AN OBJECT MODEL SUPPORTING AD-HOC APPLICATION
PARTITIONING IN HETEROGENEOUS NETWORK
ENVIRONMENTS

THOMAS KIRSTE
Fraunhofer Institute for Computer Graphics
Joachim-Jungius-Str. 11, D-18059
email: kirste@egd.igd.fhg.de

Abstract
Application partitioning is an important technique for providing optimized performance of
complex, interactive applications in heterogeneous network environments with widely varying
capacities of execution and transport resources. The basic idea is to partition an application
into a number of component processes that may be allocated to the various available network
nodes with the objective to balance the load and optimize the overall performance. Partitions
and allocation schemes may possibly be adjusted during runtime to account for changing
situations. However, it is in general unclear how to find suitable partition borders: typical
object code is essentially unstructured and therefore does not lend itself to an easy (i.e., syn-
tactic) computation of potential component processes.

On the other hand, distributed hypermedia systems such as the World-Wide Web provide
already a very well structured representation for complex documents, where components can
easily be derived using a simple syntactic analysis. This paper introduces the concept of a
unified structured representation for applications and data, where syntactic techniques for
data analysis can be carried over to application partition computation. Furthermore, an execu-
tion model is provided that allows for dynamic migration of statefull partition processes,
supporting run-time reallocation and mobile objects.

Keywords: Mobile Computing, Distributed Object Models, Mobile Objects, Application Par-
titioning, Languages, Operating Systems.

1. Motivation

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. Appli-
cations 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 accept-
able 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 [2, 11].

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

Clearly, the computing environment of the World-Wide Web is, especially with respect to bandwidth dynamics, very similar to the mobile infrastructure outlined above. Indeed, the concept of suitably structured applications, whose components may be allocated dynamically to the available computation resources for optimizing execution behavior, has given rise to the implementation of Java-based systems supporting mobile objects (such as Objectspace: http://www.objectspace.com). This leads one to suppose that generic concepts for automatic application partitioning mobile environments might also be applicable to the problems of efficient distributed computing in the Web.

The point of this paper is to propose a very pragmatic solution\(^1\) to this problem based on a specific end-user 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 and distributed computing are:

- To realize the general importance of an end-user data model for mobile and Web-based information systems.
- 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.

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 [12], in [3] (using the model of Relocatable Dynamic Objects), in [1] (within the definition of the language Obliq), and in [4] (based on a combination of Object Fragmentation and Process Migration). However, the important problem of how to partition an application is unsolved [3,4]. In the current proposals, application partitioning schemes are individually designed on a per-application basis by a program-

\(^1\) This work has been supported by the German Science Foundation (DFG) within the scope of the researcher group "MoVi" (Mobile Visualization).
mer; 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 through downloading 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 an end-user 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. AN END-USER DATA MODEL FOR DISTRIBUTED 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 [4].)

Such an environment has to provide a suitable end-user data model, containing simple, yet powerful abstractions for describing, creating, structuring, and manipulating information, which allow the user to tailor and interlink the available information to suit his personal needs.

Today, the Web does not offer such facilities to the end user. (However, the little used LINK-Method of HTTP has clearly been designed with such a functionality in mind.)

One data model offering the required flexibility is the “frame” model. (The concept of “frames” as data model should not be confused with “frames” for simplifying screen layout, as provided by some HTML extensions.) Originally, frames have been developed as a means for knowledge representation in artificial intelligence applications [9]. Powerful frame representation languages have been developed as early as 1977 (e.g., FRL [10]). From this application area, frames have inherited the ability to cope with structured, fast changing information – as it also exists in an integrated information environment.

Other typical uses of frame models are – incidentally – Hypermedia-based information management systems (e.g., Oval [6]) and personal information management systems, such as the Newton Message Pad [7]. Any object-based data model providing an unrestricted linking operation (such as, at least in principle, provided by HTTP’s LINK) is in many aspects comparable to a simple frame model.

Basically, frames can be thought of as objects whose inheritance chain and instance variable structure may be changed during their lifetime. 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 [7]. This means, frames integrate a powerful end-user data model and a means for defining non-trivial interactive applications. And, most important, 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.
FrameStructure :=
  (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: Sample frame structure (left) and possible allocation (right)

4. THE MOBILE FRAME MODEL

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 re-clustering (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 fig. 1\(^2\). 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 (FrameStructure.a): f(10), producing 3628800.

In order to support this kind of computation, the standard single-threaded execution model of frames needs to be extended to support parallel evaluation in a multi-user environment. Here, the following straightforward strategy can bee used:

---

\(^2\) The language used for the example frame definition is similar to NewtonScript. The expression ($s_1; v_1, \ldots$) denotes a frame with slots (instance variables) $s_1$ that contain values $v_1$. Values may be numbers, strings, etc., and functions (denoted by $func \left( params \right)$ body). The expression $f.s$ retrieves slot $s$ of frame $f$. The expression $f.s(p_n, \ldots)$ denotes method invocation of the function stored in $f$.s with params $p_n$. The pseudo-variable $\text{self}$ used in a function body will refer to the receiver frame when the function is called by method invocation. The slot _proto refers to a frame’s prototype, from which slots may be inherited.
<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetSlot(s, v)</td>
<td>Set receiver frame’s slot s to value v</td>
</tr>
<tr>
<td>Lookup(s, o)</td>
<td>Lookup of slot s in receiver frame. If s is found, its value is returned to the originator frame o.</td>
</tr>
<tr>
<td>Call(m, v)</td>
<td>Remote invocation of method m with parameter value v in receiver frame.</td>
</tr>
<tr>
<td>CreateFrame(state)</td>
<td>Requests creation of a new frame with initial state state.</td>
</tr>
<tr>
<td>Go(p)</td>
<td>Tells receiver frame to migrate to place p. (The receiver frame then requests migration by sending Migrate(state, p) to the underlying process manager.)</td>
</tr>
<tr>
<td>Return(v)</td>
<td>Return a value (e.g., for Lookup)</td>
</tr>
</tbody>
</table>

Table 1: Fundamental Messages of the MFM

- 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 for communication between frames in the MFM are summarized in tbl. 1.

The distributed slot lookup process is outlined in fig. 2. In this example, a 3-level inheri-
tance chain is given, where frame A inherits from B, B from C, and C from D. If a frames slot
is referred, (such as in the expression A.x), the system first tries to find the slot in the referred
frame (in this case, A). If the slot can not be found, the system looks for a _proto_ slot,
whose value – if it exists – is a reference to the frames _prototype frame_. The system then
sends a lookup message to the prototype frame, where the lookup message contains the slot
being looked for as well as the frame originating the lookup. This lookup message may travel
upwards the inheritance chain until eventually a frame is reached where the slot is found or no
further prototype exists. This frame returns the resulting slot value (or nil in case of a
lookup failure) directly to the originator frame indicated in the lookup message.

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

- A recursive "let" construct for the definition of mutually recursive frame structures.
- The choice between static inheritance ("cloning") and dynamic inheritance.
- The choice between remote message execution in calling frame or in called frame.

The latter two points provide a range of tradeoffs with respect to data dynamics and message
traffic. See [5] for detailed discussion of the MFM.

It is now interesting to ask, how the definition of a data model is able to answer the ques-
tion of application partitioning: It is based on the introduction of an _integrated_ representation
of applications and data, resulting in the automatic applicability of data structuring techniques
to application structuring. This structure is then used to automatically determine application
partitions based on a _syntactic_ analysis. (Note that the same technique would be applicable to
an application design using conventional OOP, but at the price of requiring two _different_ mod-
els for representing applications and user data, resulting in the duplication of functionality.)

5. CONCLUSION

In this paper, a pragmatic approach to automatic dynamic application partitioning has been
described. 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. Further-
more, the basic frame model has been enhanced to support parallel execution and migration of
active frames. In [5], a detailed, executable specification of the Mobile Frame Model can be
found.

The complete model also provides support for the flexible visualization of frame struc-
tures, based on the concepts of _Facets, Display Methods_, and _Viewers_.

The Mobile Frame Model thus proves that a unified, flexible model for a structured, hyper-
text-like representation of data and applications can be quite easily defined. So it is seems
to be possible – at least in principle – to derive a similar data model for the Web that com-
bines hypermedia navigation, flexible user manipulation, and structured, mobile computa-
tions. Clearly, setting up system support for the MFM (or any similar approach) on Web scale
eventually amounts to establishing a single, global distributed operating system.

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
identifying suitable strategies for an efficient implementation of the Mobile Frame Model
(_e.g._, caching of inherited slots).
REFERENCES


Author

Thomas Kirste studied computer science at the Darmstadt Technical University (TUD), from which he received his PhD in computer science in 1995.

From 1989 to 1993 he worked as research assistant at the Computer Graphics Center (ZGDV) in Darmstadt on the topic of open hypermedia systems, where he was responsible for the design and implementation of the “HyperPicture” system, as well as several applications built on top of this system within the scope of industrial projects.

In 1993, he began his work on Mobile Computing in the ZGDV department “Mobile Information Visualization”. He was technical manager of the “HyperFunk”-project, the development of a prototype mobile information system sponsored by DeTeMobil.

In 1994, he joined the Interactive Graphics Systems group of the Department for Computer Science at Darmstadt Technical University (TUD-GRIS). Here, he worked as technical coordinator of the Rostock-centered cross-institutional researcher group MoVi (“Mobile Visualization”) funded by the German Research Society (Deutsche Forschungsgemeinschaft, DFG). The ongoing work in MoVi concentrates on basic research on concepts and strategies for the mobile visualization of globally distributed multimedia information services.

Since August 1996, he is heading the R&D department “Mobile Multimedia Technologies” at the Fraunhofer Institute for Computer Graphics Rostock.