Mobile Frames: A pragmatic approach to automatic application partitioning based on an end-user data model

Thomas Kirste

Interactive Graphics Systems Group

Darmstadt Technical University, Department of Computer Science,
Wilhelminenstr. 7, D-64283 Darmstadt, Email: kirste@igd.fhg.de

Abstract

One important optimization technique for coping with resource constrained mobile hosts is *application partitioning*, the dynamic, resource-dependent distribution of application functionality across the available network nodes. This paper proposes a very pragmatic approach to this concept based on a specific *end-user data model*, the *mobile frame model*, which is an extension of the conventional frame model.

Keywords

Mobile computing, application partitioning, distributed systems, data models, frame models

1 INTRODUCTION

The introduction of mobile components (e.g., mobile computers, wireless networks, mobile users) into a distributed system infrastructure creates an application execution environment that is highly dynamic with respect to quantity and quality of the available resources. Applications that want to provide optimal performance in such an environment have to be *resource aware*. This includes the ability to *adapt* to the current resource configuration.

A substantial amount of research in the field of mobile computing is devoted to making applications adaptive and resource aware. This includes the definition and tracking of acceptable resource states as well as the introduction of suitable reconciliation strategies (such as substituting a monochrome still image for a high-definition color video stream) – see for instance (Davies et al., 1994; Satyanarayanan et al., 1995).

One important adaption technique is *application partitioning*, the dynamic, resource dependent distribution of application functionality across the available network nodes. A fundamental prerequisite for a successful generalized use of this technique is the definition of a strategy for computing a suitable partitioning scheme *dynamically* at application startup, so that the currently available execution environment is optimally utilized. Currently, there is no suitable concept available.

The point of this paper is to propose a very pragmatic solution to this problem based on a specific data model, the *mobile frame model*, which is an extension of the conventional frame model. The interesting fact is that support for application partitioning results as *inherent consequence* from the introduction of this end-user data model.

The contributions of the work presented in this paper to the field of mobile computing are:

- To realize the general importance of an expressive data model for mobile computing.
- The proposal of a suitable model, integrating both end-user data management and definition of interactive applications.
- The observation that fully dynamic, automatic application partitioning can be provided by a straightforward extension of this model.

(It is worth noting that the claim of this paper is not “mobile frames are *the* solution to application partitioning”, but rather “the data model needed anyway is *also* able to solve the partitioning problem for a small additional price”.)

The paper is further structured as follows: Section 2 gives a detailed discussion of the “application partitioning” technique. Section 3 discusses the need for an expressive data model and proposes the frame model as a suitable alternative. Section 4 describes, how application partitioning can be achieved by a simple extension of the frame model, the *mobile frame model*. Conclusions are given in Section 5.

2 APPLICATION PARTITIONING

The basic idea behind “application partitioning” is to divide an application into individual component processes, which have specific requirements in terms of computation and communication resources. Upon application startup, the component processes are dynamically allocated to the network nodes based on available resources (*startup adaptivity*). As the resources change during application execution (e.g., drop of bandwidth of a wireless communication link), the application components may dynamically be reallocated to other nodes (*runtime adaptivity*).

The basic concept is described, for instance, in (Watson, 1994), in (Joseph et al., 1995) (using the model of *Relocatable Dynamic Objects*), in (Cardelli, 1994) (within the definition of the language *Obliq*), and in (Kirste, 1995) (based on a combination of *Object Fragmentation* and *Pro-*

cess Migration). However, the important problem of *how* to partition an application is unsolved (Joseph et al., 1995; Kirste, 1995). In the current proposals, application partitioning schemes are individually designed on a per-application base by a programmer; the partitioning scheme is fixed at *implementation time*. The partitioning is usually even tailored towards a specific resource configuration, so that application partitioning today is very similar to conventional client/server programming – with the additional feature of client migration at startup (as provided, e.g., by Java (SUN Microsystems, 1995) through the down-loading of applets).

But in order to support startup and runtime adaptivity, the partitioning scheme must not be fixed at implementation time. It must be dynamically computed from the application definition based on the currently available execution environment.

In the next section, a seemingly quite unrelated problem is discussed, the question of a data model for a specific class of mobile applications. The interesting point is that this data model provides a solution to the problem outlined here!

3 DATA MODELS FOR MOBILE INFORMATION SYSTEMS

One of the visions of mobile computing is the creation of an integrated information environment, providing unrestricted access to both public and private data – anytime, anywhere. (A discussion of such a scenario and a prototype system is given in (Kirste, 1995).)

Besides location transparency, this environment should provide a *service transparent* data access. This means, the operations available for a data entity are determined by the entity type, and not by the service used for accessing this entity. (Today, access to, e.g., a spreadsheet, is not service transparent: access through the user's file system using its native application allows much more operations than when accessing it, say, as HTML-page through the World-Wide Web.) This requires an *expressive* data model, in which arbitrary data types and operations can be represented, in order to make them available for easy access by the user in a distributed environment.

Also, because the integrated information environment has to be open towards arbitrary future enhancements and extensions of its data set with respect to properties, types, and functionality, it needs a data model supporting *incremental type and data definition*.

One data model integrating incremental data definition, ease-of-use, expressiveness, and the ability for representing arbitrary interactive applications, is the “frame” model.

Originally, frames have been developed as a means for knowledge representation in artificial intelligence applications (Minsky, 1975). Powerful frame representation languages have been developed as early as 1977 (e.g., FRL (Roberts and Goldstein, 1977)). From this application area, frames have inherited the ability to cope with structured, fast changing information – as it also exists in the above scenario. Incidentally, other application areas for frame models, besides knowledge representation, have been Hypermedia-based information management systems (e.g., (Shi-

bata and Katsumoto, 1993) and Oval (Malone et al., 1995)) and personal information management systems, such as Apple's MessagePad (Smith, 1994).

Therefore, it seems appropriate to assume a frame-based model as underlying data model for an integrated information environment, rather than starting with the less expressive data model provided by the current incarnation of this scenario, the World-Wide Web. (See also (Kirste, 1996a) for a more detailed discussion of this point.)

4 THE MOBILE FRAME MODEL

4.1 Basic aspects of frames

The specific frame model underlying this work is basically the one provided by the MessagePad's object system (Smith, 1994), which in some aspects is a simplified version of the object system built into the language *Self* (Ungar and Randall, 1987).

As far as this paper is concerned, a frame is – quite similar to an object – an entity with a unique identity that contains a set of name/value pairs (“slots”). Among conventional data types, frame references may be slot values, so that frame structures can be created. Also, slot values can be *functions*, which may be invoked by message passing. Finally, frames can use other frames as *prototypes*, from which they inherit.

As an example, consider the frames *p* and *q*, defined as follows*:

```
p := {a: 1, f: func() a+b};
q := {_proto: p, b: 2}
```

p is a frame with two slots, *a*, which contains an integer, and *f*, which contains a function. *q* is a frame with two slots, *_proto*, containing a frame reference (*q*'s prototype), and *b*, containing an integer. So *q* inherits from *p*. As one would expect, *q.b* gives the value 2, *q.a* the value 1. The message invocation *q.f()* gives 3. Here, the values of *f* (and *a*, which is accessed in *f*) are inherited from *p*. The assignment *q.c := 3* creates a new slot *c* with the value 3 in *q* (so, *q.c* now gives 3). The assignment *q.a := 4* overrides the slot *a* inherited from *p*, so now *q.a = 4*, and *q.f() = 6*, but still *p.a = 1*.

It is also possible to define a frame

*The frame language syntax used throughout this document is based on NewtonScript (McKeehan and Rhodes, 1994), using the following conventions: The notation “ $\{s_i : v_i, \dots\}$ ” denotes a frame with slots s_i , which have values v_i . “ $f.s$ ” denotes the access of *f*'s slot *s* and “ $f.s := v$ ” is the assignment of the value *v* to *f*'s slot *s*. “ $\text{func}(x_i, \dots)$ *body*” denotes function with parameters x_i . “ $f:m(a_i, \dots)$ ” denotes the invocation of the function stored in *f.m* with parameters a_i (i.e., a method invocation).

```
r := {f: func() a*b}
```

and change q 's prototype by assigning $q.\text{proto} := r$. So now $q:f() = 8$, because this time f is inherited from r .

The central differences to the conventional class/instance model of object-oriented programming (OOP) are the dynamic addition/removal of slots to/from frames, which enables incremental data definition and data reuse/data sharing, and the prototype-based dynamic inheritance mechanism that allows a frame to inherit slots from an arbitrary other frame.

Frames therefore provide a dynamic environment that allows the user to flexibly create, augment, and modify information structures to suit his personal needs. It is now very interesting that frame structures can be used for the definition of interactive applications. The most prominent example for this is probably NewtonScript. This means, frames integrate a powerful end-user data model *and* a means for defining non-trivial interactive applications. As a consequence, the *same* system services (*e.g.*, frame migration and replication, frame caching) can be used for managing the access of data and applications.

In the next section, it will be outlined how the basic frame-model can be extended to support automatic application partitioning.

4.2 Frame-based application partitioning

The fundamental idea for supporting application partitioning is to group the frames defining an application into *clusters*. These clusters may then be allocated to the available network nodes (each cluster representing an application partition). Initial cluster setup (startup adaptivity) and dynamic reclustering (runtime adaptivity) are supported by the fundamental frame migration / replication facilities needed for basic remote data access. Clustering itself is guided by estimations of a frame's requirements in terms of computation, communication, and storage resources.

A (very simple) example for frame-based application partitioning is given in Figure 1. The "application" – computation of a factorial – consists of a "front-end" frame (doing some pre-processing) and two "back-end" frames, doing the "number crunching". The application can be called, *e.g.*, using the expression $a: f(10)$, producing 3628800.

4.3 The mobile frame model

In order to support the above application partitioning facility including runtime adaptivity, frames need to be *mobile* (*ie.*, migratable). Also, frames must support the parallel execution of methods to allow for multi-user interaction. For achieving these goals, the following strategy has been used:

```

{a: {f: func(x) b:g(abs(rnd(x)))},
 b: {g: func(y) if y <= 1
      then 1
      else c:h(y)},
 c: {h: func(z) z*b:g(z-1)}}

```

Figure 1 Example frame structure (left) and partitioning(right)

- Each frame is mapped to an individual process whose state also contains the frame's slots. Frame processes solely communicate by message passing (i.e., no hidden shared memory is introduced).
- Frame processes are designed to support recursive remote method calls (maintaining original frame execution semantics).
- Frame processes also support multi-threading – e.g., lookup of slot values during method execution (enabling basic parallel execution). Specifically, the model of *thread diffusion* is supported, providing complete location transparency for processing threads.
- The frame state completely describes the frame's process, so that copying the frame state implements migration of the frame process (which is required for runtime adaptivity).

The resulting model is called the *mobile frame model* (MFM). It provides the basic facilities for automatic application partitioning with startup- and runtime adaptivity in a multi-user environment. It is also quite easy to add basic support for “itinerant agents” by allowing frame processes to *actively* move between nodes.

The fundamental messages defined by the MFM for communication between frame processes are summarized in Table 1.

Furthermore, the current definition of the MFM provides additional important aspects, such as:

- The choice between static inheritance (“cloning”) and dynamic inheritance.
- The choice between remote message execution in calling frame or in called frame.
- A frame definition language (FDL) that allows to denote frames and frame structures. (The concrete syntax of the FDL adheres to the notation used in this paper.) The FDL language semantics describes the mapping of FDL terms to communicating frame processes.

Message	Meaning
SetSlot(<i>s</i> , <i>v</i>)	Set receiver frame's slot <i>s</i> to value <i>v</i>
Lookup(<i>s</i> , <i>o</i>)	Lookup of slot <i>s</i> in receiver frame. If <i>s</i> is found, its value is returned to the originator frame <i>o</i> . Otherwise, the receiver frame forwards the message to its prototype
Call(<i>m</i> , <i>v</i>)	Remote invocation of method <i>m</i> with parameter <i>p</i> in receiver frame
CreateFrame(<i>state</i>)	Requests creation of a new frame with initial state <i>state</i> .
Go(<i>p</i>)	Tells the receiver frame to move to place <i>p</i> . (The receiver frame then requests migration by sending the message Migrate(<i>p</i> , <i>state</i>) to the underlying process manager)

Table 1 Fundamental messages of the MFM

- A recursive “let” construct within the FDL that supports the simple definition of mutually recursive frame structures.

The first two points provide a range of tradeoffs with respect to data dynamics and message traffic. See (Kirste, 1996b) for detailed discussion of the MFM.

5 CONCLUSION

In this paper, a pragmatic approach to automatic dynamic application partitioning has been outlined. It is based on the introduction of an end-user data model supporting the creation of frame-based structured application definitions, which are used to guide partitioning. Furthermore, the basic frame model has been enhanced to support parallel execution and migration of active frames.

The mobile frame model is currently in the state of a theoretical concept with a precise mathematical specification (including the frame definition language). The fundamental viability of mobile frames has been verified by an abstract implementation of the model and its language using the functional language Haskell (Hudak et al., 1992). Here, frame processes are represented by Haskell functions of type $\text{PROC} = [\text{Rep}] \rightarrow [\text{Req}]$. I.e., a process is a function mapping a sequence of replies to sequence of requests. (See (Kirste, 1996b) for the details.)

The complete model also provides extended support for the flexible visualization of arbitrary

frame structures, based on the concepts of *Facets*, *Display Methods*, and *Viewers*, see (Kirste, 1996a).

A major drawback of the work presented in this paper is the lack of actual experiments with the behavior of frame-based application partitioning. These experiments are required for answering, *e.g.*, the following questions:

- Which strategies should be used for optimizing an implementation of the mobile frame model (*e.g.*, caching of inherited slots).
- How to design frame structures such that frame borders coincide with low traffic interfaces.
- How to describe a frame's resource requirements to guide clustering.

Future research will address these questions.

ACKNOWLEDGMENTS

This work has been supported by the German Science Foundation (DFG) grant En 123/20-1 as part of the researcher group "MoVi" (Mobile Visualization).

REFERENCES

- Cardelli, L. (1994). Obliq – A language with distributed scope. White Paper, Digital, Systems Research Center.
- Davies, N., Blair, G., Cheverst, K., and Friday, A. (1994). Supporting adaptive services in a heterogeneous mobile environment. In (MCSA'94, 1994).
- Hudak, P., Peyton-Jones, S., Wadler, P., Boutel, B., Fairbairn, J., Fasel, J., Guzman, M., Hammond, K., Hughes, J., Johnsson, T., Kieburtz, D., Nikhil, R., Partain, W., and Peterson, J. (1992). *Report on the functional programming language Haskell, Version 1.2*. SIGPLAN Notices 27.
- Joseph, A., de Lespinasse, A., Tauber, J., Gifford, D., and Kaashoek, M. (1995). Rover: A Toolkit for Mobile Information Access. In *Proc. Fifteenth Symposium on Operating System Principles*.
- Kirste, T. (1995). An infrastructure for mobile information systems based on a fragmented object model. *Distributed Systems Engineering Journal*, 2(3):161–170.
- Kirste, T. (1996a). A flexible presentation model for distributed information systems. In *Proc. 4th Eurographics Workshop on Multimedia (EGMM'96)*, Rostock, Germany.
- Kirste, T. (1996b). The mobile frame model. Technical report, Darmstadt Technical University, Department of Computer Science, Interactive Graphics Systems Group, Germany. Current

version available at

<ftp://dakeeper.igd.fhg.de/outgoing/kirste/mfm.ps.gz>.

- Malone, T., Lai, K.-Y., and Frey, C. (1995). Experiments with Oval: A radically tailorable tool for cooperative work. *ACM TOIS*, 13(2):177–205.
- McKeehan, J. and Rhodes, N. (1994). *Programming for the Newton*. AP Professional.
- MCSA'94 (1994). *Proc. Workshop on Mobile Computing Systems and Applications (MCSA'94)*, Santa Cruz, CA. IEEE Computer Society.
- Minsky, M. (1975). A Framework for Representing Knowledge. In Winston, P., editor, *The Psychology of Computer Vision*. McGraw-Hill, New-York.
- Roberts, R. and Goldstein, I. (1977). The FRL Primer. AI Memo No. 408, Artificial Intelligence Laboratory, MIT, Cambridge, Mass.
- Satyanarayanan, M., Noble, B., Kumar, P., and Price, M. (1995). Application-aware adaption for mobile computing. *Operating Systems Review*, 29(1).
- Shibata, Y. and Katsumoto, M. (1993). Dynamic Hypertext and Knowledge Agent Systems for Multimedia Information Networks. In *Proc. Hypertext'93 (November 14–18 1993, Seattle, Washington)*, pages 83–93. The Association for Computing Machinery.
- Smith, W. (1994). The Newton Application Architecture. In *Proc. IEEE Computer Conference*, San Francisco.
- SUN Microsystems (1995). *The Java Language Specification Release 1.0 Alpha 2*. Mountain View, CA.
- Ungar, D. and Randall, B. (1987). Self: the power of simplicity. In *Proc. OOPSLA'87 Conference*, pages 227–241, Orlando, Florida. Published as *SIGPLAN Notices* 22, Dec. 1987.
- Watson, T. (1994). Application design for wireless computing. In (MCSA'94, 1994).

6 BIOGRAPHY

Thomas Kirste has received his MSc and PhD in Computer Science from the Darmstadt Technical University. From 1989 to 1993, he worked as research assistant for the Computer Graphics Center (ZGDV), focussing on extensible Hypermedia systems.

From 1993 to 1995, he was a research assistant at the Darmstadt Technical University, working on the topic of mobile computing. Since 1995, he is responsible for the technical coordination of the MOVI project.

His research interests include mobile computing, mobile visualization, and hypermedia. His focus is the development of object models and language concepts. Also, he feels a strong inclination towards functional languages.