ARC: a Bottom-Up Approach to Negotiated QoS

Hylke van Dijk
Koen Langendoen
Henk Sips

publication	:	UbiCom-Publication/2000/4
date	:	October 10, 2000
to appear in	:	3rd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2000)
status	:	final
keywords	:	mobile computing, multimedia, adaptive systems, quality of service, negotiation, hierarchy.

The work reported here has been funded by the Delft University of Technology DIOC research programme. DIOC programmes are an initiative of the ARTD ("Advisory Council Technology T.U. Delft"), and aim at solving, in a multi-disciplinary way, specific technological problems with engineering and social significance.

Ubiquitous Communications
Delft University of Technology
P.O. Box 5031, 2600 GA Delft,
The Netherlands

www: www.ubicom.tudelft.nl
email: info@ubicom.tudelft.nl
phone: +31 15 2786206
fax: +31 15 2786190

ARC: a Bottom-Up Approach to Negotiated QoS

Hylke van Dijk Koen Langendoen Henk Sips
Faculty of Information Technology and Systems
Delft University of Technology, The Netherlands
{hylke,koen}@ubicom.tudelft.nl

Abstract

Mobile systems operate in a resource-scarce environment and thus must adapt to external conditions; all layers must make cost-based decisions about what mode of operation to use in response to performance feedback. This paper focuses on the generic interface between adjacent layers (client and server) in a multi-level hierarchy. The Adaptive Research Contracts *(ARC) framework uses a bottom-up approach in which the server exposes a range of quality/cost modes to the client above. This allows the client to trade off various algorithms generating different workloads for multiple resources. A case study shows that control can be distributed effectively over multiple layers with ARC; global cost-effective solutions can be obtained with exchanging a small fraction of all possible control settings.*

1. Introduction

Developing applications and systems software for mobile computing is a challenging task, because available resources such as processing power, memory, and battery energy are limited. To warrant a reasonable time of operation, software – applications and operating system – should collaborate to make the most efficient use of hardware resources. When wireless communication is required, the task becomes even more complicated, because of varying external conditions that influence the quality of the wireless transmission (i.e., effective bit rate). Changes can result from a number of different sources:

- the location and speed of a mobile user determine the transmission errors caused by multi-path fading, interference, etc.
- interactive applications, like web browsing, typically generate bursty data traffic over the wireless link.
- the number of users in a cell and their activity directly influence the available bit rate in a best effort network.

The combined effect of all these fluctuations is that changes in performance and demands of the wireless link can be quite large *and* quite sudden. This calls for adaptive systems that continuously reconsider their behavior.

The most scarce resource in a mobile terminal is battery energy. Consequently, adaptive components always consider their alternatives with respect to the power they dissipate. When multiple resources are involved, interesting trade-offs can be made. For example, it might be possible to off-load tasks to the backbone network at the expense of increasing the traffic over the wireless link. More specifically, when streaming video from a server in the internet to a mobile client, a trade-off can be made between compression and communication. If the video stream is compressed heavily, then a few kbps will be transmitted over the wireless link, but significant processing power is required for decoding. If, on the other hand, the video is transmitted directly, a lot of energy is consumed by the wireless radio receiving the uncompressed data stream. The optimal solution depends on the power dissipation of the radio and processor under various workloads induced by different compression factors. The best strategy cannot be determined when establishing the video stream, since the performance of the radio changes over time, influencing the trade-offs involved.

In general, adaptive systems must make a trade-off between the Quality of Service (QoS) they offer and the resources (Cost) they use. Since adaptive systems are structured hierarchically, just as any other complex system, additional effort is required to coordinate the (independent) decisions at different layers. Adjacent layers, which we will refer to as client (upper) and server (lower), need an interface to exchange knowledge. At the minimum, a client must know the current performance offered by the server below, so it can adapt accordingly. For making trade-offs involving alternative solutions, as with the video streaming example above, servers must in addition offer a range of QoS/Cost alternatives. This leads to a system where adjacent layers, each having multiple alternatives, must negotiate about the best overall combination. This paper outlines a generic approach, named *Adaptive Resource Contracts* (ARC), that structures such negotiations.

2. Related work

Quality of Service has been addressed by a large number of researchers. This section focuses on the negotiation aspects, and reviews various approaches described in literature; a general survey of QoS architectures is presented in [11]. The simplest negotiation scheme is *call admission*, where at the time of establishing a connection the application specifies its requirements; the network layer tries to allocate the necessary resources on the route to the final destination and reports success or failure. For example, the ATM standard specifies how service requirements must be formulated; at call setup a client selects a traffic class (e.g., Constant Bit Rate), and supplies a number of parameters. Call admission does not support negotiations during a call, so applications are tempted to allocate enough throughput to accommodate their most demanding mode of operation. This worst-case allocation behavior locks up network resources that could be used by others most of the time.

With the rise of multi-media applications (e.g., video conferencing) less rigid QoS policies have been developed to support applications that can dynamically scale their demands (e.g., reduce the frame rate). The typical approach is to have the application specify its capabilities through *benefit functions* [5], also known as *utility functions* [3]. A benefit function relates the network performance (i.e., bit-rate) to the quality that the application can realize with it. This information allows graceful handling of network congestion. Whenever the total traffic exceeds the network capacity, a decision can be made which client should adapt, and to what level, while maintaining the highest overall quality. Instead of optimizing for overall quality, other policies may be implemented in the network layer [3, 9].

Although benefit functions allow for a rather flexible use of resources, the disadvantage is that knowledge propagates *downwards* in the hierarchy. Consequently, decisions must be taken at the lowest level where all knowledge accumulates. Taking decisions at a centralized point becomes too complex for large systems involving many different resources. It is impractical to combine expert knowledge of all resources involved and still make effective trade-offs, for example, between processing and communication. A better approach is to let QoS information (i.e., feedback) flow *upwards*. Decisions can then be made at the "right" level. Another advantage is that the bottom-up approach effectively filters most of the QoS changes at the lower layers reducing the number of adaptation decisions at the upper layers; only large QoS changes propagate to the top and incur handling costs at each layer.

When a server is capable of providing performance feedback, a simple feedback loop can be used to control the adaptations in the client on top. This approach is taken by the SWiFT [6] and AQUA [8] projects, which provide generic support to control processing and network resources. Feedback loops work well if the service can be characterized with one parameter and fluctuations are relatively small. Both assumptions do not hold for wireless communication systems, resulting in ad-hoc control structures. For example, Bodic et al. map generic QoS requests to predefined bearer classes, and dynamically reconsider this mapping to adapt to changes in the wireless link [4]. The set of bearer classes is kept small so that coarse decisions (i.e., remapping) can be taken by a single adaptation component. The UNIQuE project takes a more structured approach and uses a hierarchy of monitor and control components to adapt process, data, and network scheduling [1]. The feedback, however, is implicit through buffer under and overflow signalling, which makes reasoning about alternatives difficult.

The work by Bhatti and Knight [2] on adaptive internet applications is very interesting, because it provides performance feedback in terms of application specific parameters. The application specifies its own so-called QoSSpace defined as an orthogonal combination of parameters. The parameters are selected such that each mode the application may operate in, can conveniently be described as a subspace (e.g., a cube in a 3-D parameter space). The network performance feedback is then mapped into the QoSSpace signalling how well each sub-space can be served under current conditions. Since the feedback is in application-specific terms, adapting to changes amounts to selecting the mode associated with the best sub-space. Unfortunately, the QoSSpace approach cannot be used in the area of mobile computing since costs (i.e., power) are not addressed. Supporting cost aware systems requires a different control structure as will be shown in the next section.

3. Multi-level optimizations

Layers in a hierarchically structured system that operates in a resource-scarce environment must be prepared to share knowledge about their alternative modes of operation so cost-effective decisions can be made. In contrast to traditional approaches it is important that both layers, referred to as client and server, expose their modes of operation. The server should not quote only the best performance it can offer, but list a range of alternatives *and* their associated costs instead. The client should make its different modes of operation explicit by stating the different possible workloads and their associated qualities. When both sources of information (i.e., operation spaces) are available an optimization can be carried out to select the best combination of client and server mode. The resulting contract is used to control the operation mode of client and server.

Figure 1 gives a simplified overview – two layers only – of the general approach to multi-level optimizations. A pre-

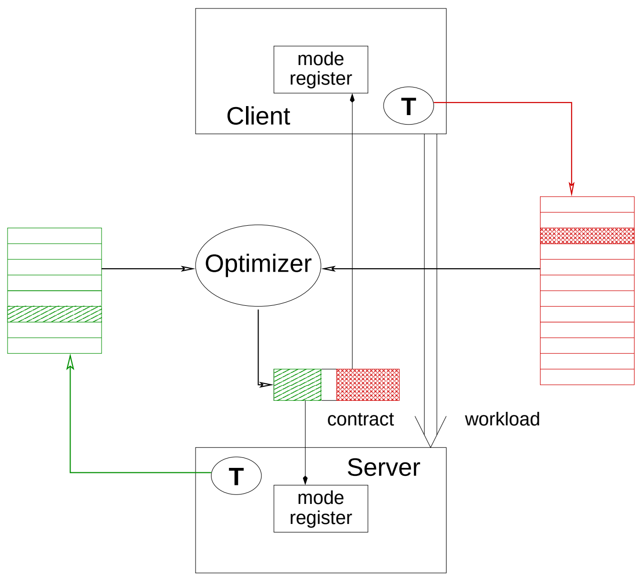

Figure 1. Operation-space optimization.

requisite is that client and server use a common language; the client's workload and server's performance must be described by the same parameter tuple. For example, when the server is a wireless link obvious parameters are throughput, error rate, and energy consumption. Additional parameters can be introduced to yield more detailed descriptions, but the gain must be put off against the increased complexity. Usually a small number of parameters (≤ 5) is sufficient to capture the dominating factors. To avoid confusion we will refer to the common parameters as *resource* parameters.

Exposing an operation space amounts to translating the internal modes of operation into the relevant resource parameters. The translation, denoted by T, usually requires some processing to determine the effects of switching an internal parameter from one mode to another. For example, changing the quantization factor in the video encoder will change the throughput (and distortion) of the compressed stream, but there is no simple relation between quantization factor and throughput. When changes in internal parameter settings result in rather large and discrete steps in the resource parameter space, exposing the operation space by means of an expression is not an option. In such cases we simply tabulate the operation space, where each entry is a parameter tuple. To master the complexity the number of entries in the exposed operation space (i.e., the length of the table) should be limited, which may require pruning and sub sampling of the internal options.

The task of the optimizer is to take the client's operation space (i.e., different workloads) and match it with the server's operation space (i.e., performance levels) and select the best combination. What is considered the best choice is dependent on the system at hand. Typical quality measures from the mobile computing domain are energy consumption, throughput, and CPU cycles. The optimization criteria may even change over time, for example, the importance of energy consumption may be related to the available amount of battery energy; full or nearly empty batteries are different situations requiring different trade-offs.

It is instructive to see how the various approaches discussed in the related work section fit into the operation-space optimization scheme of Figure 1. They can be classified according to who (client, server, or both) is exposing its operation space and where (in the client or server) the optimizer is located. With call admission (Figure 2) the client can issue just a single request and the server grants or denies it. The optimizer, located in the server, has a simple task: does the service level exceed the request from the client? If so, the request is granted, otherwise it is rejected.

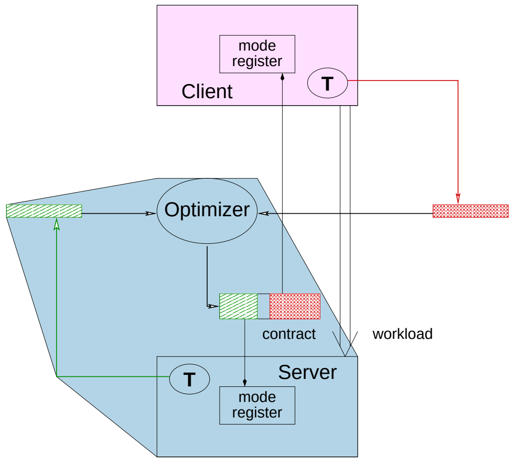

Figure 2. Call admission.

In the case of benefit functions (Figure 3) the client exports its operation space, so the server can dynamically control the operation mode of the client. As with call admission, the server is still offering only one performance level, but the optimizer is more complicated since it may dynamically switch the client's mode of operation to adapt to changes. Finally, consider the case where the server provides performance feedback to the client (Figure 4). The optimizer is now located with the client who uses the current service level to determine the best possible mode.

At first glance, it appears that both the benefit functions and the performance feedback approach can be upgraded to full multi-level optimizations by simply modifying the server to expose its internal operation space instead of only providing its best-effort mode. Unfortunately, it is not that simple, because of the following two reasons:

1. The purpose of multi-level optimizations is to make cost-based trade-offs, which often involve multiple resources (e.g., radio vs. CPU). This rules out benefit

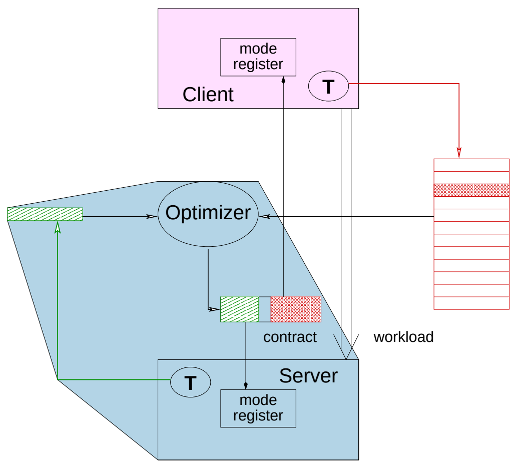

Figure 3. Benefit functions.

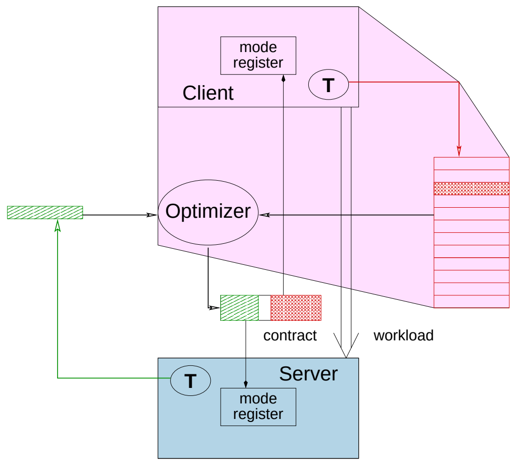

Figure 4. Performance feedback.

functions, since adding resources would result in a situation where the client specifies its alternatives to multiple *independent* servers, each wanting to control the client's mode of operation. This clearly does not work, unless the different servers are redesigned to collaborate and make a joint decision. Performance feedback has little difficulty in handling multiple resources, since a single optimizer (part of the client) allows for straightforward combining of the services involved. For each client mode the costs of the individual services required are accumulated; the mode with the lowest overall costs is selected.

2. Systems are frequently structured in a hierarchy of more than two layers. This requires a more complicated control structure than supported by the two-level performance feedback approach: a middle layer must act both as a client (requiring service from resources at the layer below) and as a server (offering services to the layer above). This changes the role of the optimizer significantly. Instead of selecting the best mode from the server below, it must provide a range of alternatives to the client above. The final decision about what alternative to use is now taken at layers higher up in the hierarchy.

The effects of applying multi-level optimizations in a hierarchical context are shown in Figure 5. The optimizer composes a range of alternatives ($\boldsymbol{w}_k$) out of the two operation spaces (i.e., tables $\boldsymbol{u}_{k-1}$ and $\boldsymbol{y}_k$). In effect, each alternative reflects the added value of the client to the basic service and includes all (server+client) costs. To prevent state explosion the optimizer may not simply include all possible combinations, $(\boldsymbol{u}_{k-1} \times \boldsymbol{y}_k)$, but must filter out irrelevant entries. There is a risk of filtering too much, especially since there is no (direct) information about what the upper layer optimizers are after, but preliminary experience shows that even simple thresholding rarely misses opportunities (see Section 5).

After filtering the relevant contracts, the optimizer passes the resulting set on to the client. The client uses this set to expose its operation space to the next level up. When the client is at the top of a (sub) hierarchy, the set is used to effectuate a contract. A client is at the top of a sub hierarchy if all contracts at its server interface are settled. A given contract at the server-side interface matches an entry in the server-side operation-space exposure $\boldsymbol{u}_k$, which in turn was constructed upon a single entry from the range of possible contracts $\boldsymbol{w}_k$ at the client-side interface. The server-side contract (indirectly) determines the client-side contract.

Operation-space exposures propagate up the hierarchy, whereas contract establishments propagate down the hierarchy. The exposures and contracts are specified in resource parameters, which may be different from the internal parameters used inside a layer. The conversion is taken care of by transformations Ts and Tc shown in Figure 5. These transformations comply with common state space description from control theory [10]. Let $\boldsymbol{q}_k$ be the internal state of a layer k, $\boldsymbol{u}_k$ and $\boldsymbol{y}_k$ be the operation-space exposures at the server and client-side, respectively, and $\boldsymbol{w}_k$ be the possible contracts as in Figure 5. Note that $\boldsymbol{q}_k$, $\boldsymbol{u}_k$, $\boldsymbol{y}_k$, and $\boldsymbol{w}_k$ are lists of parameter tuples. Then we have

$$\begin{array}{rcll} \boldsymbol{u}_k & = & f_k(\boldsymbol{q}_k, g_k^{-1}(\boldsymbol{w}_k)) & \text{(Ts)} \\ \boldsymbol{y}_k & = & g_k(\boldsymbol{q}_k) & \text{(Tc)} \end{array} \qquad (1)$$

The transformations Ts and Tc implement functions $f_k()$ and $g_k()$ that map input and internal parameters to output parameters. Locating Ts and Tc in the same layer circumvents the necessity to share information between Ts and Tc to implement $g_k()$ and $g_k^{-1}()$. In general, $g_k^{-1}()$ need not exist, but $g_k^{-1}(\boldsymbol{w}_k)$ must exist. The inverse values can be obtained through a look-up table recording the actions of $g_k()$.

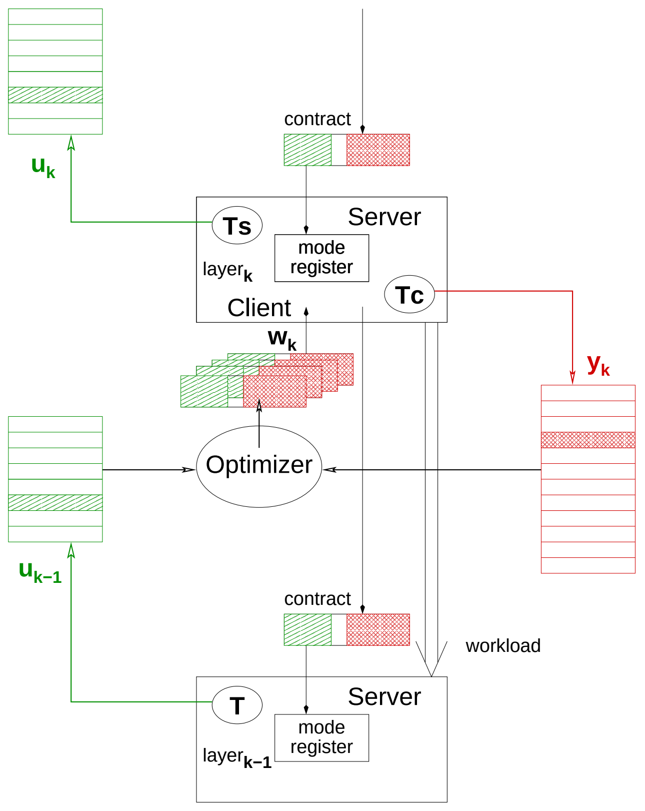

Figure 5. Hierarchical optimization.

4. Adaptive resource contracts

Within the Ubiquitous Communications (Ubicom) program at Delft University of Technology [7] we have developed a generic approach for adaptive systems to be used in resource-scarce environments. The driving target of Ubicom is a wearable augmented-reality terminal with a wireless connection to information and services on the backbone network. The experimental mobile Ubicom platform, currently under construction, is structured as a hierarchical system; each layer is designed to be capable of adapting to changes in the environment (transmission quality, application load, etc.) by selecting from a set of possible modes of operation. Layers interact through a generic concept, named Adaptive Resource Contracts (ARC), based on the multi-level optimization discussed above.

As the name suggests, the basic unit of interaction between two layers is the *contract* specifying the mutual agreements to which both client and server are committed. In a wireless environment no hard guarantees can be given, so contracts can and will be violated, but this must be done by explicitly signalling the other layer involved in the contract. A contract consists of a set of constraints on the resource parameters, which characterizes the performance of the lower layer (server) and the induced workload of the upper layer (client). Each constraint is a pair of values specifying the allowed range of operation for the associated resource parameter. It is important that a range is specified, not a single point, so the server has room to adapt without involving the client. Small changes will be handled locally; large changes (violating a contract) will propagate to the next layer.

An efficient implementation of the hierarchical optimization (Figure 5) minimizes the communication overhead. The optimizer observes $\boldsymbol{u}_{k-1}$ and $\boldsymbol{y}_k$, and generates $\boldsymbol{w}_k$ with only significant entries, in effect discarding unattractive combinations. So why bother to communicate irrelevant entries in $\boldsymbol{u}_k$ and $\boldsymbol{y}_k$ in the first place? A straightforward solution is to combine the optimizer either with the client or server component. Below we argue that combining the optimizer with the client is a good choice and we derive a method to generate and control the resulting $\boldsymbol{u}_k$.

Recall that contract establishment propagates down the hierarchy. From the client's point of view, therefore, it is efficient to be able to control the volume and range of alternative contracts $\boldsymbol{w}_k$ generated by the optimizer (and depending on $\boldsymbol{u}_{k-1}$). We can focus the generation of alternatives by constraining the resource parameters; these constraints can be specified by partially filling a contract. A top-level client can specify such a target set of partial contracts ($\boldsymbol{v}_k$). Given $\boldsymbol{v}_k$ a hierarchical component generates a set of partial contracts ($\boldsymbol{v}_{k-1}$) of its own, which when acknowledged ($\boldsymbol{w}_k \approx \boldsymbol{v}_{k-1}$) yields an operation-space exposure $\boldsymbol{u}_k$ closely resembling $\boldsymbol{v}_k$. In the ideal case, $\boldsymbol{u}_k$ resembles $\boldsymbol{v}_k$ and the optimizer reduces to identity. A bottom level server can directly generate an operation-space exposure $\boldsymbol{u}_{k-1}$ closely matching the requested (partial) $\boldsymbol{v}_{k-1}$. Obviously, locating the optimizer in the client allows for the necessary volume control of the operation-space exposures. In addition the optimizer can take advantage of the work performed to handle the partial contracts. We refer to these partial contracts as requests.

Although an individual contract captures commitments between two layers only, the process of drawing up contracts involves a complete hierarchy. It is a three-sweep negotiation process:

1. The topmost layer (L_T) initiates a *request* sweep by asking an offer from the layer below (L_{T-1}) about the current set of cost/quality levels that it supports. To focus the inquiry, the request may contain restrictions on the ranges of the resource parameters. Layer L_{T-1} interprets the request, considers its options, and passes a new request down to layer L_{T-2} to collect the information needed to answer the original request from above. This reformulation of requests continues downwards.
2. At the bottom level (L_0) the server has all basic information available and is able to respond directly to

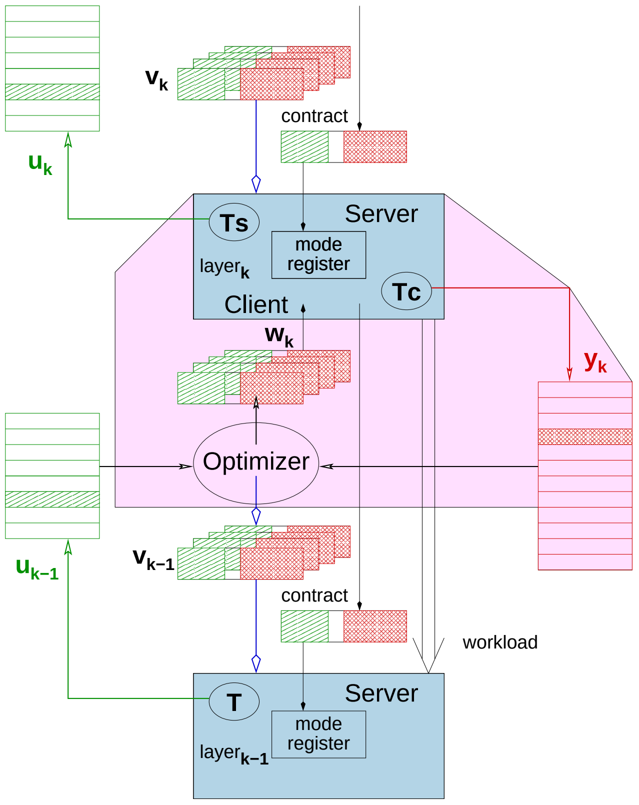

Figure 6. ARC generic interface.

the request. It replies with an *offer* listing all current cost/performance levels that fulfill the requirements stated in the request. This is the start of the second sweep, since the offer by L_0 is used by layer L_1 to construct its offer to layer L_2 above, etc.

3. Finally, when the offers reach the topmost level, a decision is made which option is best. This is laid down in the contract between layers L_T and L_{T-1}. Now layer L_{T-1} knows what it must do, and draws up the contract with layer L_{T-2}, etc. until finally the contract between layers L_1 and L_0 is established.

Figure 6 shows the generic ARC interface. In comparison to Figure 5, the optimizer is now part of the hierarchical layer-k component so that it can operate directly on the internal parameter space. The translations between internal and resource parameters (Ts and Tc) are reduced to thin client and server-interface wrappers. Another difference to Figure 5 is that additional information (request set $\boldsymbol{v}_k$) is provided to the server as to direct the search.

Note that at each layer multiple resources may be active. For example, at the bottom layer of the Ubicom hierarchy we distinguish the radio and CPU as two different resources. Therefore, when layer L_1 (channel encoding) receives a request it must query both the radio and CPU before it can reply with an offer. In general requests (and offers) propagating down (up) the hierarchy follow a DAG pattern.

The more specific the request, the smaller the set of options for the layers below. Nevertheless, even a detailed request may generate large numbers of alternatives. Therefore, a request will be accompanied by a number (N) stating that the client is interested in a offer containing at most N alternatives. When the server can offer more alternatives than requested, it must select the N most applicable ones. Different criteria may be used for pruning the intermediate results, but since the server is not fully aware of the client's objectives it probably is wise to return a large spread by some form of sub sampling. The client can get more detail by rephrasing the request, for example, by narrowing the ranges around one or more promising alternatives.

In adaptive systems like Ubicom the (external) conditions vary, so contracts frequently have to be renegotiated. Fortunately, it is not always needed to renegotiate each and every contract throughout the entire hierarchy. Minor changes can usually be handled inside a layer by adapting its operation mode. Moderate changes require mode adjustments in a few layers, which are established by applying the three-sweep process to a sub hierarchy. Only significant changes affect the entire hierarchy. Thus, it depends on the magnitude of the changes how many layers participate in the renegotiation process.

5. Ubicom case study

This section presents a case study from the Ubicom project, which involves a high-level model of a typical Ubicom application. The model is hierarchical with ARC interfaces between components. In line with the ARC concepts the model implements distributed control and local adaptations. The aim of our case study is to analyze ARC in a relevant setting. We address the issue of having a consistent concept for modeling system components. We study the effects of applying local optimization and adaptation routines with respect to overall system performance. Finally, we monitor the volume and density of respective operation-space exposures, which should be small for ARC to be effective.

The Ubicom case models a view point sharing application that offers mobile users the opportunity to look over the shoulder of other users walking around on the campus. We assume that users are interested in controlling power dissipation and video quality, on a per video stream basis. For example, one may choose to watch high quality video for a short while or poor quality video for a longer time. Power is considered the dominating factor in our system and is part of each ARC interface.

The case includes four layers (from top to bottom): the *video broker*, *source coder*, *channel coder*, and *transceiver*.

All layers require processing and induce a workload on the CPU being part of the mobile user's terminal. According to Figure 6 each layer in the application model has a client *and* a server-side interface. We have consistently modeled each layer with internal parameters, $\boldsymbol{q}_k$, a client-side workload generating function, $\boldsymbol{y}_k = g_k(\boldsymbol{q}_k)$, and a performance indication function, $\boldsymbol{u}_k = f_k(\boldsymbol{q}_k, g_k^{-1}(\boldsymbol{w}_k))$. To keep the model manageable, we have limited the number of resource parameters on the ARC interfaces to three; likewise the number of internal parameters controlling the mode of operation of each layer is also limited. Note that in practical systems, designers can use as many parameters as needed, but the number should be small to limit complexity. Selecting which parameters to include is a difficult task, but the overall component-structure usually guides the selection of functionality and associated parameters at each interface. Despite the simplifications in the Ubicom case the total set of options is large. Clients therefore use requests to control the volume and density of the operation-space exposures. Finally, the optimization routine in each layer discards irrelevant entries by considering the ratio of power versus some local notion of quality.

5.1. Concise system description

The system configuration with its ARC and internal parameters is outlined in Figure 7. To avoid clutter, the CPU and its interfaces to the four layered components are not shown. The energy consumed by the CPU, however, is accounted for in the power parameter exposed at each ARC interface. The plots at the left-hand side represent operation-space exposures of the corresponding components. The plots on the right-hand side present details of the $f_k()$ and $g_k()$ mappings. Since our goal is to experiment and analyze the ARC interface system we will be brief on the details of the applied models and parameter choices.

Transceiver. The transceiver makes an optimal choice for the number of sub carriers (N), a modulation scheme M, and the transmitted power (E_b/N_o). Since the transceiver is at the bottom layer, it estimates the noise level (N_o) of the physical channel and the distance (d) to the mobile station. At its server-side the transceiver exposes throughput, error probability (P_e), and power.

The plot on the right-hand side in Figure 7 shows the resulting error probability with a trade-off between using more transmit power or increased processing power (selecting N and M). The generating function is given by

$$P_e \propto \frac{1}{2}(N-1)\ \mathrm{Erfc}\left(\sqrt{\frac{3\ {}^2\log(M)}{2(M-1)}\frac{E_b}{N_o}\frac{1}{d^2}}\right)$$

where Erfc() is the complementary error function.

Given (N_o, d) and an estimate of the generated workload on the CPU, optimal choices can be made for the transmit power (E_b/N_o), the number of sub carriers, and the modulation scheme. The resulting operation-space exposure is on the left-hand side in Figure 7.

Channel coder. The channel coder applies (n, k) forward error coding; it sends k source bits in packets of size n. The server-side exposure of the channel coder is in terms of throughput, bit error rate (BER), and power. We apply a simple performance estimate of the so-called sphere packing bound. This is a theoretical best performance coding technique, to trade-off throughput and BER. The plot on the right-hand side demonstrates that for different (channel) error probabilities increasing the number of source bits (k) yields a decrease of the resulting error rate (BER). The operation-space exposure is on the left-hand side. A trade-off between applying a heavier coding scheme and requesting a better channel is included in the optimization function.

Source coder. The source coder applies a progressive coding technique; the more bits transferred correctly, the more variance is transmitted and, hence, the less distortion is observed. The internal parameters to optimize for are the encoded block length (L), the source characterization (M), and the (fixed) image resolution [12]. The source coder exposes distortion, frame rate, and power as performance metrics at its server-side operation space.

For block length L and source characterization M, the encoded variance is given by

$$\text{encoded variance} \propto \ln\left(\frac{L}{M}+1\right)$$

A normalized graphical representation is on the right-hand side in Figure 7. Typical values for M are: 2^{-16} for static still images, 2^{-8} for natural images, 2^{-3} for a video stream, and 1 for the notorious MTV video clip.

Broker. Finally, the broker sets a (fixed) maximum frame rate and exposes quality and power metrics to the user, who is at the top of the hierarchy. We assume a simple model where we consider a high frame-rate and low distortion the best possible quality, low frame-rate and high distortion are considered to be of poor quality. Intermediate values are linearly mapped, thus a low frame rate and low distortion sequence has the same quality as a high frame-rate, high distortion sequence.

The requests for the broker are ranges for quality and power dissipation. The resulting operation-space exposure is plotted in the top left corner of Figure 7.

5.2. Analysis

The result of layering the internal models and their operation-space exposures is that the user is presented with a concise view of the trade-offs for the complete Ubicom

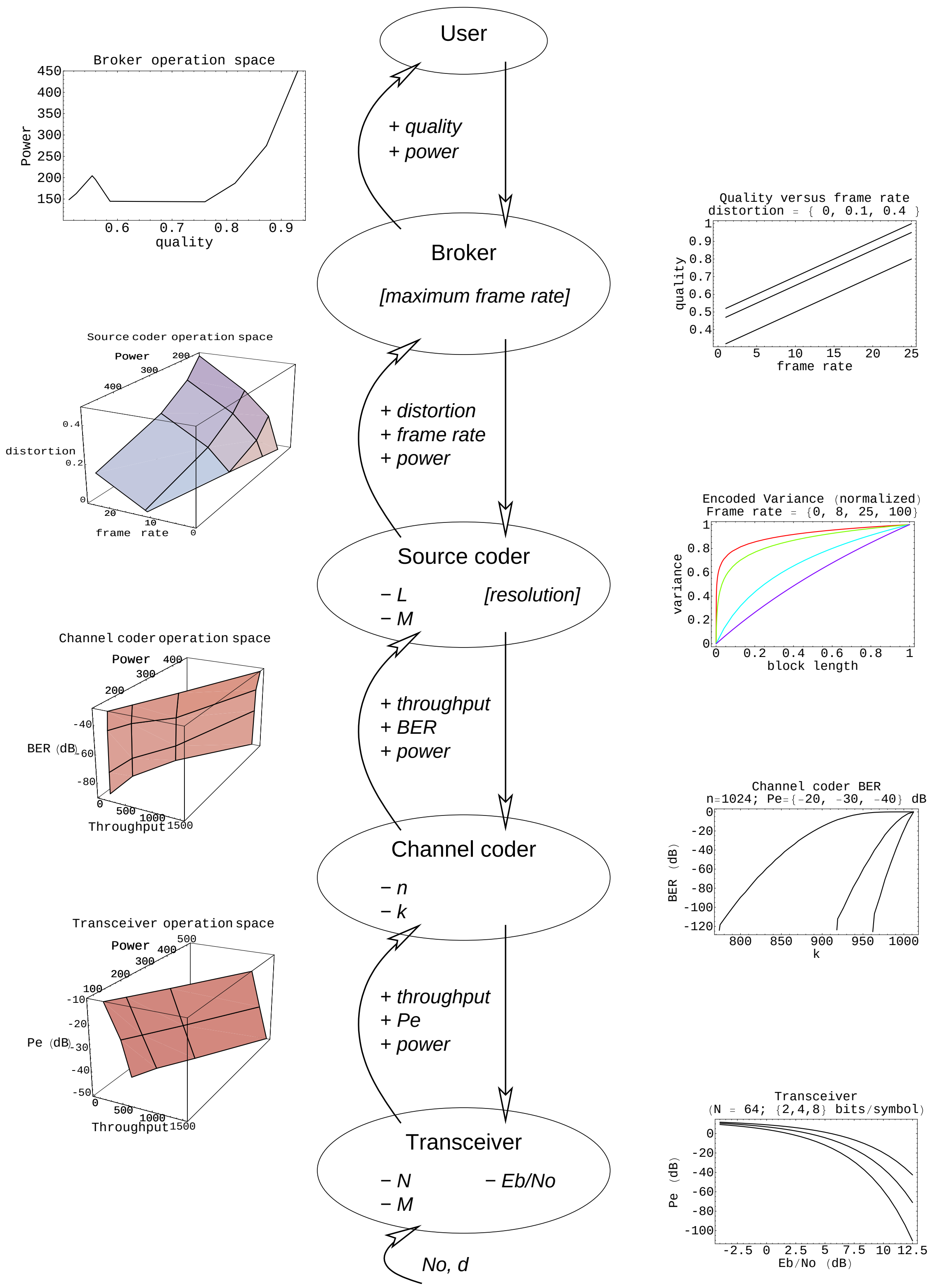

Figure 7. Ubicom case study: View Point Sharing. The operation spaces exported across the interfaces are shown on the left; the internal models are visualized on the right.

application. The ARC operation-space exposure of the broker, the top left-hand plot in Figure 7, gives the user the opportunity to trade-off power vs. quality. Each point in the plot corresponds to a chain of contracts and internal parameter settings throughout the system. For example, when the user chooses the contract with the lowest power/quality ratio, the subsequent contract-establishment sweep bypasses the channel coder $(n = k)$ and sets the transceiver to operate with just a few subcarriers $(N = 32)$. Choosing the highest quality, on the other hand, yields a channel coder with a low resulting bit error rate (BER = 10^{-7}) and a transceiver operating with many sub carriers ($N = 256$). This shows that a layered system with local optimizations is capable of supporting a wide range of control settings.

The ARC framework is quite efficient in that the total number of offered contracts throughout the system is just about 60 in this particular case study. To determine the quality of the top-level contracts we compared them with all possible offers that result from a straightforward enumeration over all internal component parameters. With 7 adjustable internal parameters and a moderate 6 different values per parameter this yields a global operation space of $6^7 \approx 300,000$ points. Figure 8 is a copy of the broker's operation space (top left-hand plot in Figure 7) augmented with the points from the global operation space. (For readability only one out of eight points is plotted). The plot shows that ARC does a good job: the vast majority of points in the global operation space denote unattractive settings that offer lower quality and/or dissipate more power than the alternatives provided by ARC. (Most points are located above and/or to the left of the ARC curve.) The plot also shows that ARC is not perfect and misses a few operation points with a better price/performance ratio. (Some points are located to the right and/or below the ARC curve.) This is a consequence of the huge – three orders of magnitude – reduction in the number of operational points communicated (60 vs. 300,000). More global optima can be identified by lowering the thresholds in the various optimizers, at the expense of increasing the number of contracts communicated across the ARC interfaces. It is the designer's task to strike the right balance between accuracy and overhead.

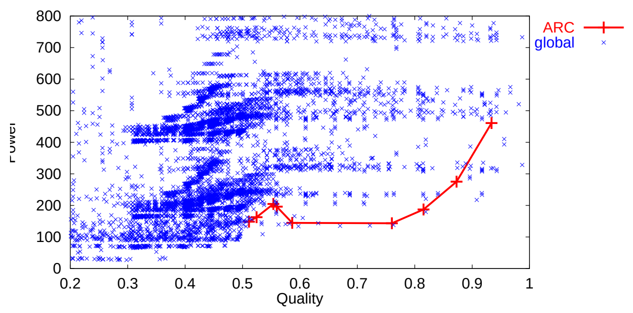

Figure 8. Global vs. ARC operation space.

6. Open issues

The experience with working out the Ubicom case study, which involved many discussions with our colleagues about specific components, showed that the ARC approach has an edge in designing hierarchical systems operating in resource-scarce environments. The ARC concept, however, is by no means finished; some open issues must still be addressed before ARC can be seamlessly applied in a working Ubicom prototype. We will now list the most important issues, and our initial views about how to approach them.

Policing. ARC assumes that all layers and components cooperate to achieve a common goal; there is no policing entity checking whether or not contracts are obeyed. An ill-behaving component, which consumes too much or delivers too little, will degrade system performance. Therefore, some form of policing behavior is needed, if only to catch errors in cooperative components.

Resource overloading. ARC does not specify what policy to use when a resource becomes overloaded. Should the server renegotiate with all clients that established a contract? If not then some specific contract must be selected for renegotiation, but which one: the last established contract, the biggest contract? For now, servers in an ARC context will have to establish an ad-hoc overload policy.

Incompatible modes. Servers often support multiple clients. A problem occurs when a server has some internal parameters that can only be switched on a per resource basis, not per client. For example, a transceiver may offer multiple modulation schemes, but only support one at a time. When offering all modes, two independent clients may indirectly select two different modulation schemes forcing the server to deny one contract. Simply denying the last contract leads to a static situation where the first client determines the modulation scheme during its lifetime.

Time. Parameter values in a contract are predicted averages with limited lifetime, for example, throughput of the wireless link for the coming second. From a system point of view it is important that both layers know the approximate lifetime of parameters. This way they can select an optimum mode of operation with respect to stability and agility of the entire system. Usually parameter lifetimes are fixed at design time. When lifetimes of parameter values are long, however, contracts may be violated occasionally. Not every violation will jeopardize the overall stability, therefore it may be beneficial to control the deviations from the contract using second order statistics (how much, how long) at run time. One solution is to use an *achievement* factor that specifies the fraction of time the service must be within the negotiated ranges [4].

Overhead. Applying ARC induces some overheads. In particular, request processing and operation-space optimization may be expensive when applied frequently (i.e., in lower layers). Reducing the frequency is not always an option since that affects the agility of the system. An alternative approach is to sacrifice generality and only allow for certain specific types of requests; bearer classes are an extreme case where all ranges are predefined.

7. Conclusions

Mobile systems operate in a resource-scarce environment and must adapt to changes in external conditions; all layers must make cost-based decisions about what mode of operation to use in response to performance information provided by neighboring layers. Since battery energy is the limiting factor, layers must coordinate their actions to deliver the best available quality for a given cost. Often trade-offs between multiple resources (e.g., CPU and radio) must be made requiring quality/cost information for various workloads. Traditional QoS negotiation mechanisms only provide best-effort information. Therefore we have designed a framework, named Adaptive Resource Contracts (ARC), that provides a generic concept for exchanging ranges of quality/cost settings.

With ARC QoS negotiations between two layers in a hierarchy (client and server) occur in three sweeps: 1) the client issues a request about the alternatives in a specific part of the parameter space, 2) the server responds with an offer listing all its options, and 3) the client determines the best combination of client and server mode and issues a contract. The contract consists of a set of ranges specifying within what region the server may operate. This provides the server some room to adapt to small changes in the environment. Large changes will cause the server to violate the contract, which will be reported back to the client, so it can take appropriate action and renegotiate a new contract.

We demonstrated the use of ARC by giving a small case study taken from the Ubicom project. The view point sharing application controls the quality and cost of transmitting camera images from one mobile user to the other, and consists of four layers with ARC interfaces. The study shows a consistent modeling of components with local optimization and adaptation routines. The overall performance of the system with ARC approaches the global optimum. ARC is also communication efficient: state explosions can be prevented using simple threshold optimization routines. From a system-design perspective a major advantage of ARC is that expert knowledge is captured in individual components. To gain insight into the identified open issues, such as overhead, agility, and stability, we are rewriting Ubicom software and incorporate ARC-style negotiations in the Ubicom test system.

Acknowledgements

This work was conducted within the Ubicom program (`www.ubicom.tudelft.nl`) and funded by the TU Delft, DIOC research program. We thank Chris van den Bos, Adrian Bohdanowicz, Slawomir Pietrzyk, and Arjen van der Schaaf for providing valuable input for the case study. We thank Dick Epema, Inald Lagendijk, Kees van Reeuwijk, and the anonymous reviewers for commenting on draft versions of this paper.

References

[1] M. Bechler, H. Ritter, and J. Schiller. Quality of service in mobile and wireless networks: The need for proactive and adaptive applications. In *Hawaii Int. Conf. on System Sciences (HICSS-33)*, Jan. 2000.

[2] S. Bhatti and G. Knight. Enabling QoS adaptation decisions for Internet applications. *Journal of Computer Networks*, 31(7):669–692, Mar. 1999.

[3] G. Bianchi, A. Campbell, and R.-F. Liao. On utility-fair adaptive service in wireless network. In *6th International Workshop on Quality of Services (IWQoS'98)*, May 1998.

[4] G. L. Bodic, J. Irvine, and J. Dunlop. Resource cost and QoS achievement in a contract-based resource manager for mobile communications systems. In *Proceedings of Eurocomm*, May 2000.

[5] S. Chatterjee, J. Sydir, B. Sabata, and T. Laurance. Modeling applications for adaptive QoS-based resource management. In *High Assurance System Engineering workshop (HASE'97)*, Aug. 1997.

[6] A. Goel, D. Steere, C. Pu, and J. Walpole. Adaptive resource management via modular feedback control. Technical Report CSE-99-03, Oregon Graduate Institute of Science and Technology, Jan. 1999.

[7] R. Lagendijk. The TU-Delft Research Program Ubiquitous Communications. In *Proceedings of the 21st Symposium on Information Theory in the Benelux*, May 2000.

[8] K. Lakshman and R. Yavatkar. Adaptive resource management for multimedia applications. In *High-Speed networking for multimedia applications*. Kluwer Academic Publishers, 1996.

[9] W. Lee and B. Sabata. Admission control and QoS negotiations for soft-real time applications. In *IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, June 1999.

[10] B. Li and K. Nahrstedt. A control theoretical model for quality of service adaptations. In *6th International Workshop on Quality of Services (IWQoS'98)*, May 1998.

[11] C. Uarrecoechea, A. T. Campbell, and L. Hauw. A survey of QoS architectures. In *7th International Workshop on Quality of Services (IWQoS'99)*, 1999.

[12] A. van der Schaaf and R. Lagendijk. Independence of source and channel coding for progressive image and video data in mobile communications. *Visual Communications and Image Processing (VCIP2000)*, June 2000.