A Harness Control Application for Hand-Held Devices

Tomasz Tyrakowski†, Vaidy Sunderam† and Mauro Migliardi†

†Department of Math & Computer Science
Emory University
Atlanta, GA
30302
{ttomek | vss | om}@mathcs.emory.edu

Abstract. This document describes the design of a Harness control application for hand-held devices. Some features and future directions are covered, as well as the limitation caused by the current state of the development tools for mobile computers. Section 1 describes the general idea of mobile interfaces to the Harness metacomputing system. In section 2 some of the mobile control application design issues for Harness are described, while section 3 analyzes the functionality of this application. Section 4 covers some other possibilities of the users collaboration utilizing Harness proxy plugins. Section 5 gives our conclusions about the most appropriate way of implementation and porting of the control application.

1 Mobile Interface To The Harness Metacomputing System

Harness is an experimental metacomputing framework based upon the principle of dynamically reconfigurable, networked virtual machines. Harness supports reconfiguration not only in terms of the computers and networks that comprise the virtual machine, but also in the capabilities of the virtual machine itself. These characteristics may be modified under user control via a *plugin* mechanism, that is the central feature of the system. The plugin model provides a virtual machine environment that can dynamically adapt to meet an applications needs, rather than forcing the application to conform to a fixed model [4].

The fundamental abstraction in the Harness metacomputing framework is the *Distributed Virtual Machine (DVM)*. Heterogeneous computational resources may enroll in a DVM at any time, however at this level the DVM is not yet ready to accept requests from users. To begin to interact with users and applications, the heterogeneous computational resources enrolled in a DVM need to be loaded as plugins. A plugin is a software component that implements a specified service. Users may reconfigure the DVM at any time both in terms of the computational resources enrolled, and in terms of services available by loading / unloading plugins [3].

The current prototype of Harness is a set of Java packages, which implement the functionality of the DVM, as well as some standard services (a farmer / worker model, the messaging subsystem and a PVM compatibility plugin) that allows a programmer to develop distributed applications with relatively little effort. The system is able to run in totally heterogeneous environment, as long as the nodes enrolled are connected to the network and can communicate with each other.

The recent advances in the field of hand-held computing are motivating designers and programmers to consider adding mobile interfaces to their systems. Hand-held devices are very specific, because of their limited resources and the way people use them

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2073, pp. 367–374, 2001.

(e.g. it can not be assumed that a device is permanently connected to the Internet). Therefore the mobile component of the system has to be designed with care so that the majority of processing is performed on the system side and not at the hand-held.

We postulate that there exists the following four main modes of operation involving mobile computers [6]:

- a single wearable computer
- a peer-to-peer wireless network of wearable computers
- a wearable computer and metacomputing system connected together
- a peer-to-peer network of metacomputing wearable computers

In terms of Harness, modes three and four are of most significance. In the third mode of operation, single users will connect to a back-end metacomputing system to conduct experiments, simulations or indeed any other activity that requires more processing power than is available at the hand-held [6]. Such powerful, heterogeneous computational back-ends can be provided by a Harness DVM. As it will be shown later in this document, the control of the DVM from a mobile device can be achieved with little additional implementation.

With the current capabilities of the hand-held devices and the development tools it's not possible to run a fully functional Harness kernel in such a limited environment. The first problem is the lack of a complete Java Virtual Machine for hand-held devices (there is a very limited virtual machine called KVM, but it's not powerful enough to run a Harness kernel). Further challenges are presented by the limited resources available. Therefore instead of extending Harness to hand-held devices, we propose a new design for a control application that is suited to mobile computers. Using such an application, a user of a hand-held device would be able to connect to the DVM, check it's state and modify some parameters. Due to technical issues (most notably object serialization and de-serialization used heavily by Harness), the application must have a 'proxy' at the Harness side. Thus, a specialized Harness plugin, which would translate requests from the application to the internal Harness protocol and vice versa is required. Since the application will be running on a limited device, the protocol between the mobile computer and the proxy plugin has to be as simple as possible. The remainder of this paper is structured as follows. Section 2 describes the application-plugin model more specifically, whilst in section 3 we present some conclusions about the possible functionality of this model.

2 Harness Control Application For Hand-Held Devices

As noted above, it is not possible with the current capabilities of hand-held devices to run a Harness kernel on them. Nevertheless, it is quite possible to implement a specialized application to control the DVMs state (i.e. to load / unload plugins, grab / release hosts, query the current state and intercept particular events). The model of such an application is a complementary pair of entities: a Harness plugin executing on a desktop computer and a small application running on the hand-held device. The plugin executes commands concerning the DVM on behalf of the user, who issues commands using a small application on the hand-held device. The main issue in this model is the communication between the application and the plugin. It is not profitable to implement specialized modules in the plugin, which would handle different types of communication (serial, modem, infrared, network), especially as each would require a significant

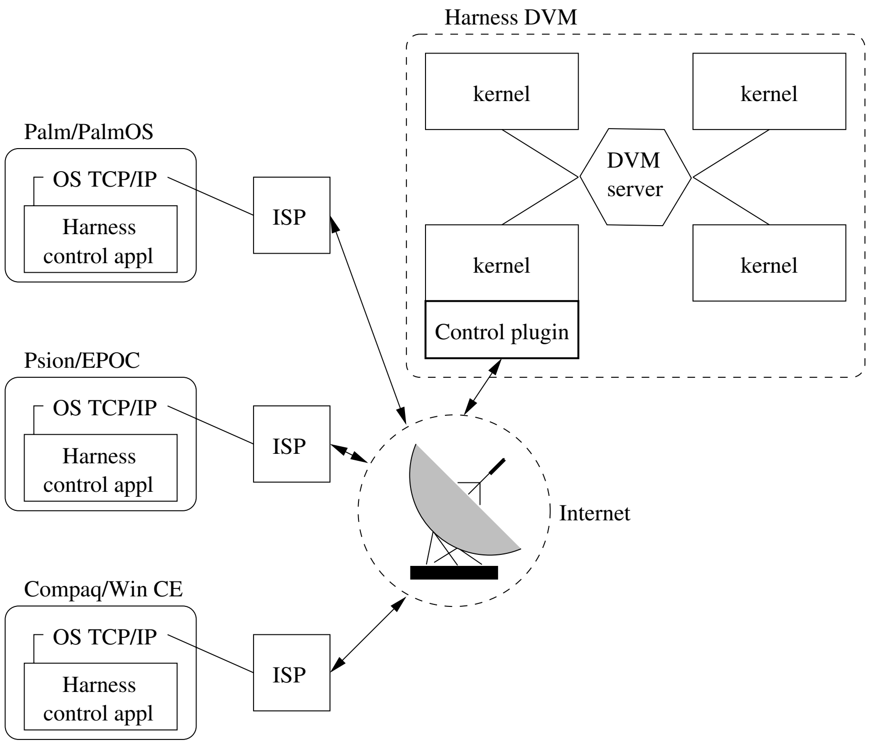

Fig. 1. The Harness Control Application Communication Model.

effort to implement. Instead, a more general solution is to implement the communication using the TCP/IP layer and the socket model. This strategy is utilized by most of the applications for hand-helds and offers the most general mechanism.

A TCP/IP connection is be provided by the operating system via an ISP (Internet Service Provider). This in turn assigns an IP address to the device. Thus, the control application itself does not depend on the type of the connection between the hand-held device and the ISP. Wired modem connections can be used as can GSM wireless connections or indeed any other bridging technology (e.g. infrared, LAN). Therefore, the principle of this model is that the type of Internet connection is completely transparent to the control application.

The protocol between the control application and the plugin should be extensible and heterogenous. In this instance, a simple text-based protocol is the most appropriate. A more sophisticated protocol can be introduced when experience with the system is gained. Thus, to avoid problems with differences in data representation, object serialization and several other technical issues, a text-based protocol will be adopted.

As an example exchange using the protocol, consider the following. The control application, after connecting to the plugin via a socket connection, sends the command LIST HOSTS and obtains a list of hosts as the answer. Other commands can be introduced (e.g. GRAB hostname, RELEASE hostname, LOAD plugin name hostname) to change the DVM status.

The implementation of the control application for hand-held devices is a pertinent issue. Since the devices, their operating systems and SDKs differ, there is no possibility to develop one application for all possible types of hand-held device. One can envisage such an application in Java, but Sun's Java 2 Micro Edition [2] (targeted specifically at hand-held devices) has very limited functionality and is still unavailable on some platforms. Therefore we propose to build native applications for different platforms exploiting software re-use to make migrating to a standardized approach trivial in the future. With the correct abstraction, it is possible to separate the system independent functionality of the application from the communication and visualization components, which are system specific.

Preliminary investigation explored the possibility of implementing a Harness extensions on the Palm Computing platform [5]. The prototype can be written in C using GNU-based PRC Tools (v. 2.0) for Palm and PalmOS SDK v. 3.5. The PalmOS platform contains a very convenient NetLib library that offers functionality similar to the Berkeley socket library. Although the data structures and function calls differ from the Berkeley version, it is possible to rapidly produce a working prototype using this library, for the purposes of testing and conducting further experiments. The NetLib library abstracts from the underlying transport layer rendering it suitable for use in this project. The application will use a system-level TCP/IP stack making the physical connection transparent. Thus, the Harness control application for PalmOS will connect to the Internet using any mechanism supported by the operating system.

It is impractical to create a fully functional Harness front-end for PalmOS without a significant amount of implementation effort. The difficulty stems from the nature of internal Harness communication. The Harness protocol relies on serialized Java objects being sent between the kernels and plugins. All Harness events are in fact Java objects and as such cannot be passed directly to a Palm (especially when they contain Java remote references, which are completely unmeaningful and useless on Palm). There are two types of events in Harness: system-generated and user-defined. The system-generated events form a relatively small set and can be translated by the proxy plug-in to a simpler form, understandable by the control application on the hand-held. The user-defined events can contain any serialized Java objects, thus there is no way to translate them into a form useful for the control application (assuming the application is not able to de-serialize and utilize Java objects).

Another problem with the Palm computing platform is that an application cannot be multithreaded. That implies a particular model of communication. An application can not be event-driven (in terms of Harness events) unlike the Harness Login GUI. Instead of waiting for an event to arrive from the proxy plug-in (which would block the whole application) the application has to actively query the plugin to obtain the recent system events. This can be achieved either by user request, or automatically, by downloading the list of events periodically. Both modes cause some unnecessary network traffic (the original Harness Login GUI receives the events only when they occur), but given the limitations of the system, this is unavoidable. The general model of a Palm application is shown in fig. 2 [1].

Figure 2 shows why there can not be a socket listening operation in the event loop. If a block is introduced in the 'event queue' step, then the whole application will block (as it will not receive any more system events). Although this is a technical limitation, it causes the application model to be constructed in a specific way i.e. the active querying described above must be adopted.

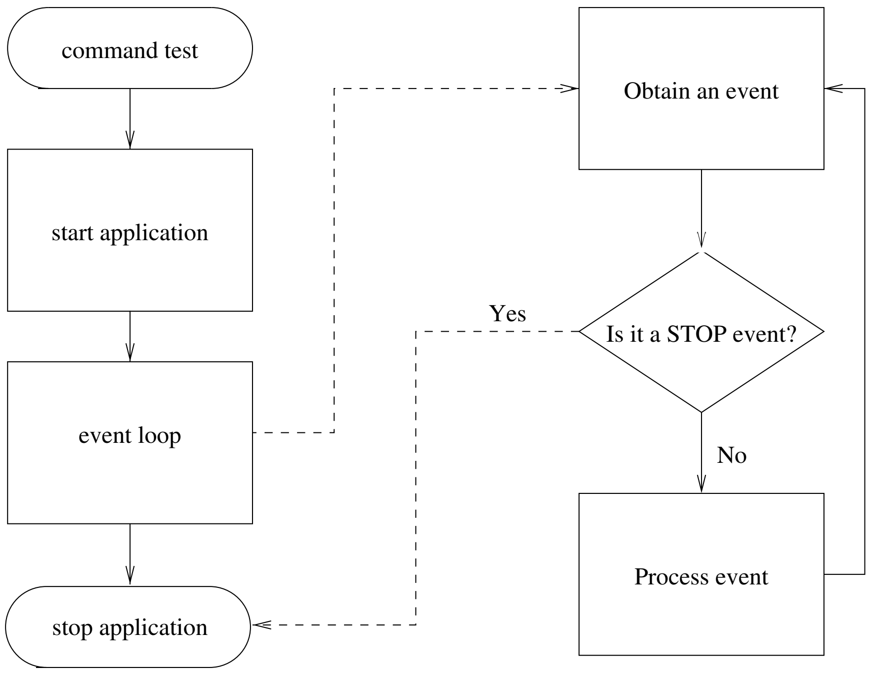

Fig. 2. General Model Of A Palm Application

3 Functionality Of The Harness Control Application

According to [6], the Harness control application represents an approach, in which a single mobile computer is connected to a heterogeneous computational back-end. Although fig. 1 shows more than one hand-held device connected to the Harness metacomputer, it is still a single mobile machine model because each of the devices shown is connected to the proxy plug-in independently. Thus, the users operating the hand-held devices may not have knowledge of each other. Therefore that is not simply a model with a network of wireless devices connected to a computational back-end.

M. Migliardi and V. Sunderam [6] proposed more a sophisticated model for connecting a wireless device to the Harness metacomputing system. This approach assumes a proxy plug-in is a fully functional Harness kernel, appearing to the rest of the Harness system as another node enrolled into a DVM. The hand-held device can communicate with the ‘ghost’ kernel and interact with the whole DVM using it as a proxy. Unfortunately the implementation of this model requires a significant investment of effort, because either the original Harness protocol must be used to provide a hand-held device with data, or all messages must be translated to a simpler form that is understandable by the hand-held device application. This approach can be easily implemented if a fully functional Java virtual machine existed for a particular mobile architecture, but as was noted above, the Micro Edition of Java 2 offers limited functionality and is still under development.

Thus, currently it is easier to add some extra functionality to the proxy plug-in and extend the hand-held-application-to-plugin protocol, than to implement a partially Harness compliant module for hand-held devices.

With a relatively small effort it is possible to implement a control application and a proxy plug-in. This will facilitate the following:

– query the status of the DVM (get a list of hosts, a list of plug-ins loaded at each host and a list of the hosts available to grab),
– grab and release hosts,
– load and unload plugins,
– intercept some of the events (those which can be translated to a non-object form and sent to the hand-held device),
– inject some user events (containing a string as payload).

This functionality offers a remote user good DVM control and limited control over simulations. Note, that there are only two ways a user can start / stop / modify a simulation. First, is by loading and unloading plug-ins. This method is not convenient as there is no way a user can provide or change the simulation parameters. The second method is by injecting some user events. This permits the inclusion of some parameters for the simulation and to control it more fluently, but the plugins, that do the simulation itself have to be written in a specific way to understand those events and act on them properly. The hand-held device can also receive some information from the plugins enrolled in the simulation. However, this information is usually simulation specific, and too verbose to be shown to the user in a readable form. Nevertheless, to control a very complicated simulation with large amounts of data, it is more appropriate to implement a specialized simulation front-end for hand-held devices, which would be able to represent the simulation status in a graphical form. Such an application would work in the same way as the DVM control application, but instead of connecting to a general DVM proxy and using a standard protocol, it would connect to a specific simulation control plugin and use the most convenient protocol.

4 Collaboration Using A Harness Proxy Plugin

Our investigation into the Palm computing platform shows, that other types of collaboration are possible, not only in terms of simultaneous simulation control, but also between the users themselves. The proxy plugin, described in previous sections, can be used also as a central point in the collaborative communication between the users of the hand-held devices. Moreover, this communication can also include users operating desktop computers, providing they are using appropriate front-ends.

The Harness event mechanism is a good tool to achieve this goal. Suppose the users connected to the proxy plugin want to talk to each other using their mobile computers. All the messages can be implemented as a special variety of user-generated Harness events. By extending the proxy-plugin is is possible for it to intercept and inject that type of events making them available for both users of the mobile computers and users of the desktop computers. Text messages are not the only type of data, which the application-proxy mechanism can manipulate. By adding new types of events users can send / receive bitmaps, sounds and other multi-media. The important issue is that the proxy plugin will keep the link with all of the connected mobile users, thus, it is not necessary to use a special server to cooperate with the other users. Moreover, as only people

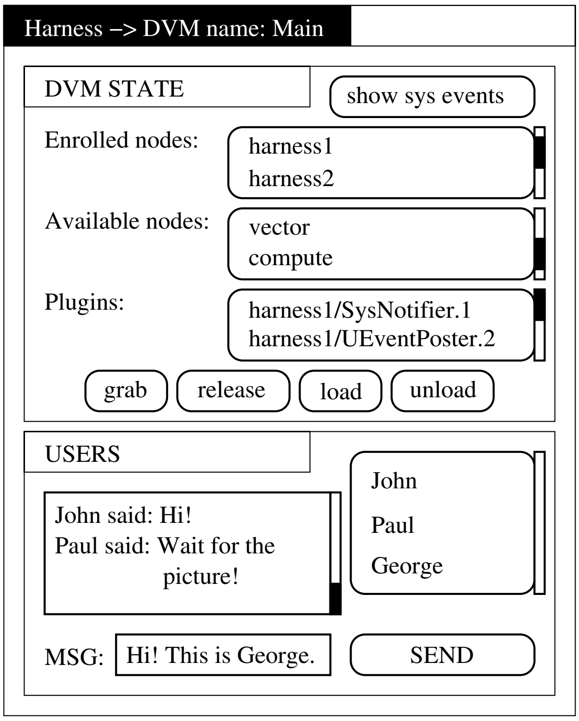

Fig. 3. The design of the control application GUI for Palms.

working on a common project will connect to a particular DVM they automatically will be able to communicate with their co-workers.

The proxy plugin can translate user-generated events of the specified type (user messages) into the format understandable by the control application. By extending the control application we can offer the users the ability to talk to each other. This brings the fourth mode of operation, described in section 1, into operation.

As an illustration, a possible GUI for the collaborative control application is shown in fig 3. The plugin itself, apart from the functionality described earlier, will perform the following activities:

- keep track of the users currently connected
- detect the additional types of events and take appropriate actions (translate the event's payload and send it to the message recipient or to all users if it is a broadcast message)
- queue the user messages for each connected user separately (since the messages will be sent to the user only when the control application asks for them, thus it is possible that there are more than one message present between such requests).

5 Conclusions

The Harness control application for mobile devices is a powerful extension of the Harness metacomputing system. The ability to check and change the DVM state and steer the simulation from different locations is a desired feature. Unfortunately the Java 2 Micro Edition architecture in it's current development and hardware support stage is not powerful enough to implement all of the features mentioned in [6]. Although facilities exist in the Java 2 Micro Edition distribution to handle basic network connections, it is not possible to serialize / de-serialize objects, which is a required feature to implement a fully functional Harness extension. Nevertheless, the implementation of a simpler control application for mobile computers (the one described in the previous section) would be particularly useful in terms of collaborative computing, and this can be implemented on most hand-held computers using their native SDKs. Providing that the application is designed to exploit the benefits of software re-use, it can be easily ported to the different platforms by re-writing the communication and visualization modules. The use of a simple text-based protocol guarantees that all platforms, which can handle network connections, will also be able to communicate using this protocol.

In the near future the control application for the PalmOS platform will be implemented. It can then be ported to the Psion/EPOC and Windows CE platforms. This would comprehensively cover the majority of the hand-held computer market. Simultaneously, an alternative version in Java can be produced, since it is likely that Java 2 Micro Edition will also develop in parallel.

The prototype at first will handle only the DVM state tracking / control. At later stages, the extended user collaboration functionality can be added to both the application and the plugin, as described in section 4. The complete collaborative control application can then be ported to different platforms and the Java front-end for desktops workstations can be implemented.

References

1. A. Howlett and T. Tso and R. Critchlow and B. Winton. *GNU Pilot SDK Tutorial*. Palm Computing Inc.
2. Curtis Sasaki. *Java Technology And The New World Of Wireless Portals And M-commerce*. Sun's Consumer Technologies, Sun Microsystems. available from: `http://java.sun.com/j2me/docs/html/mcommerce.html`.
3. M. Migliardi and V. Sunderam. Heterogenous Distributed Virtual Machines In The Harness Metacomputing Framework. In *Proc. of the Eigth Heterogeneous Computing Workshop*, April 1999.
4. M. Migliardi and V. Sunderam. The Harness Metacomputing Framework. In *Proc. of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, March 1999.
5. Palm Computing Incorporated. *PalmOS Software Documentation*. available from: `http://www.palmos.com/dev/tech/docs/`.
6. V. Sunderam and M. Migliardi. Mobile Interfaces To Metacomputing And Collaboration Systems. Technical report, Emory University, 2000. A Proposal To The NSF.