

A Platform Supporting Coordinated Adaptation in Mobile Systems

Christos Efstratiou¹, Adrian Friday¹, Nigel Davies^{1,2} and Keith Cheverst¹

¹*Computing Department
Lancaster University
Lancaster, LA1 4YR
United Kingdom*

{efstrati,adrian,nigel,kc}@comp.lancs.ac.uk

²*Department of Computer Science
University of Arizona
Tucson, Arizona 85721
USA
nigel@cs.arizona.edu*

Abstract

Mobile environments are highly dynamic, characterised by frequent and sudden changes in resource availability. As a consequence, adaptive mobile applications need to be capable of adapting their behaviour to ensure they continue to offer the best possible level of service to the user. Our experience of developing such applications has led us to believe that existing mobile middleware platforms fail to consider adaptive applications on a host as an ensemble of entities competing for the same resources; instead, focusing on the requirements of each application in isolation. A new approach is required which offers the mechanisms to support coordination of the adaptive behaviour of multiple applications in order to achieve a common goal. In this paper, we present a platform designed to meet this objective. Our platform is based on the notion of the definition of system-wide flexible adaptation policies written using a form of Kowalsky's event calculus, that may be adapted according to user needs. Moreover, we also believe that by using our approach it will soon be possible to identify and resolve conflicts caused by the need to adapt to multiple contextual triggers.

1. Introduction

Mobile environments are characterised by frequent and sudden change in both context and resource availability. As a consequence, mobile applications need to be capable of adapting to these changes to ensure they offer the best possible level of service to the user [10, 14].

Early adaptive systems have tended to focus on the specific limitations introduced by wireless connectivity [13, 21, 10]. However, more recently there has been an increasing interest in applications that adapt to a wider range of general environmental and contextual triggers, e.g. changes in a system's physical location or based on a set of personal

preferences. The GUIDE system [1, 2] for example, supplies users with information tailored to their current location and individual profile. The combination of resource driven adaptation, together with context-aware adaptation presents us with a new set of requirements for mobile applications; such applications must be capable of adapting to multiple adaptive stimuli.

The majority of current middleware platforms aiming to support adaptation tend to focus on limited sets of triggers, e.g. based on network Quality-of-Service. Such platforms would require the integration of separate adaptation mechanisms in order to achieve the necessary multi-triggered adaptation. In this paper we argue that this approach can cause *conflicting adaptation* or suboptimal operation of the system as a whole. We believe there is thus a requirement for a platform that supports multiple adaptation triggers, allowing for resolution of conflicts and system-wide behavioural optimisation.

In this paper we present a new middleware platform that provides such support for coordinated adaptation triggered by multiple adaptive and contextual attributes. Our platform is based on an event calculus based policy specification language [8], which allows the coordination and arbitration of adaptive actions on a system-wide level. More specifically, section 2 illustrates the shortcomings of existing adaptive approaches using a number of example scenarios. An analysis of existing adaptive systems is provided in section 3 which feed into a set of general architectural requirements for future mobile support platforms. The design of our platform is detailed in section 4 and section 5 gives a description of our current implementation status. Finally, section 7 contains our concluding remarks.

2. Identifying the Problem Domain

Mobile systems need to be capable of adapting to a wide range of system and environmental attributes, such as network bandwidth, location, power and so on. In general, cur-

rent adaptive systems provide limited support for adaptive applications by notifying them when certain “interesting” changes in these attributes occur. For example, if the bandwidth falls below some specified minimum threshold, it is then the responsibility of the application to adapt in an appropriate way. As we highlight in the following sections, such approaches can be shown to lead to inefficient overall behaviour due to the lack of coordination between the adaptation policies of multiple simultaneously executing applications. Furthermore, these approaches do not allow sufficient control over the implications of having multiple, and possibly conflicting, attributes triggering adaptation. In the following sections we present a set of scenarios illustrating these problems.

2.1. Scenarios

Coordinated application adaptation for power management. In this scenario we consider the case where a user is running several applications, each incorporating a periodic ‘auto-save’ feature (which, as we shall see, has an associated effect on the power consumption of the system).

Existing power management systems e.g. the ACPI [12] model, enable the switching of hardware into a low power mode when not in use. For savings in power consumption to be made, applications should attempt to keep the hardware resources in an idle state for as long as possible so that the time spent in low power mode is maximised. In the case of the ‘auto-save’ facility, the level of power optimisation is proportional to the time intervals of consequent hard-disk accesses. If each application periodically checkpoints its state in isolation, then the disk will be accessed arbitrarily. In contrast, if applications are able to coordinate their auto-save processes such that the disk writes are clustered, longer periods of inactivity will result, allowing the disk to remain spun down for longer. Such an approach would clearly be more power efficient.

Conflicting adaptation. In this scenario we illustrate an example of potentially conflicting adaptation to network resources. Consider a mobile system with limited available power resources running several adaptive network-based applications. When the battery level falls below some specified threshold, one of the applications is triggered to reduce its power consumption by reducing utilisation of the network interface. However, as the first application adapts and relinquishes some of its share of the overall bandwidth, this action may be interpreted independently by the other applications as an *increase* in available network bandwidth and may consequently increase their use of the network. In this case the request to utilise available bandwidth is *in direct conflict* with the initial goal of reducing power consumption.

Sharing demand for network bandwidth. In this scenario, we consider the case of two applications; an adaptive web browser and an adaptive video stream player, competing for the same network resources. Following a drop in available bandwidth the two applications could react in a number of possible ways:

1. The web browser could stop downloading in order to dedicate its portion of bandwidth to the other application.
2. Both applications could adapt and share the available bandwidth equally.
3. The video stream viewer could adapt by reducing its bandwidth requirements, e.g. by reducing its frame rate, in order to allow the web browser to increase its share of the available bandwidth (an important download might be taking place.)

The reaction that would be most appropriate clearly depends on attributes such as the total network bandwidth, but also crucially, the user’s requirements. In order for the two applications to adapt in a coordinated manner there is a basic requirement for *system-wide adaptation policies*. Such policies would be responsible for coordinating the behaviour of adaptive applications considering both the applications’ interdependencies and user specified requirements, such as the priority to be given to individual applications. Without such policies, coordinated adaptation between applications is difficult as each application is only capable of independent action.

A more detailed discussion of adaptation scenarios and the associated potential pitfalls and resulting outcomes can be found in [6, 7].

2.2. Analysis

The scenarios we have identified above illustrate a range of potential issues that are not addressed in current adaptive systems and middleware platforms. Moreover, based on our experience in this area we believe that these are not exceptional cases, but general problems inherent in considering adaptation on a per application basis.

Examining current adaptive architectures has allowed us to develop a framework for analysing the architectural model of such systems. The framework comprises two layers; the upper represents the application, the lower the adaptation support platform. Between these two layers we can identify four distinct flows of control and information (Fig. 1).

Flow A: Represents the requirements set by the application for resources or attributes supported by the underlying infrastructure. In the case of network adap-

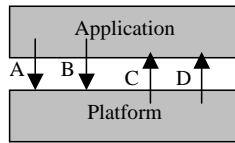


Figure 1. Directed flows between applications and platform.

tation this flow would represent the application’s network QoS requirements.

Flow B: Represents the ability of the application to control the functionality of the underlying infrastructure. For example, in the case of accessing a GPS device this flow could represent the frequency the device is polled for new position readings.

Flow C: Represents an information flow from the platform to the application. This provides the notification mechanism informing the application when its requirements cannot be met. Such notifications are intended to trigger the application to adapt.

Flow D: Represents the ability of the underlying platform to control the operation of the application. For instance, this flow could be a pro-active request from the system for the application to perform a specific adaptive behaviour, e.g. the application might be requested to reduce its demand for network bandwidth or disk usage.

Consideration of this framework enables a classification of current systems according to the types of flows supported. For example, network based adaptive systems such as BAYOU[25], Odyssey [21], MOST [10] and Rover [13] support flows A and C.

Context aware applications like GUIDE [1, 2], Stick-e Notes [22] and Cyberguide [18] are based on flow B, representing the access to the ‘context-sensors’, and flow C representing the information flowing from the sensors to the application.

Provision of a flow of control has not been widely exploited by mobile adaptive systems. Some distributed systems platforms, such as ISIS-META [19], do provide such a flow, but have not been adapted for mobile environments or context-aware application development. One more recent and notable exception is the Puppeteer system [5]; which provides support for mobile environments through control of non-adaptive applications by the platform. However, Puppeteer does not currently support application aware adaptation or system-wide coordinated action.

3. Architectural Requirements

The previous sections have identified the limitations of current approaches in supporting adaptive mobile applications. These approaches lack the appropriate support for enabling applications to adapt to multiple adaptive stimuli in an efficient and coordinated way. A new approach is therefore required which provides a common space for the coordinated, systemwide interaction between adaptive applications and the complete set of attributes that could be used to trigger adaptation.

In this section we provide a set of requirements that are used to develop an appropriate architecture for supporting adaptive mobile applications.

3.1. Sharing of adaptation attributes

The first key requirement of the architecture is to provide mechanism for sharing the adaptation attributes used by the system between applications. By sharing information about adaptive stimuli and application state we are able to construct applications that coordinate their adaptive strategies. Such support would also provide a platform for constructing applications that adapt to multiple adaptive stimuli.

3.2. Multiple adaptation attributes

Current research has already clearly identified the need to provide adaptive solutions based on the combination of different attributes [4, 9, 16]. It is important, however, that new attributes can be introduced dynamically into the system when they become significant, e.g. the cost of specific services to mobile users or information about human physiology for wearable computers. The fact that new contextual attributes for triggering adaptation can arise implies that:

1. The set of attributes that can trigger adaptation needs to be extensible.
2. The characteristics of all these attributes vary.

The first of these implications places a specific requirement on the extensibility of the architecture in order to allow for the incorporation of future, unpredictable attributes that will become important as new applications and systems are developed for mobile systems. The second implication limits the extent to which a single unified interface appropriate to all devices can be provided. One possible way in which we might mitigate this issue, is to provide a ‘meta-layer’ that will allow each device or application to specify the functionality of its interface itself, e.g. using XML.

3.3. Application Control and Coordination

A second requirement is the need to be able to control adaptation behaviour across *all components* in the system. As described earlier, one of the main limitations of current approaches is the fact that the applications themselves are responsible for triggering an adaptive mechanism when the underlying infrastructure notifies them about any changes. In order to support flexible and coordinated adaptation there is a requirement for triggering adaptation on a system-wide level. Given this approach, the decision about when and how an application should adapt is pushed into an external entity, with cross-application knowledge, while the adaptive behaviour is still a part of the application's characteristics.

3.4. Support for System-Wide Adaptation Policies

A further requirement for our system is to support the notion of system-wide adaptation policies. Such coordination would require the externalisation of policies to some coordinating entity or set of entities. We believe this requirement to be best supported through a dedicated human-readable *policy language* which should also satisfy the following requirements:

1. Provide a flexible syntax for defining rules that can take into account an arbitrary number of factors affecting each adaptation decision. The policy language *should not* be bound to a specific adaptation domain (e.g. network-based adaptation).
2. The policy specification and evaluation should incorporate as a central tenet the fact that the triggering of adaptation is inherently event based.
3. The specification of policy rules should allow the specification of fine grained temporal relationships between events. In most cases conflicts or instabilities in adaptive systems occur due to a lack of consideration of time or temporal dependencies between invoking adaptive mechanisms, e.g. waiting for the system to stabilise after changes are made or detected. The policy language should allow the fine tuning of adaptation mechanism to allow the resolution of such types of conflict.

3.5. User Awareness and Interaction

A final requirement that is often neglected in current adaptation platforms is the involvement *of the user* in the whole process of adaptation. Adaptation is usually an action that affects the Quality of Service that the user experiences. Moreover, it is often only the user that can determine the relative merits of a given tradeoff – for example, the system might attempt to optimise the running configuration of

applications to make best use of the available resources, yet in the same situation a user might choose to pause or stop an application that they deem insignificant to their current objectives. It is through promotion of awareness of the consequences of resource limitations and the actions of adaptive strategies, that the most profound adaptive effects can be achieved. Thus, there is a requirement that the user can introspect the adaptive behaviour of their system to determine and affect their desired outcomes. Furthermore, it should be possible to explicitly involve the user (as appropriate) when a pivotal adaptation decision has to be made.

The requirement to provide user awareness in adaptive systems can be further divided in to the following set of goals:

1. Explicit notification of the user when adaptation takes place. This notification should also provide information *about the reasons why* this adaptation took place. As we have stated, the provision of detailed explanation to the user could allow them to alter his/her behaviour in order to avoid results that can cause adaptation that degrades the quality of an important application. This characteristic of the user's behaviour could allow us to consider the user as part of the whole adaptation cycle. While there is research suggesting that it is possible to model the behaviour of the user with sufficient accuracy to modify the system's operation entirely automatically [11], we do not plan to support this in our platform, since we believe that it is non-trivial to correctly predict how users behave in all situations. Nevertheless, we acknowledge that user awareness can be of great benefit to the operation of the whole system.
2. Involvement of the user in the adaptation decisions. The definition of system wide adaptation policies is one aspect of user involvement. Though this mechanism is the most important tool for controlling the operation of all parts of an adaptive system, dynamic involvement of the user is necessary in order to fine-tune the behaviour of the system according to their specific needs at a given time. Dynamic interaction of the user could allow the injection of policies into the platform and/or alteration of existing ones. Such functionality becomes highly desirable when the platform faces a conflicting situation. There may for instance, be cases that the currently defined policies cannot resolve. In such a case user interaction might be the best strategy in order to achieve the appropriate adaptive behaviour.

In the following section we present our own architecture, designed in address the aforementioned requirements.

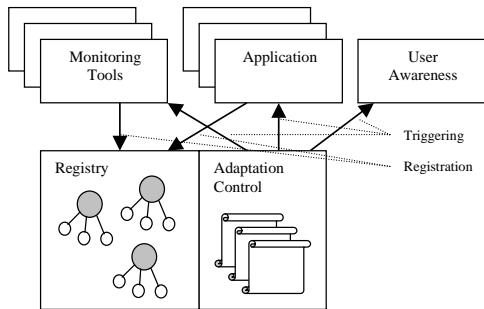


Figure 2. The overall architecture

4. Architecture

4.1. Structure of the Platform

In order to realise the requirements described in the previous section, we believe a platform should be developed in which adaptive mechanisms and policies are decoupled. In such a platform adaptive mechanisms can be exposed and externalised in order to enable control by independent entities.

Figure 2 illustrates the design of our platform. The architecture is dependent on two key attributes, namely:

1. the provision of a meta-layer holding the representation of the functionality offered by the applications.
2. the coordination of the behaviour of the applications according to the user defined policies.

More specifically, the operation of the platform is as follows. The platform allows applications and tools that monitor changes in the systems' environment to provide a description of their functionality, including the kinds of information that they can offer and actions that they can perform. These descriptions are collated by the platform's 'registry'. The 'adaptation control' is responsible for making decisions about the appropriate adaptive actions that each application should take. This component is driven by the policies exported by each application (defined by the application author) and augmented or redefined by the systems' user. The active participation of the user in the adaptation process is achieved by the 'user awareness module'. This module notifies the user of the adaptive decisions being made by the platform, and allows the dynamic resolution of potential conflicts.

4.2. Application Registration

Adaptive applications must register with the platform when they start up. The registration includes an XML description of the different adaptation modes implemented by

the application and the set of 'state variables' that can be accessed by the platform in order to identify the internal state or attributes of the application.

The application developer must implement a set of adaptation modes appropriate to the type of service that the application offers. This process is intrinsic to the implementation of *any* adaptive application, and must occur irrespective of the chosen support platform.

The first part of the application's registration notification presents the platform with XML describing the name of each adaptation mode and the set of parameters that can be passed into and out of each method by the platform.

The second part of the registration identifies the state variables that affect the current state of the application. Each state variable has a name and a 'basic type' (e.g. integer, string, etc.) The application is required to generate events informing the platform when one or more of its state variables have changed. For variables that generate continuous values, the platform can specify activation thresholds on each event (based on the policy specification).

The last part of the application's registration includes a set of default policy specifications. These policies are specified in accordance to the policy language described in section 4.4 and describe the default adaptive behaviour of the application as intended by the application developer. This behaviour however could be altered either by the active modification of the default policies or the introduction of new policies by the user.

An example of a typical application registration can be seen in figure 3. The design of the registration information has been highly influenced by the UPnP protocol [3].

4.3. Registry

The registry component operates as the repository of all of the adaptive applications operating within the system, aggregating details of their adaptive mechanisms and current execution status.

As described in the previous section, during start-up each application establishes a communication channel with the platform registry and issues its XML description (a library for communicating with the platform is used to reduced the burden on the application developer). Using this information, the registry constructs a stub object to manage interactions with the associated adaptive application, tracking the state of the adaptation modes and state variables. When a change occurs, the stub can forward events to the adaptation controller in order to decide whether a specific adaptive reaction is required.

An interesting aspect of the registry's design is the fact that applications and monitoring tools are handled in a similar manner. Although both are conceptually different entities, considering applications and tools as generalised enti-

```

<application>
  <name>WebBrowser</name>
  <uniqueId>1234</uniqueId>
  <methodList>
    <method>
      <name>SetBand</name>
      <attributeList>
        <attribute>
          <name>bandLimit</name>
          <relatedVariable>netBandwidth
            </relatedVariable>
        </attribute>
      </attributeList>
    </method>
  </methodList>
  <stateVariableList>
    <stateVariable>
      <name>netBandwidth</name>
    </stateVariable>
  </stateVariableList>
</application>

```

Figure 3. Sample XML description of an adaptive web browser.

ties that behave as both adaptive modules and information providers allows more flexibility for the implementation of coordinated adaptations. The reason for this is two-fold:

1. In most cases a monitoring tool represents a specific hardware device within the system, such as a wireless network interface. This means that a monitoring tool could also provide adaptation mechanisms relating to the specific device, e.g. a tool that monitors the activity of the wireless network interface could also provide a mechanism for putting that device into a low power consumption mode.
2. Applications can be the source of information that can trigger adaptation. The fact that one application enters a specific state can be used as the trigger for the adaptation of another application. A simple example might be when one application receives the user focus, this information can trigger other applications to reduce the use of resources because they are not as important for the user.

The second point also implies that the platform provides a mechanism for sharing of application context and state between applications on the system. This is clearly an important mechanism for addressing the shortfall in system-wide knowledge identified in section 2 (and could be used to solve the auto-save scenario in particular).

```

event lowBand :- NetworkInterface.availableBandwidth > 19200
event highBand :- NetworkInterface.avilableBandwidth <= 19200
fluent inLowBand {
  initiates(lowBand)
  terminates(highBand)
}
condition {
  initiates(lowBand, inLowBand, t1) and
  not clipped(t1, inLowBand, t2) and
  t2 = t1 + 30
}
action {
  WebBrowser.LowBand()
}

```

Figure 4. A sample policy rule.

4.4. Policies

A central concept within our architecture is the ability to specify policies using a policy description language. In order to satisfy the requirements specified in section 4.4 we have chosen to use an event-driven policy language derived from the event calculus logic programming formalism [15, 24].

The event calculus provides a framework in which it is possible to reason about the effects of events in an event-based system. More specifically, event calculus defines two distinct entities: events and fluents. An event is something that takes place at a specific point in time. A fluent can describe anything that has a time duration and can be described as a situation within the system that is initiated and terminated by an event. For example, a fluent could represent the condition “Battery is low”, initiated by the event reporting that the battery has dropped below a specific threshold and terminated by an event reporting that the battery level is above that threshold (e.g. during charging). The event calculus allows the specification of propositions using a set of predicates defined as follows:

- **Initiates**(e, f, t) : event e initiates fluent f at time t .
- **Terminates**(e, f, t) : event e terminates f at time t .
- **Happens**(e, t) : event e takes place at time t .
- **Holdsat**(f, t) : fluent f holds at time t .
- **Clipped**(t_1, f, t_2) : fluent f is terminated some time between t_1 and t_2 .
- **Declipped**(t_1, f, t_2) : fluent f is initiated some time between t_1 and t_2 .
- Logical expressions of the form $t_1 \vartheta t_2$ where t_1 and t_2 are time points or expressions of the same type and ϑ is a comparison operator of the set $\{<, <=, >, >=, =\}$.

Based on this framework we have defined an event calculus derived policy language [8]. The policy language allows specification of rules of the form:

```

event definition1 ... event definitionn
fluent definitions1 ... fluent definitionm
condition { condition body }
action { action1 ... actionk }

```

An event definition describes when a specific event is considered to have taken place. Each policy defined event is specified in accordance to the values of the application state variables known within the system. Each event is assigned a logical expression in terms of the values of application state variables. The event is considered to take place at the instant that this logical expression transits from false to true. In figure 4, the event *lowband* takes place at the time the available bandwidth of the network interface drops below 19.2Kbps.

The definition of a fluent provides a list of all the events that can initiate and terminate it. This definition is used by the evaluation mechanism implemented in the adaptation control module to derive the actual value for a fluent (whether it holds or does not hold at any given time).

The condition body is a logical expression that is constructed using the event calculus predicates. By combining the predicates we have described, a condition rule can specify when specific events take place and reason about the duration of the conditions identified by fluents. For example in 4, the condition is true only when the fluent *inLowBand* has been initiated and not been terminated (i.e. holds) for 30 seconds. Thus, the rule evaluates to true if the available bandwidth has dropped below the low bandwidth threshold and remained there for 30 seconds.

The action body is a list of adaptive method calls that should be made when the condition body evaluates to true.

4.5. Adaptation Control

As introduced in section 4.1, the adaptation controller is responsible for monitoring the state of the whole system and making decisions about which applications are required to adapt in order to overcome a problem or to achieve a designated goal.

The functionality of the adaptation control is driven by policies defined by the user and the default policies exported by the applications. The module evaluates the policy rules and triggers the appropriate application adaptation mechanisms. The evaluation of policy rules is performed incrementally as the values of the applications' state variables change. Whenever a state variable takes a value that corresponds to the occurrence of a policy event, the appropriate predicates are evaluated and the relating fluents take their

values (either holding or not holding). This process continues until all predicates within the rules body have been evaluated and the whole condition evaluated to true. At this point the adaptation control triggers the appropriate application methods as defined by the policy rules.

The fact that default adaptation policies exist outside the adaptive applications allows for their dynamic modification by the user. For example, a policy rule defining how a network application should respond to changes in network quality can be modified in order to take into consideration the conditions of other applications or triggers within the system.

An issue that requires further investigation is to what extent the system can automate policy modification in order to overcome conflicting conditions or the introduction of new policies. More details about the policy language specification and evaluation can be found at [8].

4.6. User Awareness Support

The user awareness module is implemented as a separate application that registers with the platform. The adaptation controller uses this tool in order to notify the user of adaptation decisions or to request assistance when conflicting situations are detected.

The fact that the user awareness module operates as stand-alone application allows the user to define the operations of the user awareness tool as part of a policy action. In the policy, the user has complete control over the type of message that will be presented by the awareness module and at what level of intrusion. Currently the level of intrusion can be set at:

- Low intrusion: animating an icon on the system tray, not actually giving information about the specific action, but letting the user know that an action has taken place.
- Medium intrusion: showing a balloon notification on the system tray informing the user about the action that has been taken.
- High intrusion: presenting a popup window with information about the adaptation action.

As the user awareness module is just application software an advanced user could write their own awareness system that is more closely integrated with their needs or working practices. For example, to use less intrusive interaction mechanisms, such as audio [20].

5. Prototype

In order to evaluate the architecture described, a prototype has been implemented. The prototype is implemented

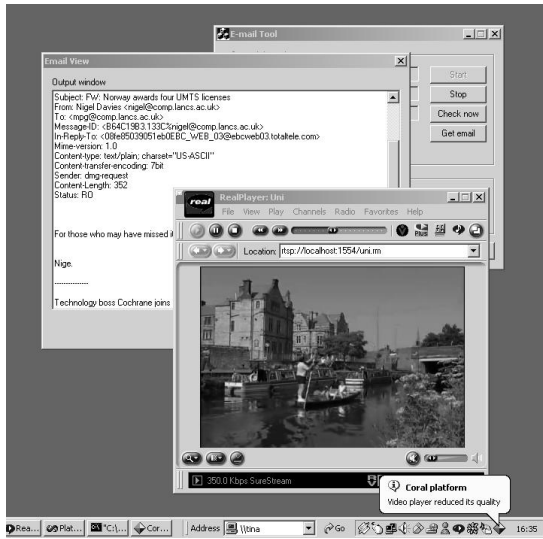


Figure 5. The platform in action.

in Microsoft Visual C++ and provides a full implementation of the registry and partial implementation of the adaptation control module. Currently the policy evaluation mechanism is capable of accepting policies specified according to the event calculus policy language presented in section 4.4. The evaluator is not yet capable of reconciling conflicts between the policies. Early indications are that the platform does allow us to achieve system-wide coordinated action and may provide the necessary underpinnings for identification of potential conflicts between adaptive applications. This is the key area of our future work.

In addition to the prototype platform, we have implemented a small set of adaptive applications that comply with our architecture in order to perform some experiments. In more detail:

- An adaptive video streaming tool has been developed based on the RealPlayer video client. This application allows the platform to request specific streaming adaptations including switching between different quality streams or the bit-rate that it is delivered.
- We have developed an adaptive e-mail client. This client provides status information about all its activities (polling the pop server for new e-mails, downloading e-mails, showing e-mails, etc.). These activities can become triggers for adaptation of other applications in the system, or can be coordinated by the platform according to the user policies.
- An adaptive web browser has been developed that can perform simple network adaptations. More specifically, the web browser utilises an intermediate proxy in order to perform network specific adaptation, e.g.

it can filter images from web pages in order to speed up the download of web pages and it can control the transfer rate of data over the network.

In order to be able to monitor the system resources two monitoring tools have been developed:

- A network monitor that can check the available bandwidth of the system's connection.
- A power monitor that reports the percentage of the battery power remaining.

A snapshot of the operation of the prototype is shown in figure 5. This figure illustrates the video player being triggered to reduce the use of bandwidth in order to allow the e-mail client to download a large e-mail within the timeliness requirements specified by the user.

6. Influential Work

In this section we briefly discuss related systems that have been directly influential on our work. ISIS/META [19] provides an infrastructure for reactive management of distributed systems. It introduces the concepts of sensors and actuators that provide information about the operation of applications and allow control by external entities. The functionality of these elements can be viewed as similar to the state variables and adaptation methods defined by our platform. The fact, however, that the META system targets the domain of reactive control imposes certain burdens in its applicability for coordinated adaptation of mobile applications. In particular the NPL language used by META for the specification of guarding commands is quite limiting since it does not allow the specification of time dependencies between events received by multiple applications.

The Puppeteer system supports the introduction of non-mobile aware applications into a mobile system. Puppeteer follows a similar approach to ours with externalisation of application mechanisms through XML descriptions. However, Puppeteer does not currently provide mechanisms for dealing with the issues of coordinated adaptation and conflict resolution of multiple adaptive applications; a key contribution of our platform.

In terms of policy systems, the event calculus policy language follows a common approach to the *PDL* language [17] used for policy based network management. *PDL* follows the same model of event-condition-action rules as the event calculus policy language. *PDL* however does not allow the explicit definition of time dependencies or elements with time duration, which we believe is essential for stable management of adaptive systems. Moreover, the event calculus provides a much simpler and more user friendly

vocabulary for the definition of policy rules, which is important in order to promote user involvement in policy definition. There are clearly other calculi with sufficient support for temporal relations (such as the Situation Calculus [23]) that may be candidates for the specification of similar policy rules as ours.

7. Conclusions

In this paper we have argued that flexible architecture supporting system-wide coordinated adaptation is required to appropriately support context-aware adaptive applications. We have identified the shortcomings of current approaches and have presented our own architecture that we believe meets the following set of requirements:

1. mechanisms to support coordination between multiple adaptive applications.
2. mechanisms to support the control of an application's adaptation.
3. support for user defined adaptation policies.
4. support for a common contextual space capable of retrieving, maintaining and sharing contextual information.

One of our goals for the future is to provide support for distributed adaptation coordination across and between communicating hosts. Such an approach would allow end-to-end policy negotiation for the adaptation of network applications. Another trend in our research is to enhance the user involvement in the adaptation process with different interaction methods and measure the effect of user awareness in the behaviour of adaptive systems.

References

- [1] K. Cheverst, N. Davies, K. Mitchell, and A. Friday. Experiences of developing and deploying a context-aware tourist guide: The GUIDE. In *Proceedings of the 6th ACM International Conference on Mobile Computing (MOBICOM) 2000*, Boston, 2000. ACM Press.
- [2] K. Cheverst, K. Mitchell, and N. Davies. Design of an object model for a context sensitive tourist GUIDE. *Computers and Graphics*, 23(6):883–891, Dec. 1999.
- [3] M. Corporation. Universal plug and play device architecture, version 1.0, June 2000. <http://www.upnp.org/>.
- [4] N. Davies, S. Wade, A. Friday, and G. Blair. L²imbo: a tuple space based platform for adaptive mobile applications. *ACM Mobile Networks and Applications (MONET): Special Issue on Protocols and Software Paradigms of Mobile Networks*, 3(2):143–156, 1998.
- [5] E. de Lara, D. S. Wallach, and W. Zwaenepoel. Puppeteer: Component-based adaptation for mobile computing. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, California, March 2001.
- [6] C. Efstratiou, K. Cheverst, N. Davies, and A. Friday. Architectural requirements for the effective support of adaptive mobile applications. Work in progress paper presented in *Middleware2000*, (USA:New York), April 2000.
- [7] C. Efstratiou, K. Cheverst, N. Davies, and A. Friday. An architecture for the support of adaptive context-aware applications. In *Proceedings of Mobile Data Management (MDM'01)*, Hong Kong, January 2001.
- [8] C. Efstratiou, A. Friday, N. Davies, and K. Cheverst. Utilising the event calculus for policy driven adaptation on mobile systems. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, Monterey, California, June 2002.
- [9] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications*, 1999.
- [10] A. Friday, N. Davies, G. Blair, and K. Cheverst. Developing adaptive applications: The MOST experience. *Journal of Integrated Computer-Aided Engineering*, 6(2):143–157, 1996.
- [11] J. M. III. *Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio*. PhD thesis, Royal Institute of Technology, Sweden, May 2000.
- [12] Intel/Microsoft/Toshiba. Advanced configuration and power interface specification, revision 1.0, 1999.
- [13] A. Joseph, J. Tauber, and F. Kaashoek. Mobile computing with the Rover toolkit. *IEEE Transactions on Computers: Special issue on Mobile Computing*, 43(3), 1997.
- [14] R. Katz. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, 1(1):6–17, 1994.
- [15] R. Kowalsky. Database updates in event calculus. *Journal of Logic Programming*, 12:121–146, 1992.
- [16] R. Kravets and P. Krishnan. Application-driven power management for mobile communications. In *Proceedings of the 4th ACM International Conference on Mobile Computing and Networking (MOBICOM '98)*, 1998.
- [17] J. Lobo, R. Bhatia, and S. Naqvi. A policy description language. In *Proceedings of AAAI*, Orlando, FL, July 1999.
- [18] S. Long, R. Kooper, G. Abowd, and C. Atkenson. Rapid prototyping of mobile context-aware applications: The Cyberguide case study. In *Proceedings of the 2nd ACM International Conference on Mobile Computing (MOBICOM)*, 1996.
- [19] K. Marzullo, R. Cooper, M. Wood, and K. Birman. Tools for distributed application management. *IEEE Computer*, 24(8), 1991.
- [20] E. D. Mynatt, M. Back, R. Want, M. Baer, and J. B. Ellis. Designing audio aura. In *Proceedings of CHI'98*, pages 566–573, Los Angeles, CA, April 1998.
- [21] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptation for mobility. In *Sixteen ACM Symposium on Operating Systems Principles*, pages 276–287, Saint Malo, France, Oct. 1997.

- [22] J. Pascoe. The Stick-e note architecture: Extending the interface beyond the user. In *Proceedings of the 1997 International Conference on Intelligent User Interfaces*, Short Papers, pages 261–264, 1997.
- [23] J. Pinto and R. Reiter. Temporal reasoning in logic programming: A case for the situation calculus. In *Proceedings of the International Conference on Logic Programming*, pages 203–221, 1993.
- [24] M. Shanahan. The event calculus explained. In M. J. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today, Vol. 1600 of LNCS*, pages 409–430. Springer, 1999.
- [25] D. Terry, M. Theimer, K. Petersen, and A. J. Demers. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, 1995.