Flexible and Efficient Platform Modeling For Distributed Interactive Systems

Xiao Feng Qiu, T.C. Nicholas Graham
School of Computing
Queen's University
Kingston, Canada, K7L 3N6
graham,qiu@cs.queensu.ca

ABSTRACT

Distributed interactive systems often rely on *platform information*, used for example when migrating a user interface to a small-screen device, or when opportunistically recruiting available peripherals. There has been to-date little work in platform modeling for distributed applications. In this paper, we demonstrate that distributed platform models are well supported by a publish and subscribe architecture accompanied by a rich filtering language. This approach allows organic construction of networks with no centralized locus of control, high scalability and fault-tolerance, and flexible customization to the needs of heterogeneous device types.

Categories and Subject Descriptors

H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—Computer-supported cooperative work.

General Terms

Human factors, performance, design

Keywords

Distributed interactive applications, platform model, groupware toolkits

1. INTRODUCTION

Modern interactive applications increasingly run in a networked setting where knowledge is required of what distributed resources are available to support the application's execution. For example, to dynamically connect a camera to a printer, the camera needs to know what printers are available in the vicinity. A user of a tabletop surface may wish to grab and use a keyboard from a nearby PC, requiring the tabletop to be able to discover the keyboard's existence. A PDA-based application might allow a user to take a photograph, and then search for a powerful computer where facial recognition can be performed. Or an application running on a Smartphone might discover that its battery is about to expire, and look for another device onto which it can migrate while offering minimal interruption to its user.

EICS'09, July 15–17, 2009, Pittsburgh, Pennsylvania, USA.

In each of these examples, the interactive application uses resources available over a network, and needs to be able to locate and evaluate these resources dynamically. This form of resource discovery can be characterized by three properties: the systems involve heterogeneous platforms, with fixed and mobile devices of very different capabilities; the connection of devices is ad hoc and based on current availability of resources; and the platform is distributed, with no central locus of control.

Most existing approaches to this problem are based on a flat network hierarchy characterized by direct communication between the user of a resource and all possible providers of that resource. This approach provides poor scalability, poor fault-tolerance, and slow network-based queries. In this paper, we present an alternative approach, combining a distributed publish and subscribe architecture with a powerful and flexible subscription filtering language. The approach has four key advantages over earlier techniques:

- It requires *no centralized locus of control*. Nodes can enter and leave the system at any time. The network can dynamically grow and contract, and can provide efficient service at arbitrarily large sizes.
- The system is *adaptable to the capabilities of its heterogeneous nodes*, allowing powerful, well-connected nodes to retain detailed platform information, while less powerful, poorly-connected nodes can restrict the information they retain.
- Platform queries are fast, as they are *resolved locally to the node*.
- The architecture provides inherent redundancy, allowing for *fault-tolerant operation*.

We have implemented and tested this approach in the *Fiia Platform Model*, part of a toolkit for developing adaptive groupware applications.

The paper is organized as follows. We first review other techniques used to model and represent platform information. We then present the *Fiia Platform Model*, and discuss its architecture and implementation. Finally, we present the results of performance evaluation showing how the system can adapt to the differing capabilities of heterogeneous devices.

2. RELATED WORK

Platform models are increasingly considered to be a key component of model-based development of interactive systems. For example, Calvary et al. explain how a platform model is a key part of a framework for automatic generation of user interfaces for diverse devices [2].

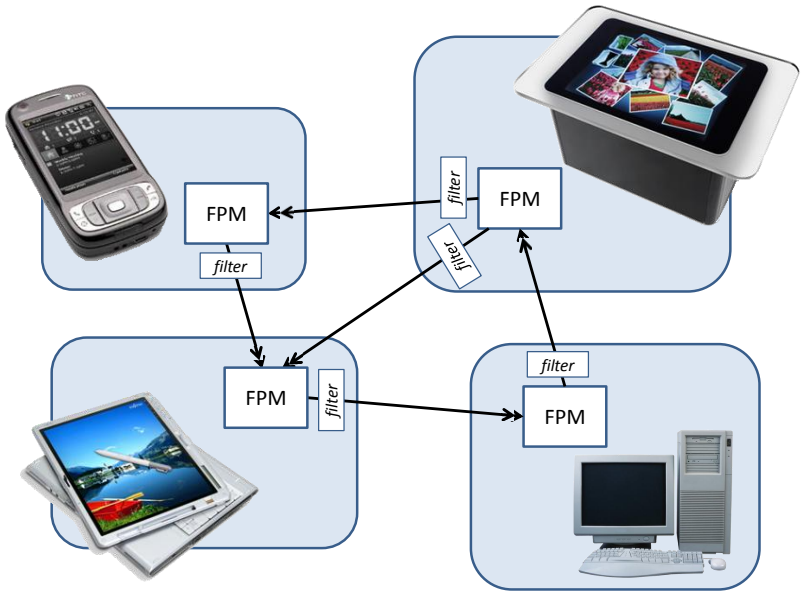

Figure 1: High-level FPM architecture

Platform models typically contain information about the device's capabilities. For example, ArtStudio describes devices in terms of their screen dimensions and resolution [14]. TERESA classifies platforms by category, annotating them in a task model as "desktop", "cellphone" or "PDA" [12]. More generally, Coutaz et al. propose that a platform model should capture *resources*, ranging over capabilities as diverse as processors, memory, I/O devices and sensors [2]. They distinguish between *elementary platforms* (a single device such as a laptop or Smartphone) and *clusters* (a set of elementary platforms).

Platforms are typically modeled as sets of attribute-value pairs. For example, Florins [6] uses UsiXML [11] to encode properties such as the platform's hardware, software and network connectivity. Similarly, the W3C's *User Agent Profile* (UAProf) protocol uses RDF to encode detailed descriptions about devices such as mobile phones, e.g., capturing voice capability, pixel resolution and text input capabilities [7]. A higher-level approach is embodied by the *recombinant computing* approach, where a small set of high-level interfaces is used to characterize mobile services such as projecting, printing or storing files [4].

Considerably less research has been performed in *distributed* platform modeling. *SNMP* (Simple Network Management Protocol) is a technology for making platform information available over a network [3]. SNMP is intended for system monitoring, and so concentrates on providing information such as current CPU load and memory availability. Each node provides an SNMP agent, which can be queried for platform information. SNMP's network topology is effectively flat, in that a node requiring platform information must individually query all other nodes of interest.

UPnP (Universal Plug and Play) is Microsoft's technology supporting dynamic connection of networked devices [10]. UPnP requires each device to provide a description of itself, thereby implementing a distributed platform model. UPnP is implemented via distributed publish and subscribe [5], where devices use subscription to express their interest in the state of other devices (e.g., in the length of the printer's queue), and updates are sent when this state changes. Similarly to SNMP, UPnP has a flat network architecture. UPnP is restricted to local area networks.

Moratus uses distributed shared memory to implement a *global context* for distributed interactive systems (which includes platform information) [9]. Similarly, *Gaia* supports interactive systems running within active spaces [13]. Gaia provides a CORBA-based set of distributed services, allowing transparent access to context and platform information.

Another interesting approach is *passive service discovery* [1], where local area networks are passively monitored for traffic that indicates the presence of services. Passive discovery can help discover, for example the presence of a printer on the network when another node accesses it. It would not help, for example, in determining the battery level of a PDA.

All of these approaches to implementing distributed platform models work well in their intended domain, but fail to address the requirements specified in the introduction. The flat network structure of SMTP and UPnP, the distributed shared memory model of Moratus and the distributed object model of Gaia do not scale to large groups. None of the approaches provide explicit mechanisms for dealing with the varying capabilities of heterogeneous devices.

3. FIIA PLATFORM MODEL

The *Fiia Platform Model* is based on a distributed publish and subscribe architecture [5]. In publish and subscribe, a *subscriber* specifies interest in data from a *publisher*; the publisher then sends data to the subscriber whenever it has news to report. In distributed publish and subscribe, these subscription relationships may cross network boundaries. As shown in figure 1, heterogeneous nodes with very different capabilities can use this architecture to share resource information.

Each node retains its own copy of platform information. Nodes may subscribe to data from arbitrary other nodes. Subscriptions may be filtered, allowing subscribers to control the quantity and frequency of information flow. This allows nodes with limited capabilities or poor (or expensive) network connections to trade off bandwidth used versus quantity and timeliness of updates.

The use of distributed publish and subscribe means that there is no requirement for a centralized locus of control. This allows new nodes to enter and leave the system without requiring access to a

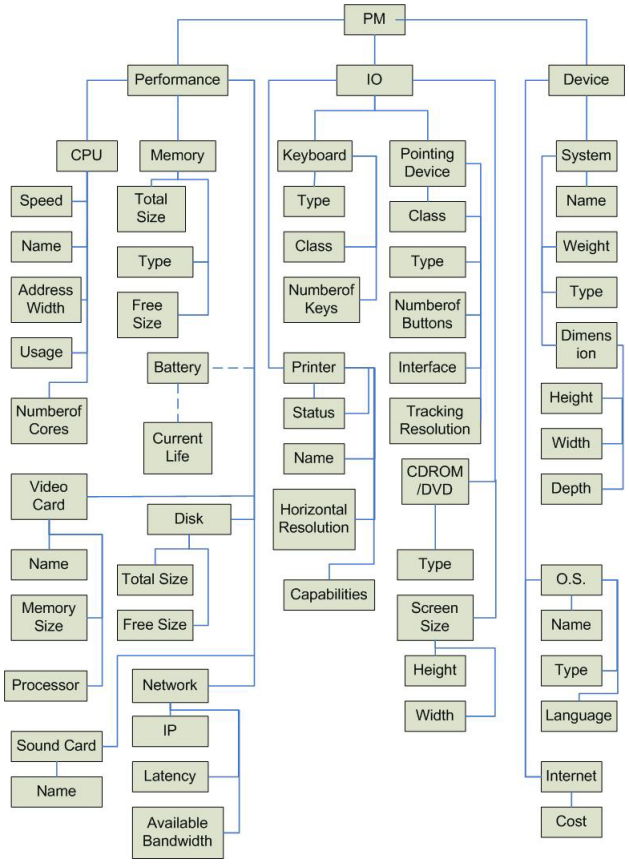

Figure 2: The *Fiia Platform Model* resource model

central server, or requiring knowledge beyond the address at least one neighbour.

All nodes can act as both publisher and subscriber. Platform information flows through the network, possibly arriving at a subscriber through multiple paths. This differs from the flat network hierarchies of SMTP and UPnP.

This architecture has built-in support for fault-tolerance. Each node can choose to subscribe to a single neighbour (low bandwidth use but also poor fault-tolerance), or to multiple nodes. In the latter case, if one publisher becomes unavailable, a subscriber will still receive data from its other publishers. In this way, when publishers leave the network (for failure or other reasons), no action is required to continue correct operation of its subscribers.

3.1 Resource Model

As shown in figure 2, the *Fiia Platform Model* structures resources in a tree format. As we shall see, this allows resource information to be recorded at fine granularity, while permitting coarse-grained filtering. Resource information is split into three broad groups specifying performance, input/output capabilities, and physical properties of the device. These allow users of the platform model to answer questions like: where is a server with sufficient available CPU to provide a user with a voice to text service, or where is the nearest keyboard that can be used to speed up interaction with a PDA.

Resource information may be *static* or *dynamic*. Static resources (e.g., number of keys on a keyboard, amount of memory on a server) do not change, and therefore require infrequent updates. Dynamic resources (e.g., CPU load on a server, remaining battery charge on a Smartphone) do change, and therefore require ongoing updates.

Our distributed approach to maintaining platform information makes it important to maintain the provenance of platform information. We therefore record for all resource knowledge the time at which the data was recorded and the number of hops through the distributed system that this data travelled before reaching the current node.

4. FILTERING

In a distributed publish and subscribe system, the data being transferred between nodes can become overwhelming. As each node passes on data not only describing itself, but also its neighbours and its neighbours' neighbours, the quantity of data becomes unbounded. The need to limit data transfer becomes particularly important for devices with limited bandwidth, or (as with some cell phone plans) where bandwidth is charged per kilobyte. We have investigated several filtering mechanisms for restricting bandwidth by allowing users of the *Fiia Platform Model* to trade off bandwidth for completeness and accuracy of information.

4.1 Interest-Based Filtering

Depending on their requirements, nodes are interested in different kinds of platform information. A tabletop computer might be interested in the availability of local devices that can help improve interaction (e.g., nearby keyboard for rapid text entry, nearby PDA to establish a second viewport). Alternatively, a PDA may be interested in nearby projectors and nearby nodes where high-performance computation can be performed. In general, low-powered devices with low-bandwidth connections need to limit the information to which they subscribe, and restricting information based on interest is an obvious place to start.

Name filtering provides a simple way of restricting data sent to subscribers. As seen in figure 2, the *Fiia Platform Model* organizes platform information hierarchically, allowing subscription to whatever part of the hierarchy is of interest. A node might subscribe to *PM.IO.Keyboard* to receive information on available keyboards, *PM.IO.Keyboard.** to receive detailed information about keyboards, or *PM.IO.** to receive information about all I/O devices.

4.2 Filtering Dynamic Data

Dynamic data, such as a device's current battery level, the number of items in a printer queue, or a node's current CPU load, can consume significant bandwidth. The *Fiia Platform Model* provides several mechanisms for filtering dynamic data. The most simple is *publishing interval* filtering, where the subscriber can specify the rate at which the publisher sends updates to dynamic information. Different rates can be requested for different resource types; for example, a node might subscribe to updates of *PM.IO.Printer.Status* every 5 seconds, and updates to *PM.Performance.Battery.CurrentLife* every 5 minutes.

Value-change filtering provides updates only when the change in the underlying value exceeds a specified threshold. For example, a subscriber may request reports on battery life be reported only following a 10% change from the last report. A variant on value-change filtering is *threshold filtering*, where a subscriber can specify that data should be sent only when some threshold is surpassed, for example, when battery life drops below 20%.

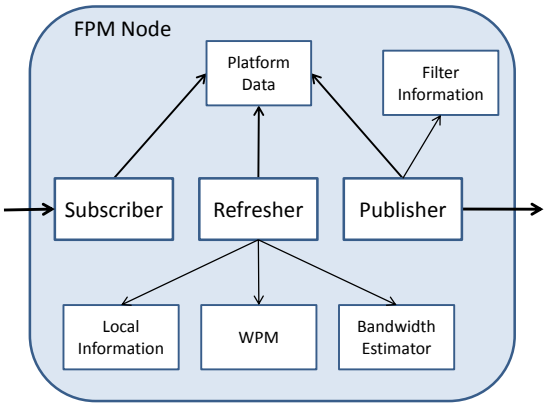

Figure 3: Architecture of a single *Fiia Platform Model* node

4.3 Filtering for Large Networks

One of *Fiia Platform Model*'s strengths is its support for dynamically created networks of nodes, without the requirement for a centralized locus of control. New nodes simply need to subscribe to another node in order to receive the network's platform information. This approach can lead, however, to unbounded data transferred via subscriptions as networks organically grow.

In practice, a given node is interested only in data in reasonable proximity. *Number of hops* filtering can be used to restrict the distance that information travels. As data is transferred from one node to another, its "number of hops" attribute is incremented. A subscriber can filter data that has traversed more than a specified number of hops, thereby controlling the locality of data in which it is interested.

5. IMPLEMENTATION

Each *Fiia Platform Model* node maintains its own knowledge of platform information, and acts as both a publisher and a subscriber. Figure 3 shows the internal architecture of a *Fiia Platform Model* node.

A *Refresher* updates information about this node to the *Platform Data* store. The refresher gathers resource information using three techniques. Performance information (CPU speed, battery life, current memory usage, etc.), is collected via Windows' *WPM* library. This library is available on both PC and mobile platforms. Information on bandwidth is estimated using the algorithm of He et al. [8]. Remaining platform information (such as the device's size and weight) is hand-coded in a file that is read on start-up. The Refresher is configured to update local information periodically, e.g., once per second.

A *Subscriber* component reads resource information sent by publishers on other nodes. If the information is newer than information already known, it is recorded in the Platform Data.

A *Publisher* component is responsible for periodically publishing information to this node's subscribers. The publisher periodically awakes, determines what new information is available in the Platform Data, and uses each subscriber's filter to determine whether to pass the data on to that subscriber.

6. EVALUATION

Two of our requirements for distributed platform modeling are support for heterogeneous devices and arbitrary scalability. The *Fiia Platform Model* meets these requirements through its distributed publish and subscribe architecture and powerful filtering

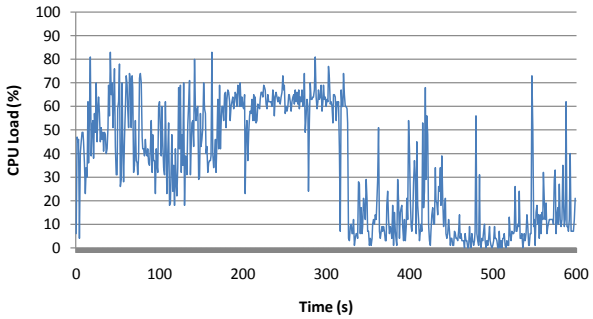

Figure 4: The CPU usage data used to experimentally validate the *Fiia Platform Model* architecture

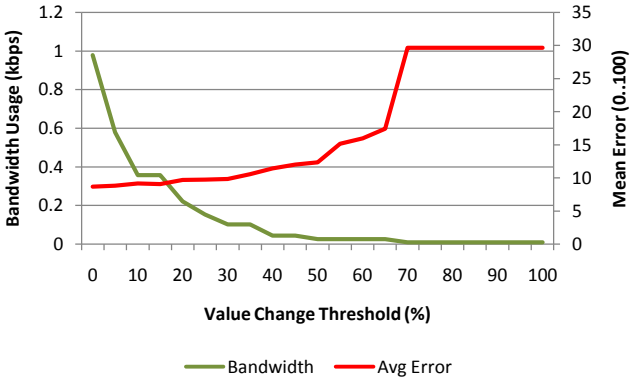

Figure 5: Effect of value-change filtering on bandwidth and mean error

language. Interest-based and dynamic data filtering allow nodes to reduce bandwidth (at the expense of error), while number of hops filtering enables tractability of very large networks. To illustrate these properties, we present two examples of the *Fiia Platform Model* in use. First, we illustrate how value-change filtering provides a tradeoff between bandwidth usage and correctness of platform information. Then, we show how number of hops filtering can be used to support large networks by trading off error, bandwidth use and coverage.

6.1 Effect of Value-Change Filtering

Value-change filtering allows *Fiia Platform Model* users to balance bandwidth consumption versus error. For dynamic data such as the current load of a node's CPU, accuracy will increase the more often messages are sent, at the cost of additional bandwidth. As we have discussed, *value-change filtering* sends messages only when the change in the underlying data exceeds some threshold. With our CPU load example, a value-change threshold of 20% would mean that a new message will be sent only when the actual CPU load differs by at least 20% from the last message sent.

To illustrate the effectiveness of value-change filtering, we performed an experiment with two nodes, one publisher and one subscriber. The publisher provided CPU load information to the subscriber with value-change thresholds ranging from 0% to 100% in increments of 10%. To provide consistency between the conditions, the same CPU load data was used in each (read from a file.) The CPU load data, shown in figure 4, represents load over a 600 second (10 minute) interval. This data was captured by sampling the CPU of a PC at one second intervals, while a user performed typi-

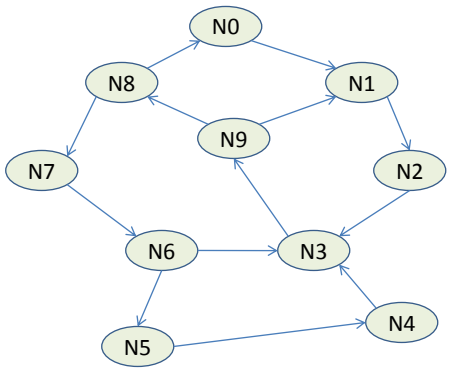

Figure 6: Network used to evaluate number of hops filtering

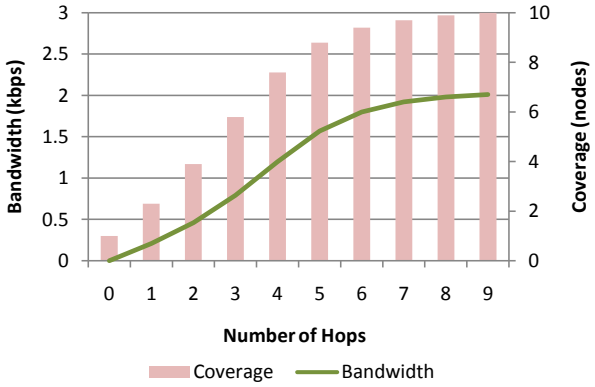

Figure 7: Effect of number of hops filtering on coverage and bandwidth

cal actions such as starting and stopping programs and performing computational tasks such as compiling and playing video. Messages were sent when the value threshold was exceeded, up to a maximum rate of 1 per second. Error was measured as the absolute value of the difference between the "correct" CPU load value (on the publisher), and that held in the platform manager of the subscriber, and therefore ranges from 0 to 100. The mean error is the average of errors sampled at 1 second intervals over the 600 second run-time of the experiment.

Figure 5 shows the impact on mean error and bandwidth of varying the value-change threshold. Not surprisingly, as the threshold increases, bandwidth use decreases while mean error increases. The relationship is not linear, however. Load values tend not to swing dramatically, meaning that a relatively low threshold dramatically reduces the number of messages. When a threshold of 70 or higher is used, no messages at all are required, since no CPU load value differs from the first value by more than 70. This leads to a plateau in error rates at an average of 30.

6.2 Effect of Number of Hops Filtering

As we have discussed, the core concept of *Fiia Platform Model* that allows ad-hoc connection of nodes with no central locus of control is the provision of *number of hops* filtering. This allows a subscriber to specify that it is interested only in data that has travelled a maximum of some n network hops. The following experiment illustrates that number of hops filtering allows subscribers to balance bandwidth use, coverage and average error.

Using a simulator, we established 10 nodes, each running an instance of the platform model, and connected with the subscription relationships shown in figure 6. Each node reported a CPU Load value, using the data shown in figure 4. The experiment ran for 600 seconds. Each node refreshed its CPU Load value once per second, and published its known data every 5 seconds.

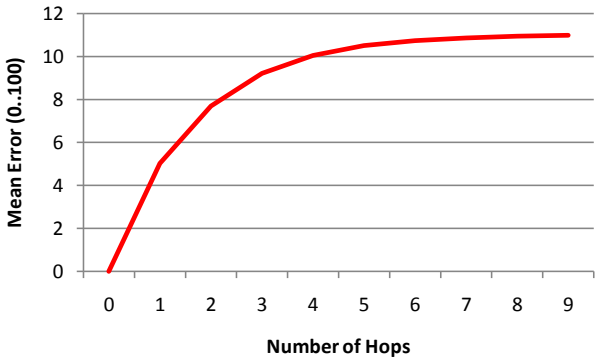

Figure 8: Effect of number of hops filtering on mean error

The graph in figure 7 shows the tradeoff between coverage and bandwidth consumed. We define coverage as the average number of nodes each other node has information about (including itself.) As the number of hops increases, so does coverage, and not surprisingly, bandwidth required. Both coverage and bandwidth increase sharply and then plateau; the existence of multiple routes between nodes sometimes allows data from distant nodes to be transferred in fewer hops.

Mean error also increases with the number of hops (figure 8), as the average age of platform data increases with the distance (hops) travelled. Error plateaus as coverage increases, and even at the maximum of 9 hops, is below 11 on a scale of 0 to 100.

From this example, it is possible to extrapolate how number of hops filtering supports very large networks. As each node maintains platform awareness of only those nodes close to it, addition of new nodes only affects nodes within a bounded number of hops. Networks can grow to arbitrary size due to this locality of information.

7. DISCUSSION

In the introduction, we identified four desirable properties of platform managers for distributed, interactive systems. In this paper, we have shown that these properties can be met with a design based on distributed publish and subscribe combined with a powerful filtering language.

The *Fiia Platform Model*'s implementation of distributed publish and subscribe does not require a *centralized locus of control*. Nodes may enter a network simply by subscribing to data from one or more neighbouring nodes. This approach provides high scalability. Despite the fact that an arbitrary number of nodes can enter the same network, most nodes will restrict the flow of incoming data using a *number of hops* filter. This means that networks organically pool into localities, where nodes have information about their neighbours, but not about the network as a whole. This gives the scalability benefits of systems with restricted size (e.g., UPnP, with its restriction to local area networks [10]), while allowing arbitrary node connection.

This approach is very powerful in mobile contexts. As a mobile node enters a new physical context, it need only connect to a local node to gain information about that context. For example, if a tour group enters a museum, one of the group member's Smartphones

might connect to a museum computer, automatically propagating information about the museum to the other group members.

The use of distributed publish and subscribe gives the additional benefit of *fault-tolerant operation*. If a publisher leaves the network (due to failure or other reasons), subscribers can rely on other paths through the network to obtain their data. Additionally, the fact that platform data is available on each node allows queries to be performed quickly, without the need for network communication.

The filtering constructs allow the flow of platform information to be flexibly matched to the *capabilities of heterogeneous nodes*. Within a network, powerful nodes should be capable of providing rich and detailed platform information. Nodes such as Smartphones or PDAs typically should subscribe only to the information that they require themselves. We have shown that several types of filtering can be useful, based on name, publishing interval, value-change, threshold and number of hops. As shown in our performance evaluations, the use of filtering allows users of the *Fiia Platform Model* to trade off completeness and correctness of platform information versus bandwidth consumed to collect it.

8. CONCLUSIONS

In this paper, we have presented mechanisms supporting platform management for distributed interactive systems. We have shown that an architecture based on distributed publish and subscribe combined with a rich and flexible filtering language supports heterogeneous device types, organic entry to and exit from the network, scalability and fault-tolerance. The viability of such an architecture has been demonstrated through its implementation within the *Fiia* groupware development toolkit.

9. ACKNOWLEDGMENTS

We gratefully acknowledge the funding of the Natural Science and Engineering Research Council of Canada and the NECTAR CSCW research network. We would like to thank Chris Wolfe for his technical help throughout this project.

10. REFERENCES

[1] G. Bartlett, J. Heidemann, and C. Papadopoulos. Understanding passive and active service discovery. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 57–70. ACM New York, NY, USA, 2007.

[2] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15:289–308, 2003.

[3] J.D. Case, M. Fedor, M. Schoffstall, and C. Davin. Simple Network Network Management Protocol (SNMP), Request for Comments 1157, Internic Directory Services, May 1990.

[4] W.K. Edwards, M.W. Newman, J. Sedivy, T. Smith, and S. Izadi. Challenge: recombinant computing and the Speakeasy approach. In *Proc. MobiCom '02*, pages 279–286. ACM, 2002.

[5] P.T. Eugster, P.A. Felber, R. Guerraoui, and A.M. Kermarrec. The many faces of publish/subscribe. *ACM computing Surveys*, 35(2):114–131, 2003.

[6] M. Florins and J. Vanderdonckt. Graceful degradation of user interfaces as a design method for multiplatform systems. In *IUI '04*, pages 140–147. ACM, 2004.

[7] Wireless Application Protocol Forum. WAG UAProf: WAP-248-UAPROF-20011020-a, 20 October 2001.

[8] J. He, C.E. Chow, J. Yang, and T. Chujo. An Algorithm for Available Bandwidth Measurement. *Lecture Notes in Computer Science*, pages 753–761, 2001.

[9] J. Janeiro, T. Springer, and M. Endler. A middleware service for coordinated adaptation of communication services in groups of devices. In *Proc. PerCom 2009*. IEEE, March 2009.

[10] M. Jeronimo and J. Weast. *UPnP design by example: a software developer's guide to universal plug and play*. Intel Press, 2003.

[11] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, M. Florins, and D. Trevisan. USIXML: A user interface description language for context-sensitive user interfaces. In *UIXML'04*, pages 55–62, 2004.

[12] G. Mori, F. Paternò, and C. Santoro. Tool Support for Designing Nomadic Applications. *Proceedings of the 8th international conference on Intelligent user interfaces IUI'03*, 12-15 January, 2003.

[13] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt. Gaia: a middleware platform for active spaces. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):65–67, 2002.

[14] D. Thevenin. *Adaptation en Interaction Homme-Machine: le cas de la Plasticité*. PhD thesis, Université Joseph Fourier, 2001.