Design and Early Implementation of the Cadmium Mobile and Disconnectable Middleware Support

Aline Baggio

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Design and Early Implementation of the Cadmium Mobile and Disconnectable Middleware Support

Aline Baggio

No 3515

Octobre 1998

THÈME 1

ISSN 0249-6399

Design and Early Implementation of the Cadmium Mobile and Disconnectable Middleware Support

Aline Baggio*

Thème 1 — Réseaux et systèmes
Projet SOR — http://www-sor.inria.fr/

Rapport de recherche n ° 3515 — Octobre 1998 — 17 pages

Abstract: The Cadmium project provides system-level support for disconnected and mobile usage. This includes basic mechanisms for ensuring data availability, such as replication, caching, prefetching, as well as services for consistency, adaptation to environment changes, and so on. Cadmium uses an application-aware approach. It allows the applications or users, to be aware of currently available resources (e.g. network bandwidth). Application-awareness is achieved through cooperation between the system and the applications. The system provides environment monitoring, upcall registration and event notification to send information about environment evolution. Applications dynamically adapt whenever required. Cadmium is still an on-going project.

Key-words: Mobile computing, objects, replication, caching, prefetch, system support, flexibility, adaptability, Cadmium

(Résumé : tsvp)

* E-mail: Aline.Baggio@inria.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : 01 39 63 55 11 - International : +33 1 39 63 55 11
Télécopie : (33) 01 39 63 53 30 - International : +33 1 39 63 53 30

Design et Mise en Œuvre du Support Système Mobile et Déconnectable Cadmium

Résumé : Le projet Cadmium fournit un support système pour les environnements mobiles et déconnectés. Il comprend des mécanismes de base destinés à assurer la disponibilité des données, tels que la réplication, la gestion de caches, le préchargement ; ainsi que des services de cohérence, d'adaptation à l'environnement, etc. Cadmium utilise une approche où les applications sont inclues dans le processus d'adaptation (*application-aware*). Ceci permet à la fois aux applications et aux utilisateurs d'être informés des modifications de leur environnement de travail, des changements d'état des ressources (par exemple, la bande passante disponible). Ce support adaptable est proposé à travers une coopération entre le système et les applications. Le système inclut des services de surveillance de l'environnement, d'*upcalls* et d'abonnement à des notifications. Les applications sont ainsi à même de s'adapter dynamiquement à leur environnement. Cadmium en un projet en cours.

Mots-clé : Informatique mobile, références, objets, réplication, gestion de caches, préchargement, support système, flexibilité, adaptation, Cadmium

1 Rationale

Laptop computers connected to indoor or outdoor wireless networks are becoming an alternative to classical fixed (i.e. immobile) computing. The challenge of mobile computing is to provide users with mobile devices as useful as fixed ones.

In this article, we mainly consider the use of powerful laptop computers, as opposed to hand-held notebooks, keyboardless devices or smart cards. Laptops are most useful if they provide a stable and quite normal working environment, despite changes such as relocation or network bandwidth varying from high-speed to none. This could be called "unplug-and-play" [1].

Our goal is to ensure continuity of service in a best-effort manner. Achieving this goal requires support for availability of data across disconnections, as well as responsiveness to environment changes in order to take advantage of all available resources [5, 10, 2].

Cadmium addresses the above problems by allowing data and code replication on both parts of the network, fixed and mobile; by providing additional support for dealing with these multiple copies; and by setting up hooks for system or application cooperation and adaptation to environmental changes.

This article is organized as follow. Section 2 presents the underlying model. Section 3 shows Cadmium design aspects. Section 4 gives details about its early implementation. Section 5 describes possible uses of Cadmium and provides a preliminary evaluation. And finally, Section 6 concludes and gives hints about the future work.

2 Model

Cadmium is based on a few assumption about the network and the interactions between the system and its applications.

Network and distribution model: Mobile devices strongly rely on the network infrastructure, wireless as well wired on one hand. On the other hand, they need to be autonomous enough to support disconnected operation and varying quality of service.

A mobile host is inherently dedicated to a single user, and might be the only machine the user has. Therefore, we assume the use of laptops which can be used as clients as well as servers. In general a laptop is used as a client. However a mobile host that holds a private application or data will also act as a server for that private information. Consequently, availability is also an issue for fixed machines that use resources held by mobile hosts.

Replication model Autonomy is preserved by using data replication[1] and by exploiting reconnection times in order to propagate updates [8]. The fact that replication also improves performance is secondary in this context. For availability, the replicas must be fully accessible, allowing writes as well as reads.

[1] By replication we intend all the algorithms based on data copying, such as caching, prefetching, mirroring, etc.

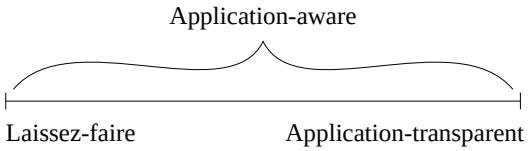


Figure 1: Taxonomy of adaptation strategies

When disconnected, some data will eventually not be available. These misses have to be managed carefully since the laptop is not guaranteed to be able to download missing data. In order to limit the number of misses, it is important to (1) select carefully locally stored data, using user hints, explicit requests, and/or application spying; (2) and to exploit connected or weakly connected periods to fetch missing data.

Adaptation model Mobile environments are inherently turbulent [9]: the software must adapt. The best way to provide this adaptation for diverse and concurrent applications is through a collaboration between the system and applications.

As the resources available to a mobile host change, applications can reflect these changes and modify the way they access data, either to consume less of some newly limited resource, or to take advantage of a sudden abundance [9]. This approach is called application-awareness (Figure 1). It provides the ground for tradeoffs, standing between application transparency where the system completely hides mobility to applications, and laissez-faire where no system-support is given and everything is up to the application code.

Cadmium has an application-aware model similar to Odyssey [13, 10]. The system, applications and users, can all be made aware of currently available resources [3].

Interaction model To achieve application-awareness, cooperation between the system and the application takes three forms: use of dynamically pluggable algorithms (i.e. strategies); environment monitoring, upcall registration and event notification; and application or user hints[2]. These mechanisms are described in the next section.

3 General design

The Cadmium project provides extensible support for the system and applications to cooperate in managing mobility. The system provides basic mechanisms as well as feedback about environment evolution. Applications dynamically take advantage of this support to adapt whenever required.

[2]Offered or requested hints complete the interaction between the system and its applications and users. However they are out of the scope of this article.

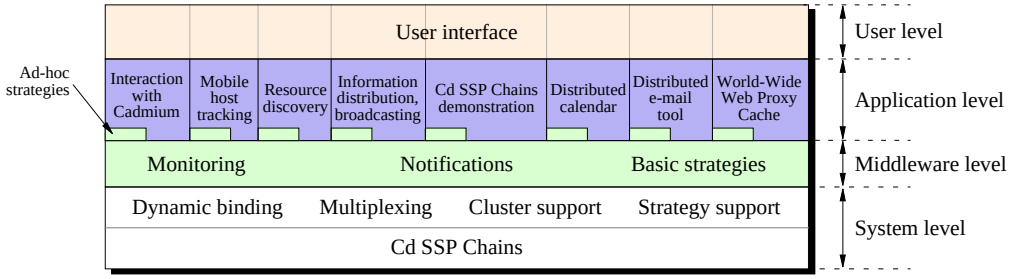


Figure 2: Cadmium design overview

The mechanisms are distributed across four levels: system, middleware, application and user (Figure 2). Each of these levels holds its part of the mobility and interaction support. The end user of Cadmium should only see a small subset of the mechanisms.

System level: The system level provides the basic abstractions. This includes objects and references. A reference retains its meaning across disconnections. This property is ensured by object migration or replication. It is tightly coupled with a reference redirection mechanism, i.e. flexible binding [7, 14, 2]. It allows to select dynamically the "best" representation for an object.

Middleware level: The middleware level supports replication. Replication can take several forms (i.e. loadable policies we call *strategies*): on-demand caching, prefetching or mirroring of objects, on-the-fly selection and replication of objects, etc. [1]. In each of these cases, the caching can be more or less aggressive. For example, when replicating a file, we can choose to fetch the single file or to preload the whole directory.

Replication requires additional mechanisms to manage replica versions, ensure propagation of updates, detection of conflicts and automatic resolution. This has to be independent of the application objects. Therefore these tools are also provided as libraries of strategies. Similarly strategies exist for access control, consistency, conflict detection and resolution, filtering, compression, etc.

Since Cadmium supports different applications, the provided strategies do not take in account the data type or the application semantics. For this reason, we also support strategy adding and loading. An application can plug in its own strategies, defined by the application programmer. This provides yet another form of adaptation.

Furthermore, an application can adapt dynamically to changes in its environment. It can switch on-the-fly between strategies. One can for example switch between a strong consistency strategy while fully connected, and a weak consistency strategy while weakly connected or disconnected.

Cadmium provides notification in order to react to the environment changes. It provides monitoring and upcall registration à la Odyssey [10]. It sends notifications about environment changes to the registered application.

Application level: The application level holds the software that interacts with Cadmium's mobility support.

This level contains application-specific strategies, i.e. ones that are too type or semantic-specific to be provided as part of the Cadmium libraries.

User level: Finally, the user level includes the user interfaces to all applications or to Cadmium. The latter includes commands for getting Cadmium state, for giving hints, writing user or application profiles, etc. Cadmium can get input and feedback by requesting hints about user needs under the from of profiles, hints about miss severity, etc.

4 Implementation

4.1 Objects and references

In Cadmium, the manipulated data are objects able to migrate along with their threads. This abstraction is provided by the underlying Objective Caml (oCaml) [12] virtual machine. The system level exports the object abstraction to the higher levels. Using this basis, we can for example transparently add methods to application classes or modify object behavior.

We consider objects with various granularities, from a single piece of data to a whole file or directory contents. Objects can be *clustered* into groups. Clusters provide an adaptable-grain access scheme. They can be used as a replication unit. The way a cluster is constructed is dependent of the application.

Finally, our objects may change behavior according to available resources, e.g. while disconnected, weakly connected, or fully connected. For example when disconnected, a printer object can queue requests, check connectivity periodically and notify the user.

An object is identified by a reference. Our reference mechanism is called Stub-Scion Pair Chains (SSP Chains) [15, 11] (Figure 3). Given a SSP Chain reference, a program can invoke the referenced object even if it moves. SSP Chains support garbage collection.

In Cadmium, we have developed extensions to the basic SSP Chains to provide mobility support, concerning four main points. We call them Cadmium oCaml SSP Chains (Cd SSP Chains).

Accessibility: In a mobile environment, it is common that some objects remain inaccessible for a while. The references to them will be temporarily unavailable. Since this is a normal state, the references must neither be recovered by a fault tolerance protocol, timeout, or break.

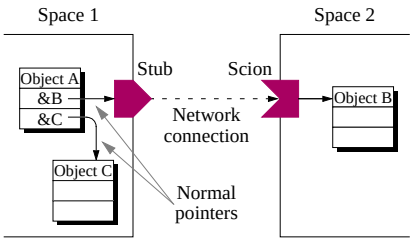


Figure 3: A Stub-Scion Pair Chain

Disconnected state: The above accessibility problem may need to be explicitly notified. The reference model includes a concept of disconnected state, and provides feedback to the caller.

Moving target: Mobile environments made commonplace the concept of a *moving reference target*. That is to say, a reference does not always point towards the same location (object migration) or towards the same object all the time (reference redirection). This is dependent of available data, host connection state, and user or application requests. The problem is exacerbated with replication: we will have to choose between different copies of the same object.

Reference tracking and replication: Cd SSP Chains enable transparent application object replication by searching entering and exiting references (called scions and stubs respectively). Given a space (e.g. Space 1 in Figure 3), reference tracking allows to discover remote objects that are currently in use (e.g. object B in Space 2). The discovered references allow to fetch these objects. The local storage can use either a specific space (i.e. a cache) or the application space itself (e.g. Space 1). The reference tracking and replication process can be initiated on-demand, triggered by an environment change using Cadmium upcalls, etc. [1].

4.2 Binding

An object is designated by a reference. If that object becomes unavailable, the reference could be redirected towards the a replacement object, by instance to some available replica. This makes replication transparent.

To deal with this "moving reference target", it is necessary to use a reference redirection protocol called flexible binding [7, 14, 2]. For instance a reference to a file can be bound to a dedicated file server when at the office, switched to a locally cached copy on the road, and then to a secondary server when reconnecting at a new location. The rebinding strategy that selects the reference target might choose between between different strategies, e.g. select

"nearest" replica, the least loaded servers, take in account connection state, the available networks, the available replicas, the user or application hints, requested quality of service, etc.

Reference redirection itself can take place after an object has been locally copied into a cache space. It is an exchange of messages between the client, the cache and the server spaces. This can happen just after the object replication or lazily after an invocation failure.

As the redirection protocol is flexible, we can choose to switch back to the original when reconnected or to keep working with the copy. This is dependent of the application behavior as well as the strategies the replicated object uses. In any case, we are able to keep track of reference to the alternative objects that were in use.

4.3 Strategies

Cadmium strategies are used by applications for adaptation purposes. A strategy can be applied on an object, cluster or whole application basis.

Strategies are defined as an extensible library of classes. Each strategy has a generic class, its set of methods and functions. It is possible to instantiate a strategy from one of these classes and attach it to some application object.

Let us take an example: a replication control strategy. The goal of a replication control strategy is to determine if an object is allowed to be copied or not. It does not perform the copy itself. This is done by another strategy which will determine if the object is fully replicated or by fragments, where it is stored, etc.

The replication control can decide to *always* allow replication, or to *never* allow it, or to allow it if and only if there are less then n copies of the object, or only on a number of *trusted* sites. Such control patterns are implemented as oCaml functions and wrapped into the application objects. Specific replication control strategies are classes that inherit from the generic replication control strategy class.

Once the application object is linked to its replication control strategy, whenever the object has to be copied, the replication right will be checked.

Cadmium defines a set of aspects that are governed by strategies, for instance, replication control, access control, replication, and consistency. For each aspect, Cadmium comes with a library of classes and functions that each implement a strategy alternative. One of the available classes is designated as the default strategy. For example, we chose that the default value for the replication control strategy is to always replicate. Obviously, the default can be overridden whenever necessary. In our example, it is easy the switch to a n copies replication control strategy.

A design choice is to use wrappers in order to achieve the real binding between a strategy and an application object. Wrappers use the ability of oCaml to pass functions as parameter. They make easy the dynamic changes of strategies but they may cost a little bit at the time of strategy invocation. An alternative approach is to dynamically load the strategy code into the object. This approach is feasible and also more transparent, however we rejected it for our prototype because of its complexity.

4.4 Monitoring

An application can monitor changes of critical resources.

For rapid prototyping and proof of concept, the current Cadmium prototype includes a tool to simulate the changing environment: network behavior, user and laptop geographical position and so on. It allows to generate real upcalls upon simulated moves or bandwidth changes and therefore evaluate the behavior of our tools and applications.

The type of resources Cadmium monitors are various. They can be local or remote data, other application or system tools, as well as available network bandwidth, latency, connection state, available interfaces and cost, disk space, memory used, CPU time, battery power, geographical position, external screen in a conference room, etc. For example we can track whether the available bandwidth is falling below 1 kbytes/s, the latency is getting higher than 3s, or the laptop moves away from a specific point (physical distance from home or university server).

An application registers its representative thresholds in order to track changes of critical resources. It is possible to register several thresholds for the same resource (high and low threshold), possibly with an hysteresis. Thereafter, whenever the actual resource state reaches a threshold, the application receives an upcall. It is up to it to react accordingly, for example by switching from one strategy to another, and to gracefully handle service degradation (or improvement).

Current thresholds are only numerical values. We can imagine to support user defined criteria if they are provided along with their comparison primitive.

5 Use and evaluation

5.1 Applications of these mechanisms

Many of the services provided by Cadmium appear to be useful for mobility and environments where the external condition are highly changing. Such mechanisms are also considered useful in large scale networks where the partitions can be frequent or high latency. For example, a collaborative World-Wide Web proxy-cache can reuse Cadmium concepts: some of the Cadmium internals, such as automatic redirection, management of multiple object sources, appear interesting for document retrieval, dynamic rebinding towards mirrors, etc. Using Cadmium, an application refers to the objects it needs independently of mobility considerations.

Cadmium can be used with various classes of applications that share data, and adapt to environment changes. Collaborative, mobile or distributed applications can benefit from this support, for example calendars, e-mail tools, collaborative editors, whiteboards, etc. Experiments with applications are however at their very beginning.

Let us take the example of a distributed calendar tool. This is a representative enough of general problems.

Our calendar manipulates objects known as items (appointments, to do lists, or reminders). The calendar, as well as items, are shared by a group, for instance a research

	100 kbytes	1 Mbyte	10 Mbytes [a]	10 Mbytes [b]
Duplication	0.06312654	0.06067371	0.09843250	0.19448732
Migration	0.01976385	0.14223115	1.45702625	12.64152310
FTP	0.0332	0.1620	1.486	1.567

Table 1: Local object duplication and migration in seconds (Blake, Mortimer)

	100 kbytes	1 Mbyte	10 Mbytes
Duplication	0.02230256	0.01688121	0.67672933
Migration	0.11964081	2.54229433	57.21317333
FTP	0.04328	0.4044	5.581

Table 2: Local object duplication and migration in seconds (Lachesis)

	100 kbytes	1 Mbyte	10 Mbytes
FTP	0.125	1.24	12.4
Migration	0.17857984	1.36830395	13.46508815
Overhead	0.05357984	0.12830395	1.065088100

Table 3: Remote migration compared to FTP in seconds (Blake, Mortimer)

project. Users can use fixed as well as mobile hosts. They can use remote items or group of items, accessing them via references.

In such an example, Cadmium can be used to replicate the shared items. Even connected users will have problems with disconnection of other if those ones hold shared objects. As a matter of fact, they will also rely upon the Cadmium replication service.

The calendar can use strategies for controlling the access or replication rights to some personal items. It can also use strategies for dealing with multiple item updates, overlapping ones, and so on. These latter strategies will be application-defined. They will take in account the semantics of calendar items. For example, several meetings can take place at the same time if the are not planned in the same meeting room and if the sets of people who attend to both are disjoint.

5.2 Evaluation

We are evaluating our prototype with micro-performance measurements. These early results are shown in the following tables and graphics.

Measurements have been taken on three machines: one laptop (Lachesis), a Compaq LTE 5400 (processor Intel Pentium 150MHz and 32 Mbytes of memory, using Linux 2.0.33); and two Digital PWS 500au machines (processor Alpha 21164 500MHz, using Digital UNIX

	Average	Highest	Lowest
Same process	0.00002817	0.00003906	0.00001951
Same machine	0.00110606	0.00285157	0.00093846
Remote machine	0.00129421	0.00475586	0.00116211

Table 4: Null method call round-trip time in seconds (Blake, Mortimer)

V4.0D), on a low load. One PWS (Blake) has 704 Mbytes of memory and the other (Mortimer), 256 Mbytes. We used the last one for comparison when dealing with 10 Mbytes objects and for remote migration of objects. All measurements involving the network have been done with a 10 Mbytes/s Ethernet.

Figures 4, 6, 7, and 9 show the values for local (i.e. same machine, different processes) object duplication and migration. Table 1 shows the associated average values. The experiment used client and server processes on Blake. The object migration results for Blake from Table 1, Figure 7 and 9 are close to local FTP transfer times.

Figure 4 (small objects duplication on Blake) presents a measurement precision problem (i.e. peaks): precision was up to 1ms. This pattern does not appear on Figure 5 (small objects duplication on Lachesis), neither for longer operations.

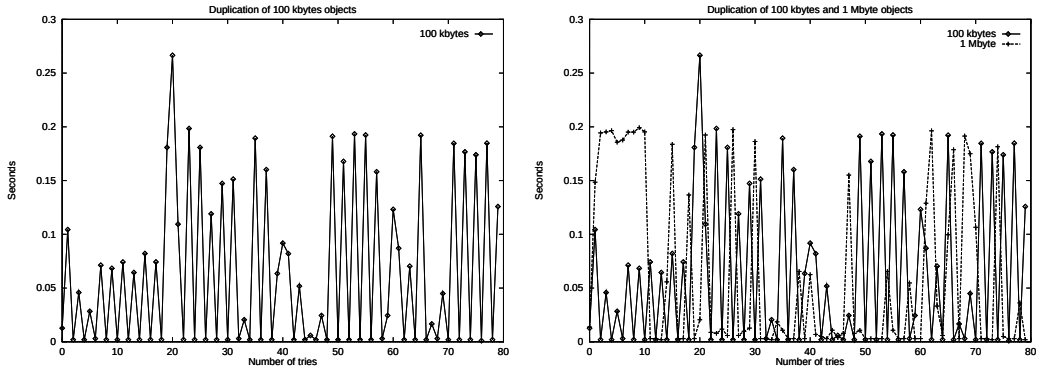
Figures 6 and 9 show several sets of 10 Mbytes objects duplication or migration on Blake and Mortimer. Table 1 column (b) gives figures for an execution using 10 Mbytes objects exclusively on Mortimer. These experiments show the time Mortimer spent in swapping.

Figures 5, 8, and Table 2 give the same experiments for Lachesis. Due to high memory consumption, we do not have yet representative results for 10 Mbytes objects.

The results given by the Table 3 and Figure 10 are remote object migrations. They were obtained using Blake as a server (i.e. object holder), and Mortimer as a client and cache. At the end of the migration, Mortimer has a local copy of the object in its cache. The overhead between our migration process and FTP transfer is due to time spent in message handlers, object invocations and strategy execution, in this case the replication control strategy. High values in the 10 Mbytes object case are due to memory consumption.

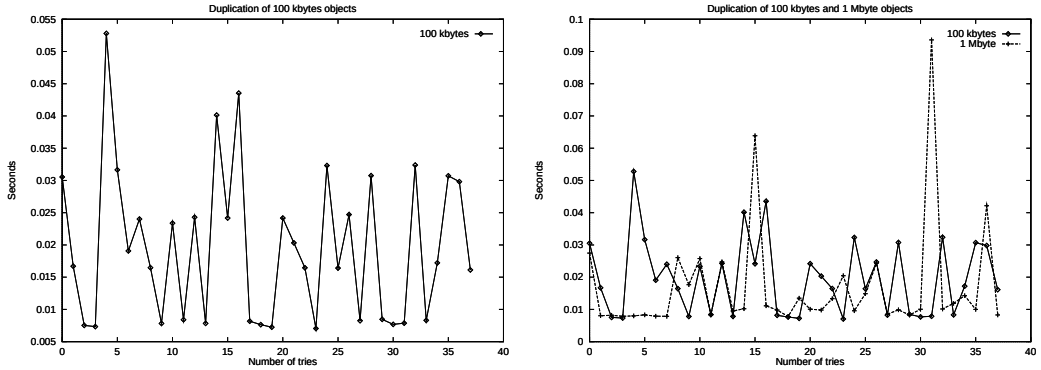
Table 4 gives an evaluation of round-trip times for a null method call. It has been tested with local calls on Blake (same process and distinct processes), and with remote call (server on Blake, client on Mortimer). Table 5 presents the same experiment with Lachesis. In this case, Blake was used as a client.

Table 6 presents the average time spent in strategy switching. Results were obtained from Blake and Lachesis. The test has been done on the replication control strategy. Figures are rather small nevertheless this is dependent of which strategies we switch: each strategy can have internal data or statistics to deal with. Strategy finalization or initialization will induce an additional overhead.



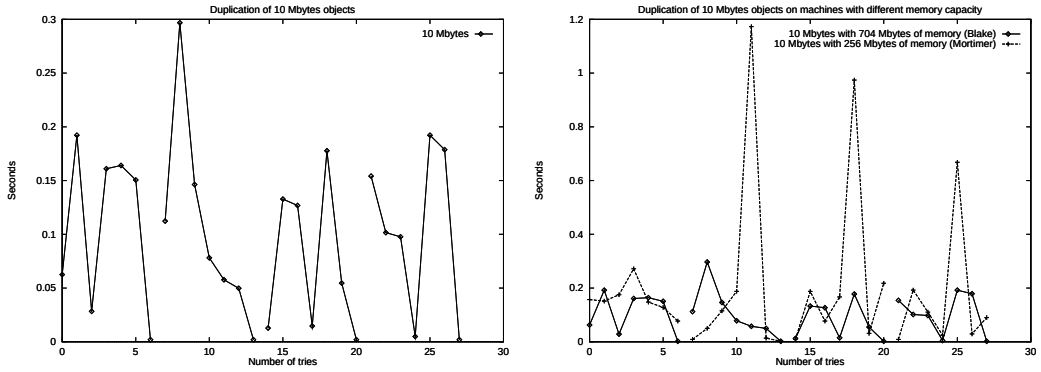
(a) 100 kbytes (b) 100 kbytes and 1 Mbyte

Figure 4: Small object duplication (Blake)



(a) 100 kbytes (b) 100 kbytes and 1 Mbyte

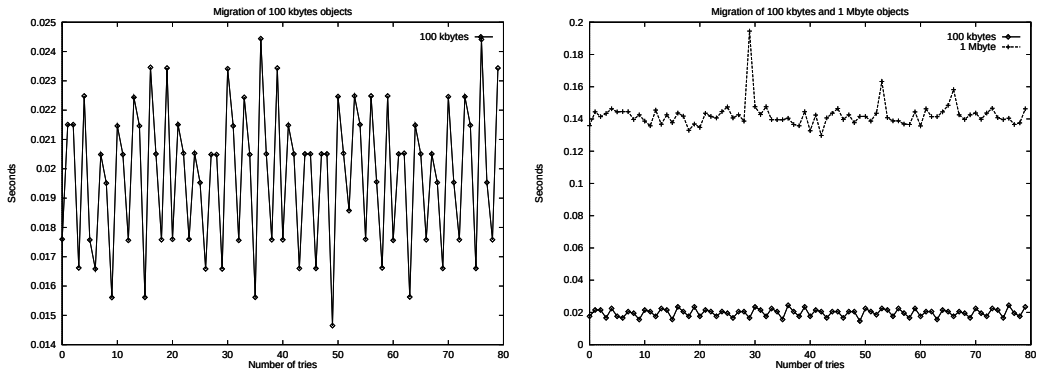
Figure 5: Small object duplication (Lachesis)



(a) 10 Mbytes

(b) 10 Mbytes on two machines

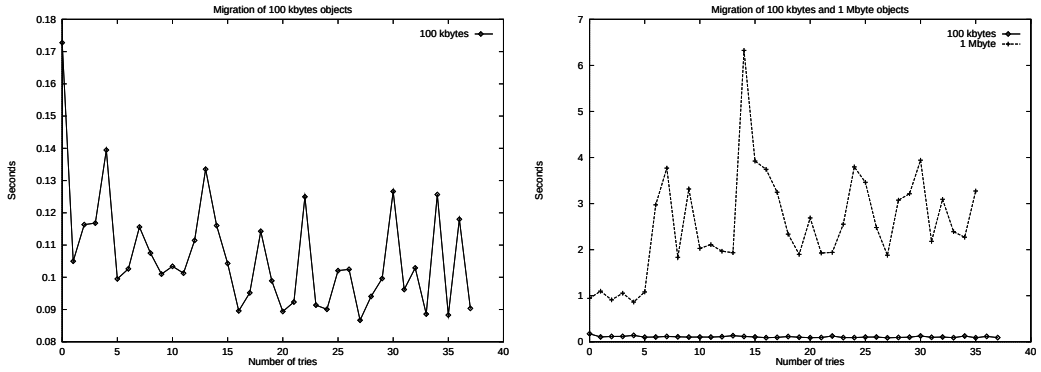
Figure 6: Big object duplication (Blake, Mortimer)



(a) 100 kbytes

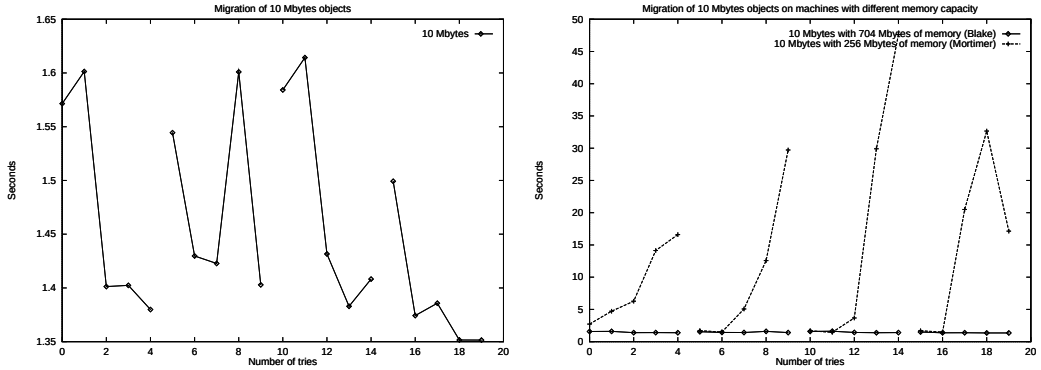
(b) 100 kbytes and 1 Mbyte

Figure 7: Small object migration (Blake)



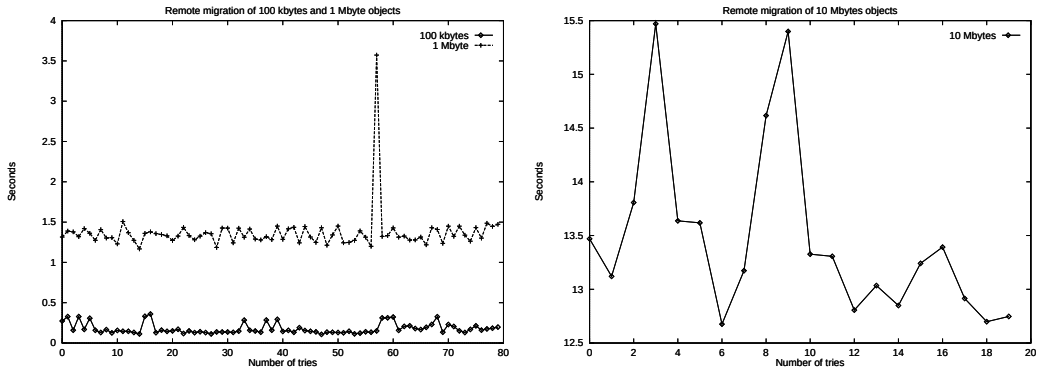
(a) 100 kbytes (b) 100 kbytes and 1 Mbyte

Figure 8: Small object migration (Lachesis)



(a) 10 Mbytes (b) 10 Mbytes on two machines

Figure 9: Big object migration (Blake, Mortimer)



(a) 100 kbytes and 1 Mbyte objects

(b) 10 Mbytes objects

Figure 10: Remote object migrations (Blake, Mortimer)

	Average	Highest	Lowest
Same process	0.01439628	0.04051300	0.01360600
Same machine	0.57311014	0.75116900	0.46903200
Remote machine	0.33798314	0.49121100	0.24732300

Table 5: Null method call round-trip time in seconds (Lachesis, Blake)

	Average	Highest	Lowest
Blake	0.00002859	0.00003910	0.00001951
Lachesis	0.01710896	0.22221500	0.01286100

Table 6: Replication control strategy switch time in seconds

6 Conclusions and future work

Cadmium aims at providing a system-level support for handling disconnection and mobility. We are mainly concentrating our efforts on object replication and adaptation to the environment.

We are using replication techniques along with reference management (flexible binding) in order to ensure availability of data even while disconnected. Our references and redirection protocols are based upon Cd SSP Chains.

We provide various strategies for access control, replication, etc. as well as hooks in the system to give applications a way of switching between strategies. In this we have an application-aware approach. Each application has the opportunity to chose whether it lets the system alone manage mobility problems or whether it can benefit from system-level information in order to best adapt.

Given the early Cadmium implementation, many aspects has to be polished. First of all, we will extend our library of strategies and refine the prototype. We also plan to deal with clusters: clusters can be generated automatically, i.e. without requesting the user interaction, based on the application semantics as well as its spying as it is done in Coda [4] or Seer [6]. For example, with a calendar application it could be possible to cluster all the appointments or all the meetings of a given research group.

Secondly, we plan various measures such as the responsiveness of our mechanisms for using strategies, switching and reacting to the notifications. This will be done according to network availability and bandwidth evolutions (soft or burst traffic). We intend upcall registration criteria (resource and thresholds) to highly affect replication efficiency. We will experiment our mechanisms with bandwidth thresholds, distance from some specific points or fixed stations, etc. Finally, availability of data and consistency are contradictory. Cache replacement and consistency strategies are supposed to provide the same behavior. We will measure the impact the replacement or consistency strategies on replication efficiency.

Thirdly, a number of open problems or lacks are still remaining in the current prototype. For example, we will probably need to tune our notification thresholds. To do so we may need a tool that will log state changes as well as the prototype reactions in order to compute the best thresholds from time to time. The results may be re-injected into Cadmium in order to adapt itself.

We currently did not worked on the combination of strategies neither on the possible conflicts that can appear between them. This include conflicts between strategies of different classes used along with the same object. For example a replication control and an access control strategies can have contradictory control patterns. This also include strategies that are used sequentially, i.e. after a dynamic strategy change. For example if we switched for a LRU cache replacement strategy to a FIFO, the data kept are not the same.

Many other interesting issues such as security, garbage collection, fault tolerance, etc. are pushed into far perspectives.

Acknowledgements

Special thanks to Marc Shapiro for his numerous comments, and to Fabrice Le Fessant for his help concerning his oCaml Virtual Machine.

References

[1] Baggio, A. Replication and caching strategies in Cadmium. Tech. Rep. RR-3409, INRIA, Apr. 1998. http://www-sor.inria.fr/~aline/.

[2] Baggio, A. System support for transparency and network-aware adaptation in mobile environments. In *ACM Symposium on Applied Computing, special track on Mobile Computing Systems and Applications* (Atlanta, Georgia, USA, Feb. 1998). http://www-sor.inria.fr/~aline/.

[3] Baggio, A., and Shapiro, M. Reconciling transparency with resource awareness in nomadic computing. In *4th Cabernet Radicals Workshop* (Rethimnon, (Crete), Sept. 1997). http://www-sor.inria.fr/~aline/.

[4] Ebling, M. R. *Translucent Cache Management for Mobile Computing.* PhD thesis, Carnegie Mellon University, Mar. 1998. http://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html, CMU-CS-98-116.

[5] Kistler, J., and Satyanarayanan, M. Disconnected operation in the Coda file system. *ACM Transaction on Computer Systems 10*, 1 (Feb. 1992), 3–25. http://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html.

[6] Kuenning, G., and Popek, G. J. Automated hoarding for mobile computers. In *Sixteen ACM Symposium on Operating Systems Principles* (Saint Malo, France, Oct. 1997), pp. 264–275. http://fmg-www.cs.ucla.edu/geoff/sosp97.html.

[7] Maisonneuve, J. *Hobbes : un modèle de liaison de références réparties.* PhD thesis, Université Pierre et Marie Curie, Oct. 1996. http://www-sor.inria.fr/publi/maisonneuve_thesis96.html.

[8] Mummert, L. B. *Exploiting Weak Connectivity in a Distributed File System.* PhD thesis, Carnegie Mellon University, Dec. 1996. http://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html, CMU-CS-96-195.

[9] Noble, B. D. *Mobile Data Access.* PhD thesis, Carnegie Mellon University, May 1998. http://www.cs.cmu.edu/afs/cs/project/coda/Web/coda.html, CMU-CS-98-118.

[10] Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., and Walker, K. R. Agile application-aware adaptation for mobility. In *Sixteen ACM Symposium on Operating Systems Principles* (Saint Malo, France, Oct. 1997), pp. 276–287. http://www.cs.cmu.edu/afs/cs/project/coda/Web/docdir/s16-reprint.ps.Z.

[11] Piumarta, I. SSP Chains - from mobile objects to mobile computing. In *ECOOP'95 Workshop on Mobility and Replication* (Aarhus, Denmark, Aug. 1995). http://www-sor.inria.fr/publi/SSPMobPP_ecoop95-mobility-pp.html.

[12] Rèmy, D., and Vouillon, J. Objective ML: An effective object-oriented extension to ML. *Theory And Practice of Objects Systems* (1998). ftp://ftp.inria.fr/INRIA/Projects/cristal/Didier.Remy/objective-ml!tapos98.ps.gz.

[13] Satyanarayanan, M., Noble, B., Kumar, P., and Price, M. Application-aware adaptation for mobile computing. In *Sixth ACM SIGOPS European Workshop* (Dagstuhl, Germany, Sept. 1994). http://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-ody.html.

[14] Shapiro, M. A binding protocol for distributed shared objects. In *International Conference on Distributed Computing Systems* (Poznan (Poland), June 1994). http://www-sor.inria.fr/publi/extended-ICDCS.300dpi.html.

[15] Shapiro, M., Dickman, P., and Plainfossé, D. SSP Chains: Robust, distributed references supporting acyclic garbage collection. Tech. Rep. 1799, INRIA, 1992. http://www-sor.inria.fr/publi/SSPC_rr1799.html.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399