Pervasive Adaptation for Mobile Computing

Tim Edmonds[1] Steve Hodges[2] Andy Hopper[1,2]

[1] *Laboratory for Communications Engineering*
Department of Engineering
University of Cambridge
Cambridge, UK
tme23@eng.cam.ac.uk

[2] *AT&T Laboratories Cambridge*
24a Trumpington St
Cambridge, UK
{seh,ah}@uk.research.att.com

Abstract

Mobile Computing platforms such as mobile phones, PDAs or wearable computers operate in a much more volatile and limited environment than their stationary counterparts. Such platforms are inherently resource poor and subject to highly changeable resource availability. Applications for Mobile Computing require adaptation for best performance under such variable conditions, to make best use of available resources without assuming the minimum set.

This paper details a framework developed by the authors for developing and deploying mobile applications. Current systems are able to notify an application to adapt but fail to say how. The application author must provide the actual adaptation mechanism. The authors aim to provide automatic adaptation to suitably constructed mobile applications. The adaptation mechanism is pervasive through application and system layers providing tight integration of adaptation both vertically (through an application) and horizontally (between applications).

1 Introduction

Mobile computing situates applications in an environment rich in resources and services but poor in resource availability or predictability. Applications that wish to make best use of available resources under such variable conditions without assuming the minimum set must adapt.

Typically, adaptation for mobile applications involves trade-offs: varying an application's quality of operation or even locality of operation to fit the current resource profile. The most common resource triggering adaptation in current applications is communications bandwidth as it is typically the most limiting and unpredictable resource in the system, particularly in the case of wireless communications systems. CPU capacity, memory availability and battery power are other adaptation criteria.

There exists many research application frameworks such as Odyssey[8, 6] or Hild's Mobile Application Framework[3] which provide adaptation *services* to mobile applications. However, these services, while providing migration capabilities and adaptation hints to applications, fail to instruct the application exactly how to adapt; the application author must themselves provide the mechanism by which the application can adapt.

In this paper, the authors present the DPROJ framework: an application construction kit and runtime system which enables automatic pervasive adaptation for mobile applications. The following sections describe the mobile operating environment, the applications expected to operate in the environment and presents a taxonomy of adaptation mechanisms. This is followed by a description of the DPROJ framework and applications.

2 Design Issues

Abstracting away from the implementation or environment details for distributed and mobile systems can greatly reduce the reliability and stability of an application[9] when the abstracting middleware is unable to account for factors such as disconnection or latency. Practical design of mobile applications must consider the special circumstances of the mobile computing environment both to cover unfortunate circumstances but also to best take advantage of available resources when they should become available.

Constructing individual, isolated, stationary applications is well understood; constructing applications that operate in a mostly connected distributed environment in which resources must be shared and may change offers considerably greater challenge. There are several key design principles

that must be considered when constructing mobile applications and systems:

Principle 2.1 (Diversity): *Mobile devices will be called upon to execute a large range of applications concurrently over a large range of conditions.*

Current devices assume that the user only requires to perform one operation at any one time and so mobile phones and PDAs often run simple operating systems that only run and present a single application at a time. Future applications may be more passive, pervasive and hence will require that multiple such applications be run concurrently on a single device or single resource set. Additionally:

Principle 2.2 (Adaptation): *To perform such operation efficiently, adaptation is required[5, 7].*

Connected mobile devices have access to resources beyond the device itself in the form of remote data and remote services. An application that uses such remote services is a *distributed* mobile application. Using remote resources can greatly enhance the capabilities of a mobile application but at the same time, the limitations of the mobile infrastructure means that the application must be able to cope with poor connectivity (up to and including disconnected operation), flagging resources (battery or CPU), and insecurity (mobile devices are much more susceptible to theft or security violations than their stationary counterparts)[8]. In particular, locality of data, locality of processing and locality of control must be resolved consistently.

Intermittent connectivity means that if data is to be accessible at all times, it must reside on the device itself. On the other hand, the mobile device is small, limited and vulnerable — only small data sets can reside on the device. The device may only manipulate restricted data sets and employ external facilities for more data. Additionally, maintaining data access and consistency (in the case of shared data) through disconnection become major issues.

Principle 2.3 (Data Locality): *Essential data must be held locally. Extended data sets may be held remotely.*

As for data, provision must be made for processing within the isolated case without restricting operation to exclude the advantages of connectivity to external facilities. However, some applications rely on a remote data source (eg: media streaming) in which case operation in a disconnected environment is meaningless.

Principle 2.4 (Process Locality): *A mobile device must be self-sufficient for the minimum of processing.*

Additionally, for adaptive distributed systems, care must be taken as to where the control of such systems originates. For mobile systems, for the reason given above for both data and processing, control must reside with the mobile device for operation through disconnection:

Principle 2.5 (Control Locality): *The mobile device must be in control of its resource usage and subsequent configuration.*

3 Adaptation Models

An adaptation model describes the structure and policies of an adaptive system. It defines the system level at which adaptation occurs and also at which level adaptation decisions are made. This section describes three broad divisions of possible models and argues that only a fully integrated application aware model can effectively provide adaptation services to a mobile system.

Transparent adaptation models perform adaptation at operating system or session levels in a manner transparent to, and concealed from, the applications. Operation of system services is modified to adapt to resource availability in a manner that is transparent to applications using that service.

An application-transparent model is beneficial in that existing applications may be run unmodified and new applications need not deal with adaptation issues. Central control of resources and adaptation avoids competition between applications and aids efficiency in resource use. However, such an adaptation model must operate without the application specific information that can help it make decisions concerning the best adaptation strategy. Working with unknown datatypes or the exact requirements of the applications, the adaptation mechanism must second-guess the nature of the application from its observable characteristics. This is the fundamental limitation of application-transparent adaptation: it is unable to efficiently handle situations outside its very limited initial remit.

In contrast to the centralised approach of the application-transparent model, application-specific adaptation places the task of adaptation solely with the application with no explicit system support.

While this allows each application to perform adaptation exactly to suit its needs, there is no cooperation or coordination between applications. Each application must compete with other applications for resources. In the absence of a system supporting prioritised resource access, an application that adapts, will most likely lose resources to one which greedily assumes maximum resource availability; applications that could share common data, do not necessarily do so. Application-specific adaptation can work well when only one application is dominating the resources at any one point in time. This may work well in situations where only

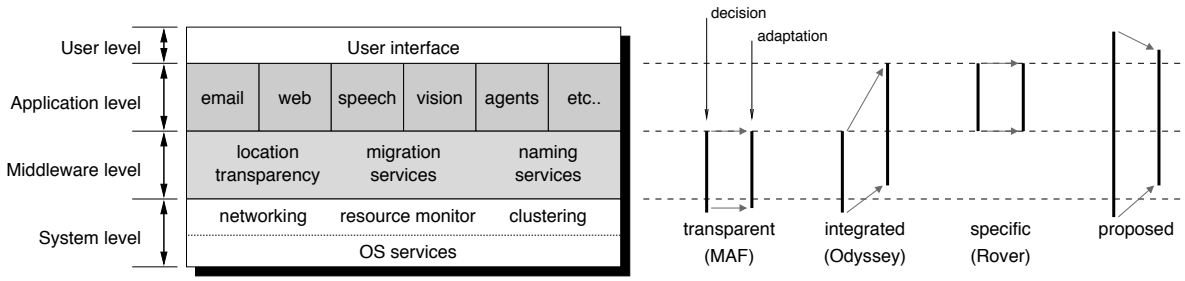

Figure 1. Decision positioning and adaptation areas of Application-Transparent, Application-Specific and Integrated adaptation models

adaptation	trans.	spec.	integ.
legacy app support	•	–	–
centralised resource	•	–	•
non-competitive	•	–	•
meaningful adaptation	–	•	•

Table 1. Adaptation Models

one application is running at any one time such as we might expect when using a small PDA or organiser.

The integrated adaptation model provides a composite of the two above models, combining the system management of resources with the application specific knowledge of exactly what it requires. This model allows multiple applications to run concurrently without competing — resources will be monitored and allocated centrally by the operating system and adaptation performed by the application, resulting in efficient use of resources and meaningful adaptation for the applications.

Integrated adaptation requires especially tight cooperation between the applications and the operating system. To achieve this, existing applications must be modified to run effectively with the system.

Table 1 summarises the pros and cons of the adaptation models described. Figure 1 illustrates the positioning of these models within the system stack. The transparent adaptation is epitomised by Hild's MAF[3] which provides transparent network monitoring and filesystem caching. As can be seen, it monitors at the system level and performs its adaptation decision at the system level. The application specific adaptation is fully contained in application layer. Such an application may use other system services (such as the Rover remote data object caching service or queued RPC[4]) but the resource monitoring and adaptation does not influence areas outside of the application proper. An integrate approach such as that taken by Odyssey[7] uses system monitoring to instruct application and system adaptation. System adaptation follows the transparent approach for some shared resources such as communications while Application specific adaptation is performed in by the application using hints from the system.

Clearly, for any multi-application mobile system, transparent or integrated adaptation is required. Transparent adaptation may be considered as a special case of integrated adaptation in which the application carries none of the adaptation burden. A hybrid system could provide integrated adaptation to suitably constructed applications while resorting to transparent adaptation for legacy applications.

4 DPROJ Framework

The proposed system extends the integrated approach to develop a monitoring and adaptation structure that spans all layers. Resource management and costing is centralised to provide efficient allocation and costing while resource utilisation is measured at a component level for individual applications. The adaptation capabilities of the system go further than providing hints to applications but provides actual adaptation services. Its pervasiveness extends to actual manipulation of application operation to achieve adaptation.

4.1 Aims

The DPROJ framework provides a truly pervasive integrated adaptation mechanism for mobile or distributed applications. The aim is to be able to simplify the development of adaptive mobile applications and enhance their functionality beyond that available if operation were constrained to the mobile device alone. The framework facilitates the construction of applications that can harness resources beyond the mobile device without making assumptions about or being limited by their varying availability.

By separating the adaptation concerns from the operational of an application, the developer can be free to concentrate on the application's functionality without having to enumerate all of the possible resource availability conditions and the configurations required to support them. The

application provides the functionality while the DPROJ runtime provides the adaptation.

The framework is integrated in a system structure that allows the execution and mutual adaptation of several such applications concurrently. As such, the framework provides a means for multi-host, multi-resource, multi-application synergistic adaptation.

4.2 Metric Unification

An important goal is to abstract over the specifics of any particular resource or any dimensionality to performance metrics. To achieve this requires the unification of resource costs to a single linear currency and the unification of performance measures to a single linear metric.

To achieve this, the framework core implements generic resource handling to deal with resource collation and demand processing facilitated by a unified resource cost metric. In this scheme, each resource is handled by a resource specific handler which performs resource monitoring and conversion of resource cost into a common metric. Following the market model, the chosen unified metric for the DPROJ framework is that of money. Further, implementing a real, hard currency, market arbitration for resources will ensure that resource price inflation will adequately control resource use. Real penalties ensure that the system will behave predictably in a competitive environment.

In this scheme, all resources are reduced to their financial cost in dollars (actually microdollars). This works well for communications usage (which especially for wireless services is a saleable commodity) and for remote processing services (which can be charge-per-process style services).

Other resources map less easily onto the currency metric. Resources such as local CPU time or battery power have a much more abstract inherent value which becomes much more dependent on the user. To combat this, user input is used to value these resources: a simple slider per abstract resource allows the user to set the relative value of that resource. Theoretically, it would be possible for a user agent to handle these details for the user.

For many applications, performance is a multi-dimensional quantity and the dimensions involved, or at least the measured dimensions involved, may not even be orthogonal. Despite this, any one particular application configuration can represent only one point in that performance space and the range of configurations available can represent only a series of points in that space. The *Unified Performance Metric* can be derived from the multi-dimensional performance metric using an appropriate transform.

Performance is an abstract quality difficult to grasp even within the context of a single application or functional element. It may be also have different meanings for different applications and aggregating total application and system performance becomes a fuzzy area. Reducing performance measures to a common dimensionless metric and baseline can simplify the process without overly compromising the utility of the measure. A streaming video player is a common and useful example of an application with multi-dimensional performance: such a player's performance is characterised by both the video frame rate and the video frame size. The overall perceived performance of the player is a combination of both of these measures: a high frame rate at a low size is just as poor as a low frame rate at large size. The unified performance metric would be some combined measure of the two of these reported by the application.

Devolution of performance metric unification to the application or functional element removes any application specific notions of performance from the framework. This simple unified performance metric allows combinatorial operators to determine the overall performance of related functional elements, the overall performance of unrelated applications executing simultaneously and also the relative performance benefits of alternate functional elements without needing to discriminate between differing notions of performance.

4.3 Design Concept

The framework achieves dynamic adaptation through two means: control of application fidelity and control of application locality[2]. This is achieved through the manipulation of the basic blocks of an application.

The basic application building block of the mobile system is the *functional element*. An element represents a unit of functionality of indeterminate scale and is defined as being some self contained executable entity of the application. Any application can be deconstructed to its basic elements through dataflow analysis; most applications are actually built by successive recombination of progressively more complicated elements. For adaptation within the DPROJ framework, an application is described as a hierarchic collection of functional elements. This hierarchy is achieved through the instantiation of *logical elements* which represent collections of one or more other blocks (either logical elements or functional elements). The three defined element classes are:

Basic Element — the minimum element of functionality. Optionally takes some data input, performs some computation and optionally emits some data.

Set Element — logical element containing a set of equivalent alternative elements. These sub-elements each perform the same function, but may differ in exact implementation, fidelity of implementation or locality of implementation. Only one sub-element is active at any

one time. The sub-elements may be logical or basic functional elements.

Container Element — logical element containing a collection of sub-elements that work together, composing the functionality of the container element. The container includes *wires* to connect the dataflow amongst the sub-elements and also to/from the container's external interface. All sub-elements are active concurrently.

Applications can be built from combinations of these three element classes. An application itself is just a Container with no data imports or exports. The application Container contains other elements which may themselves contain other elements and so on. In this manner, an application is constructed hierarchically from its functional elements.

The adaptation mechanism works through manipulation of the Set elements, dynamically selecting and activating the best sub-element for the given conditions.

4.3.1 CORBA Foundations

DPROJ focuses on providing a decision support for application adaptation, and hence uses existing software services to perform its object migration and relocation.

DPROJ's distribution system and relocation mechanism is built around a CORBA software-bus structure where each functional element is encapsulated in a CORBA object. Each element hence exposes at least three interfaces:

1. **CORBA Object** — provides standard CORBA functionality for object initialisation, destruction, remote access etc. If the CORBA ORB supports LifeCycle operations, object migration and state management methods may also be available.

2. **Element Manager** — provides element manipulation and monitoring methods including application structure analysis and resource usage monitoring.

3. **Element Specific** — provides import/export methods specific to each element type. These methods represent operation of the application proper while the above interfaces allow application control and adaptation.

Superficially, applications operating within the DPROJ framework appear as a number of processes executing on several hosts (Figure 2). The framework establishes on each host basic runtime support including resource monitors and application invocation primitives. Each application is then represented on each chosen host as a *code pool*, a collection of application functional elements. Application Proxies running on the mobile host perform the adaptation duties for the applications proper.

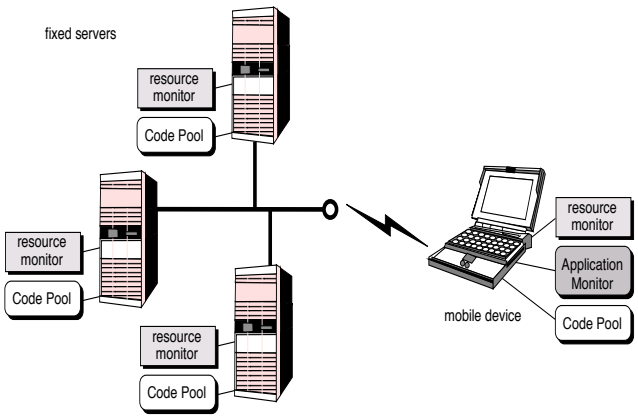

Figure 2. Framework component processes in the system.

4.4 Application Proxy

The Application Proxy performs the performance reporting, adaptation decision and reconfiguration for any logical element. Typically, one application proxy is instantiated to represent one top-level Container Element (an application) which then spawns further proxies to manage each sub-element, gradually building a proxy hierarchy parallel to that of the application. Application proxies representing application elements on different hosts may be merged together to form a single control hierarchy for a distributed application. A further Super Proxy may represent the system container and manage each of the Application Monitors as sub-elements, making the top-level configuration decisions.

Each proxy supports monitoring and reconfiguration for the element it represents. For monitoring, the proxy can produce a piecewise linear function of the cost-benefit curve of an element. This curve gives the expected cost of operation of the element for operation at a given performance point. A proxy representing a Basic functional element builds this curve from its monitoring of that elements resource consumption and claimed performance characteristics. A proxy representing a Set or Container Element forms its cost-benefit curve by combination of the curves of the sub-elements contained within it. In the case of Set Elements, the resulting curve is the minimum set of the contained curves while for a Container Element, the resulting curve is the sum of the curves contained.

With the single parameter cost-benefit curve, adaptation decisions reduce to selecting a single point on that curve and then propagating that decision to any sub-proxies. Deciding on the point can be based on a simple strategy such as 'select the highest performance that remains under this cost' or 'select the highest performance to cost ratio' or 'keep total costs for today under this amount.' This decision will re-

sult in a call to the top level proxy to operate at the decided performance point. As each sub-proxy receives this call, it decides how to act. A Set Element Proxy uses this to select which of its sub-elements offers the least cost for this performance point and activates it, deactivating any other sub-elements, and forwards the decision point to the active sub-element. A Container Element Proxy forwards this on to all sub-elements. A Basic Element Proxy forwards the set point directly to the Basic Element. This activation and deactivation of elements provides the core adaptation mechanism of the framework and occurs in a manner transparent to the application's operation.

Periodic invocation of the resource monitoring and reconfiguration process enables the application to dynamically track resource variations. Reducing the dimensions of the decision space through metric unification simplifies this reconfiguration process without sacrificing application notions of performance.

4.5 Applications

The following sections describe the implementation of some simple applications within the DPROJ framework and their operation in a controlled simulation environment. This environment uses *trace modulation* of communications resources[7] to produce a communications availability over time profile in a reproducible manner.

4.5.1 Media Streaming

Media streaming is a common test case for adaptive mobile systems. The aim here is to simulate an application that streams video or audio data from a remote host to be presented on the mobile device. Media streaming is a good test case especially for Quality of Service demonstrations where the quality or fidelity of a media stream can be varied to best fit conditions. In this example, we demonstrate how the locality of processing for media stream decoding can be varied to control resource usage. We follow this with a short discussion of how multi-fidelity operation might be included.

The Serial Test application is a chain of dummy data processing elements, each of which increases the magnitude of the dataflow along the chain. The two ends of the chain are anchored to the server and the client to enforce a split across two hosts. The Serial Test application can be thought of as a basic model of any media streaming type application; the data originates in compressed format and is decompressed through various stages resulting in increasing dataflow traffic before final presentation at the user's mobile terminal. The exact location of the decompression is unimportant but the origin of the data and the final presentation point of the data are.

Aim To check adaptation characteristics versus variations in resource cost (availability). *ie*: the ratio between server CPU cost (C_s), client CPU cost (C_c) and network cost (C_n). Initially, $C_s = 10$, $C_c = 100$ and $0 <= C_n <= 100$ is the independent variable.

Description The N-Block SerialTest application comprises a TestSrc data source and a TestSink data sink linked via N TestBlock data processing elements. The table below shows the basic characteristics of each element type. The TestSrc element generates t_0-byte data frames at 5 frames per second. Each TestBlock element exports a data frame twice the size of the one it imports. The TestSink element received the final frame and discards it. The TestSrc is constrained to operate solely on the server while the TestSink is constrained to the client. Each TestBlock may operate on either the server or the client. The location of each TestBlock is unconstrained by the location of any other. The CPU requirements of each element type is the same at 100TIPS.

		traffic		locality	
type	CPU	in	out	client	server
Source	100	0	$t_0 = 256$B	–	•
Block$_n$	100	t_n	$t_{n+1} = 2.t_n$	•	•
Sink	100	t_N	0	•	–

For the purposes of these tests, a 10-Block SerialTest application was used with $t_0 = 256$bytes at B_0 up to $t_n = 256$kB at breakpoint B_{10}. Operation of the application is enforced across two hosts; ideally, the application will only be partitioned at one point. The application may be partitioned at any of the breakpoints B_0 to B_{10}. Hence, the total cost function $C(b)$ for operation of the application may be devised in terms of the breakpoint B_b for $0 <= b <= 10$, CPU requirements per element $T = 10$ and the cost parameters C_s, C_c given previously:

$$cost = \text{server CPU} + \text{client CPU} + \text{network traffic} \quad (1)$$

$$C(b) = C_sT(b+1) + C_cT(11-b) + 5 \times \frac{t_0}{1024}C_n2^b \quad (2)$$

$$= 11100 - 900b + 1.25C_n2^b \quad (3)$$

Figure 3 shows total cost versus breakpoint over a range of communications costs. The framework will partition the application to minimise the cost, hence when:

$$\frac{dC(b)}{db} = -900 + 1.25C_n2^b = 0 \quad (4)$$

Observations Figure 4 shows the results of operation with $C_n = 28$. Equation 4 for $C_n = 28$ predicts a break at

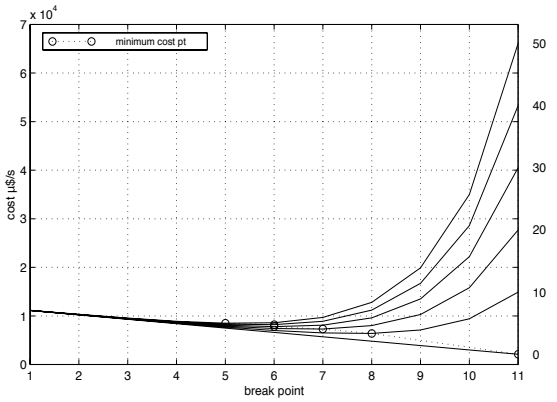

Figure 3. Serial Test cost versus breakpoint for different communications costs.

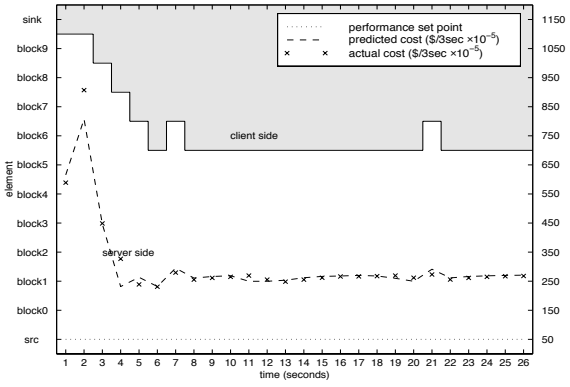

Figure 4. Serial Test locality pattern.

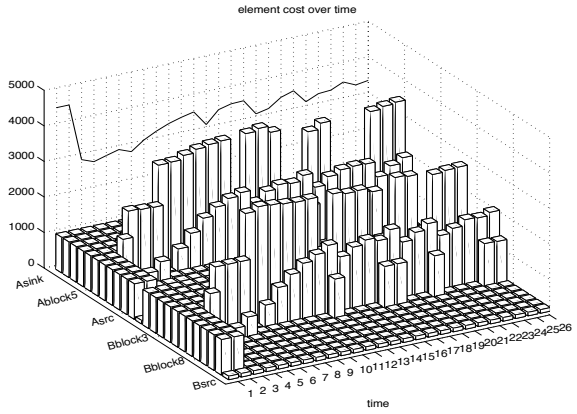

Figure 5. Parallel Test individual elemental costs.

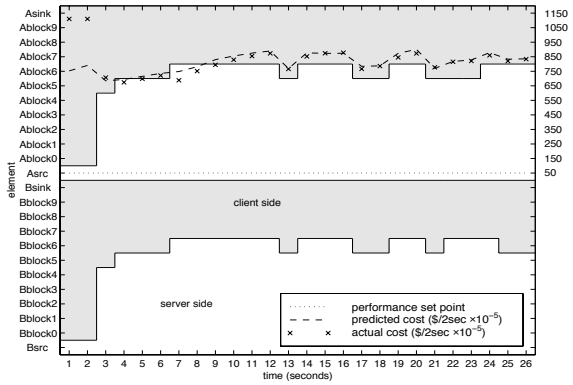

Figure 6. Parallel Test locality pattern.

point $b = 6$. In this case, the application begins partitioned between client and server with all blocks executing on the server except for the sink. This is a very high bandwidth configuration as the network traversal occurs after block9 with the 256kB frames. During execution, the application monitor calculates the application cost-benefit curve every second and performs any required reconfigurations. In this way, the application will gradually approach its steady minimum cost state. In this case, this takes approximately seven iterations. Some fluctuations occur due to variations in the measured communications use. The approach to the steady state is gradual as the framework is unable to estimate the possible cost-benefit curves of far removed configurations. This is due to the manner in which the application apportions cost of execution of a particular configuration; CPU and power costs are directly attributable to the element under execution, while communications costs are attributed equally to both the receiver and the sender. Hence, only an element that is directly communicating via the network incurs the communications costs which penalise it in the reconfiguration process.

4.5.2 Parallel Applications

The DPROJ framework performs adaptation across multiple applications, aiming to minimise the total cost of the system while maximising total performance. In this test, two identical instances of the above media streaming test were run in parallel. Figure 5 shows the individual element costs over time which clearly shows that the majority of the application execution cost derives from communications cost. The difference in processing cost is evident in the step between elements executing on the client and those on the server. Figure 6 shows the locality summary of the application elements. As expected, the framework has divided both application instances in the same manner, to minimise overall costs.

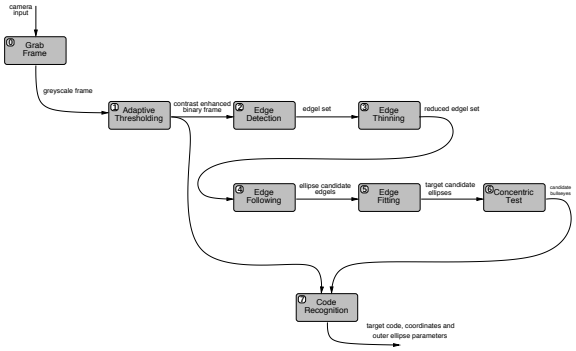

Figure 7. The TRIP target recognition pipeline.

4.5.3 MobileTRIP

The MobileTRIP application currently under development employs the DPROJ framework to construct and run a real time vision processing application[1] for a mobile user. MobileTRIP extracts the circular TRIP tags from a video stream, using the tag values to associate values with items within the view. MobileTRIP is aimed towards providing a low cost tagged approach to sentient computing. The application is designed to run on a small handheld computer which offloads much of the processing work onto remote servers. This application has both high processing and high bandwidth requirements and so is well suited for deployment using this framework. Figure 7 shows the initial decomposition of the application into functional elements. Each of these processing elements may be located either on the mobile device (client) or any one of several remote servers. There may exist more than one implementation of each block, providing the option to switch between high quality but expensive image processing and low quality but cheap processing.

5 Conclusion

This paper has presented the DPROJ framework which lays the groundwork for the development of adaptive mobile or distributed applications. The framework greatly simplifies the onus on the application author in developing applications that support integrated-adaptation at the cost of developing within a rigid structure. The framework is shown to successfully support multi-application synergistic adaptation even as a work under progress.

Further work on the framework includes further application development and more intensive testing and verification of the adaptation characteristics of the framework. Particularly, extension of the framework to accommodate integration with pay-per-use third party services as alternate functional elements. This would enable such services as speech recognition performed remotely by service providers to be used seamlessly depending on current budgets and connectivity.

Acknowledgements

The authors would like to thank AT&T Laboratories Cambridge for their support of this work and Diego Lopez de Ipina for his cooperation with the development of the MobileTRIP application.

References

[1] D. Lopez de Ipina. Trip for sentient computing. Technical report, Laboratory for Communications Engineering, Engineering Department, University of Cambridge, 1999.

[2] T.M. Edmonds, S.E. Hodges, and A. Hopper. An adaptive thin-client robot control architecture. In *Proceedings Real-Time Computing Systems and Applications*, Hong Kong, December 1999. IEEE.

[3] S. G. Hild. *Managing Mobile Connections*. PhD thesis, University of Cambridge, Sept 1997.

[4] A. D. Joseph, A. F. deLespinasse, J. A. Tauber, D. K. Gifford, and M. F. Kaashoek. Rover: A toolkit for mobile information access. In *Proceedings, Fifteenth Symposium on Operating System Principles*, Cambridge MA 02139 USA, December 1995. MIT Laboratory for Computer Science.

[5] R.H. Katz. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, 1(1):6–17, 1994.

[6] B. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*. ACM, Oct 1997.

[7] B. D. Noble. *Mobile Data Access*. PhD thesis, School of Computer Science, Carnegie Mellon University, May 1998.

[8] M. Satyanarayanan. Mobile information access. *IEEE Personal Communications*, pages 26–33, Feb 1996.

[9] J. Waldo, G Wyant, A Wollrath, and S Kendall. A note on distributed computing. Technical report, Sun Microsystems, 255 Garcia Avenue, Mountain View, CA 94043, USA, November 1994.