Lott, J., Sierhuis, M., & Van Hoof, R. (2003). Representation and reasoning for DAML-based policy and domain services in KAoS and Nomads. Proceedings of the Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2003), Melbourne, Australia, New York, NY: ACM Press, pp. 835-842.

[7] Uszok, A., Bradshaw, J. M., Johnson, M., Jeffers, R., Tate, A., Dalton, J., & Aitken, S. (2004). "KAoS policy management for semantic web services." IEEE Intelligent Systems **19**(4): 32-41.

[8] Chang, F., Karamcheti, V., 2001. A Framework for Automatic Adaptation of Tunable Distributed Applications, *Cluster Computing*, Vol. 4, N. 1, pp. 49-62, March 2001.

[9] Cheng, L., Marsic, I, 2002. Piecewise Network Awareness Service for Wireless/Mobile Pervasive Computing, *Mobile Networks and Applications*, Vol. 7, N. 4, pp. 269-278, August 2002.

[10] Gross, T., Steenkiste, P., Subhlok, J., 1999. Adaptive Distributed Applications on Heterogeneous Networks, in: *Proceedings of the 8th Heterogeneous Computing Workshop*.

[11] Kim, M., and Noble, B., 2001. Mobile Network Estimation in: *Proceedings of the 7th annual international conference on Mobile computing and networking (MOBICOM 2001)*, Rome, Italy.

[12] Lott, J., Bradshaw, J. M., Uszok, A., & Jeffers, R. (2004). Using KAoS policy and domain services within Cougaar. *Proceedings of the Open Cougaar Conference 2004.* New York City, NY, 20 July, pp. 89-95.

[13] Bradshaw, J. M., Jung, H., Kulkarni, S., Johnson, M., Feltovich, P., Allen, J., Bunch, L., Chambers, N., Galescu, L., Jeffers, R., Suri, N., Taysom, W., & Uszok, A. (2005). Toward trustworthy adjustable autonomy in KAoS. Trusting Agents for Trustworthy Electronic Societies. R. Falcone. Berlin, Springer**.**

[14] H. Zheng, M. Greis, Ongoing research on QoS policy control schemes in mobile networks, *Mobile Networks and Applications*, Volume 9, Issue 3, Pages 235-242, June 2004.

[15] K. Phanse, Quality of Service (QoS) Policy Framework.

[16] L. Rosa, A. Lopes, L. Rodrigues, Policy-Driven Adaptation for Protocol Stacks. In *Proceedings of the IEEE Self-adaptability and self-management of context-aware systems workshop (SELF)*, Santa Clara, CA, USA, July 2006.