Partitioning Applications with Agents

Oskari Koskimies

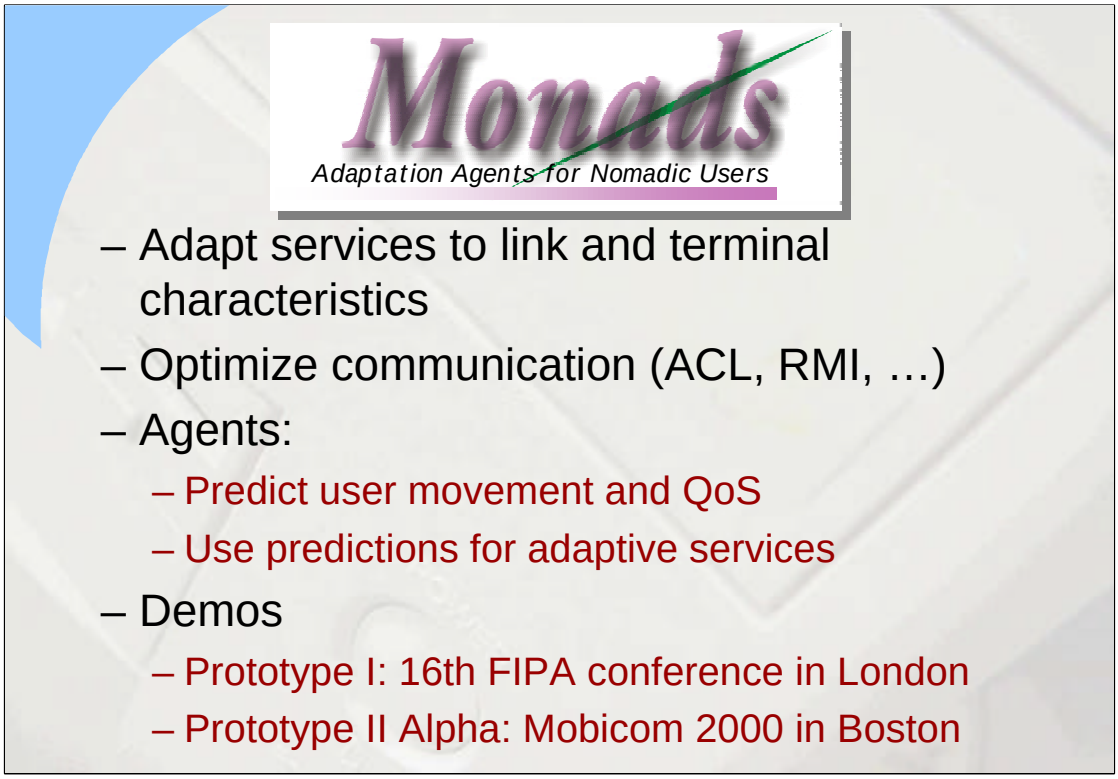
Monads Research Project

oskari.koskimies@cs.helsinki.fi

University of Helsinki

Department of Computer Science

Greetings everyone. My name is Oskari Koskimies, and I the subject of my presentation is Application Partitioning with agents. I work in the Monads project at the University of Helsinki.



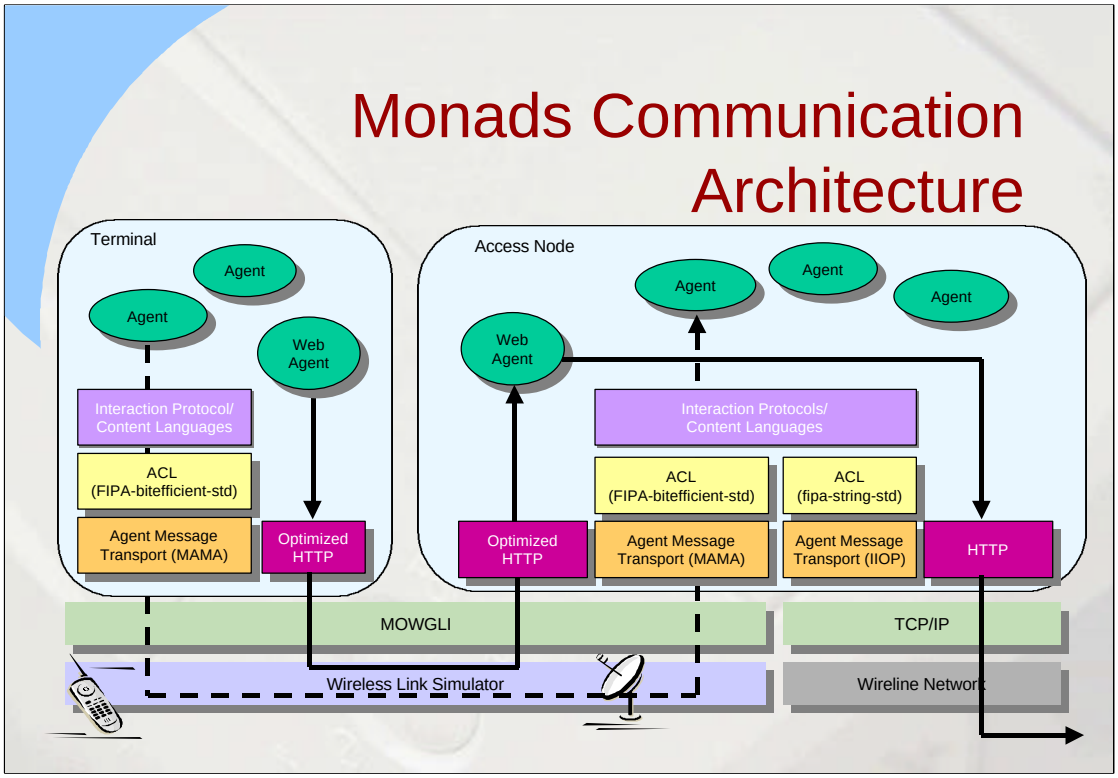
First I would like to say a few words about our project in general.

The Monads project targets nomadic, wireless users. In this kind of environment, both link and terminal characteristics vary from one end of the spectrum to the other. Thus, the goal of monads is to provide adaptive services that can offer optimal service in every circumstance. Note that optimal does not mean that the same service level should be maintained - that is impossible - simply that the service should be the best possible given the circumstances. Traditional applications are usually optimized for the most common case, and provide suboptimal service in other cases.

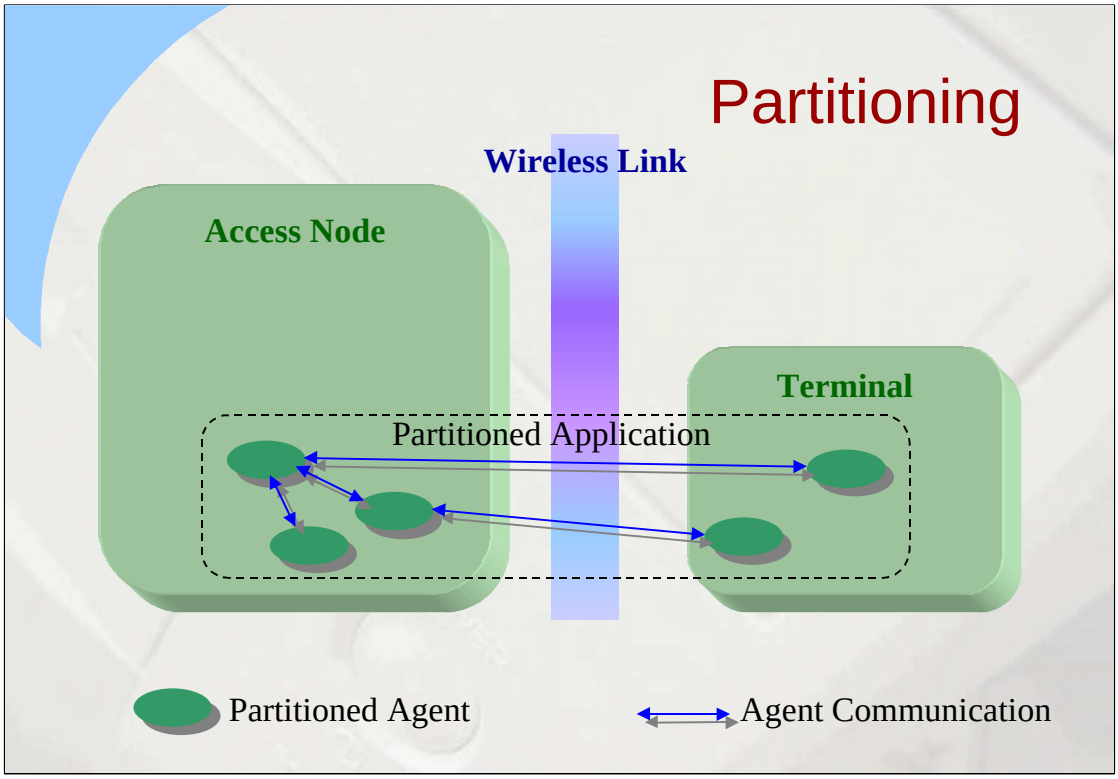
A common question is, “Why use agents?” In our case, the answer is simple. One of the goals of Monads is to evaluate how useful agents are for this problem area. Since our goal, adaptiveness, is also one of the main properties of agents, they seemed like a good choice. So far we have no complaints.

We optimize communication by using optimized versions of ACL and RMI. We use earlier results of the *Mowgli* project to optimize the transport layer.

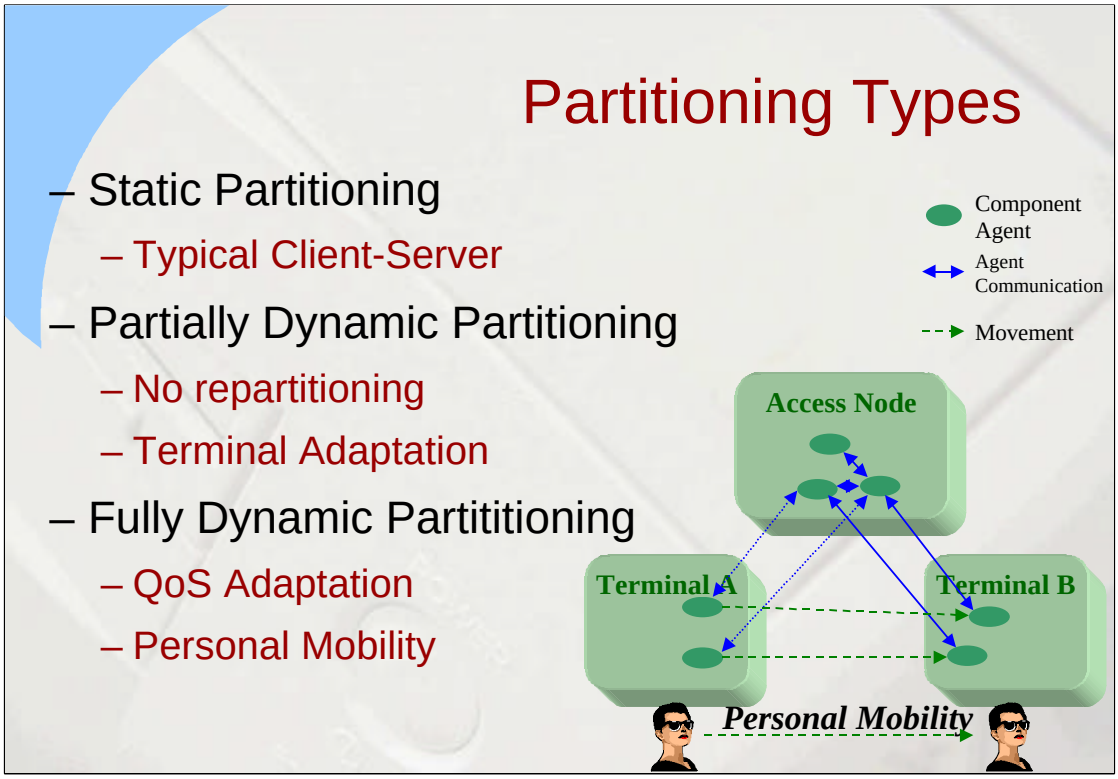
So far, we have implemented a system that uses agents for predicting user movement and Quality-of-Service. The predictions are used by other agents to provide adaptive services. Our system has been demonstrated twice: the first prototype was demonstrated in the 16th FIPA conference in London in January, and an early version of the second prototype was shown at the Mobicom 2000 conference last month.



Here is a brief look at the Monads Communication architecture. There is only one thing here which is of interest for this presentation, and that is the concept of an Access Node. It is a user's access point to the fixed network, and all traffic is routed via it. The access node is also where mediator software is placed. This allows optimized protocols to be used over the wireless link both on the transport and the application level.



With partitioning we mean dividing an application into component agents that communicate using for example FIPA ACL messages. There is nothing special about partitioning per se, every distributed application has been partitioned somehow. Actually, the communication architecture enforces partitioned software, since usually a mediator component is needed at the access node. Thus, even the client side of an application has to be partitioned between terminal and access node.



However, partitioning is usually *static*, meaning that the way an application is partitioned is frozen at compile time. Any traditional client-server application is an example of this.

This is in clear conflict with the dynamic nature of the wireless environment. Instead, we should defer the partitioning decision until the application is started. This allows an application adapt to the terminal it is running on, by choosing the partitioning that best suits the terminal's resources.

While this makes partitioning *partially dynamic*, it still does not take into account that circumstances may change even during an application session - for instance, the user may move out of WLAN range. In order to adapt to these kind of changes, an application has to have *fully dynamic* partitioning - it must be able to *repartition* itself at run-time.

Repartitioning allows adapting to changes that happen during the lifetime of an application session. The previously mentioned case of losing WLAN coverage is an example of a QoS change. An application can adapt to it by choosing a partitioning that is optimal for low QoS. However, by slightly extending the concept, repartitioning also allows support for terminal changes, or *Personal Mobility*. An application can be moved to another terminal by repartitioning it so that terminal side components are located at the new terminal. From an agent's point of view, relocating to another terminal is no different than relocating to the access node as it does during normal repartitioning. From partitioning point of view, the partitioning service of the new terminal has to take over the responsibility for the application.

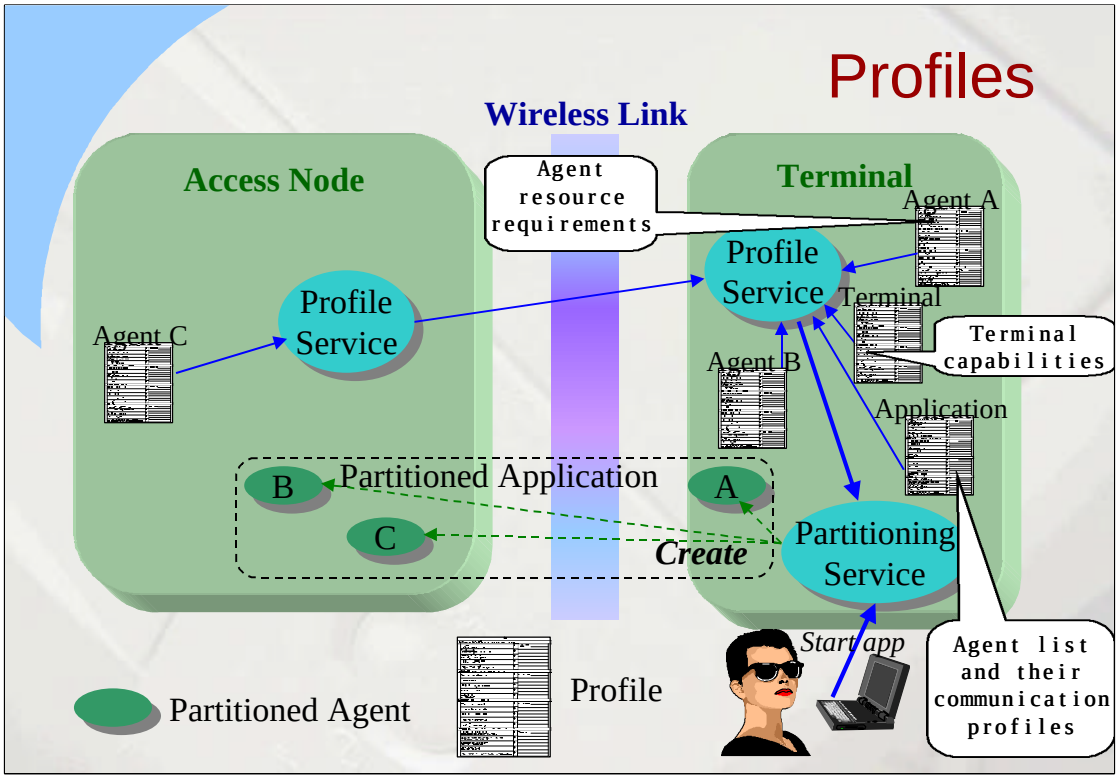
Application Start

- Profiles contain information about the agents that make up an application (size, communication patterns, etc)
- When asked to start the application, the system determines agent starting locations based on profile, terminal and (predicted) QoS, and sends create-agent requests
- Partitioning must be transactional

When an application is started, the partitioning service examines profile information about the application, the participating agents and the host machines involved.

Partitioning is a relatively heavy process. Since QoS varies, we need some idea of what the future QoS will be in order to select a partitioning that does not need to be redone immediately. For this we use the Monads QoS prediction agents.

It is worth noting that partitioning and repartitioning are transactional processes. All component agents must agree on whether partitioning was successful, and what the new partitioning is. Thus, partitioning is very much like a two-phase commit. I won't go into details like message sequence diagrams here, but they are all in the article.



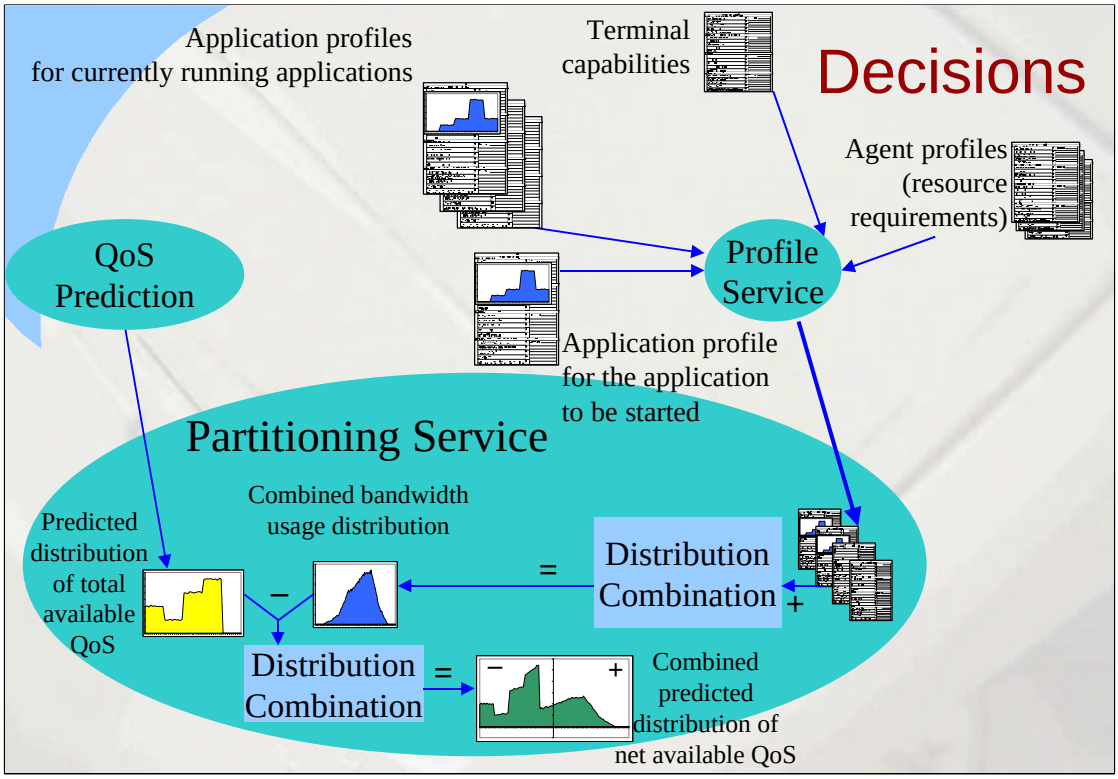
There are three types of profiles:

Application profiles list the component agents, and give a set of possible configurations. Each configuration contains agent locations and a communication profile, as well as an utility value that represents how good that configuration is from user point of view.

Agent profiles give the resource requirements for agents. They can also contain startup and movement costs. If an agent is stationary, that is noted here as well. Stationary agents will limit the way an application can be repartitioned, but in some cases an ersatz mobility can be achieved by starting a new agent at the destination instead.

Terminal profiles list terminal capabilities, such as memory and screen size.

Based on this information, the partitioning service selects a configuration and sends create-agent requests to the appropriate locations. But how does it make that decision?



Each configuration in the application profile gives a bandwidth usage distribution for that configuration. By adding together the usage distributions of currently running applications with the distribution of a configuration for the application to be started, we get combined usage distribution. This is deducted from the predicted available QoS, which we get from a QoS prediction agent. Thus we get the predicted distribution of net available QoS, *if* this particular configuration is chosen. By comparing the net QoS distributions for all configurations that are otherwise feasible, and weighing them against the utility values for the configurations, we can select an optimal configuration.

Okay, that was quite a lot to absorb at once. Would anyone like to make a question at this point?

Repartitioning

- The system monitors QoS, possibly also other resource consumption
- When circumstances change radically enough to warrant different type of partitioning (taking overhead into account), repartitioning is initiated
- This would typically happen after vertical handover (or when one is predicted)

In repartitioning, the previous decision-making process is rerun when QoS is predicted to change dramatically. Since the process is costly, smaller changes should be ignored. To further reduce overhead, one might only consider vertical handovers, which are guaranteed to result in a lasting, drastic change in QoS.

Use Case: Email

- Components:
  - User Interface agent (UDMA/Dedicated GUI)
  - Core Email agent
  - Filtering agent
  - Compression agent
- Configurations:
  - Desktop/Laptop, LAN
  - Laptop, WLAN
  - Laptop, WAN (e.g. GSM Data)
  - Palmtop, WAN / WLAN / LAN
  - Smartphone, WAN

Here is a use case to further explain the idea in partitioning. In order to have an application that everyone is familiar with, the example is sometimes a bit contrived. Please bear with me.

An email application consists of four component agents:

A *User Interface agent*, which may be either a dedicated GUI or a generic FIPA UDMA agent - by the way, our project has implemented such an agent.

A *Core Email agent*, which provides the basic application functionality

A *Filtering agent*, that groups and prioritizes messages, and handles any automatic email processing.

A *Compression agent*, which compresses messages prior to sending them to the terminal. This includes lossy methods such as leaving out attachments.

There are five possible configurations:

- A desktop computer or a laptop with a LAN connection

- A laptop with a Wireless LAN connection

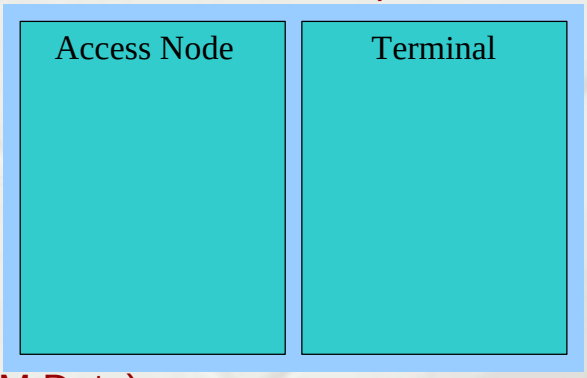
- A laptop with a Wide Area Network connection, for example GSM Data

- A palmtop with any type of connection - the terminal is the deciding factor here

- A smartphone that can only have a Wide Area Network connection.

Use Case: Email

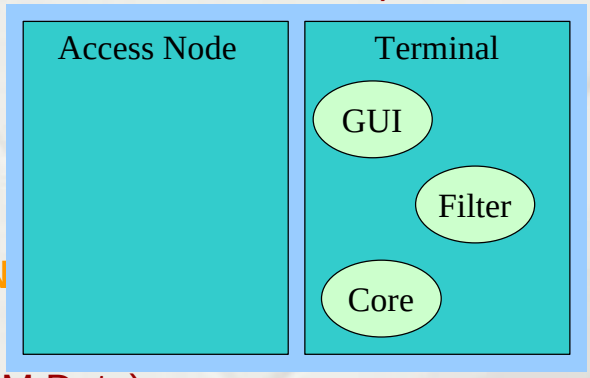
- Components:
  - User Interface agent (UDMA/Dedicated GUI)
  - Core Email agent
  - Filtering agent
  - Compression agent
- **Configurations:**
  - Desktop/Laptop, LAN
  - Laptop, WLAN
  - Laptop, WAN (e.g. GSM Data)
  - Palmtop, WAN / WLAN / LAN
  - Smartphone, WAN



I will now go through the configurations, and show how the application is partitioned in each case.

Use Case: Email

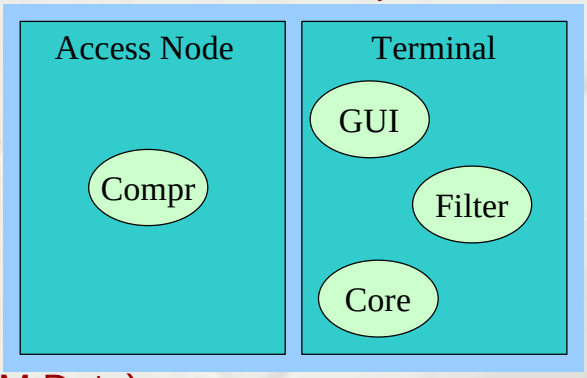
- Components:
  - User Interface agent (UDMA/Dedicated GUI)
  - Core Email agent
  - Filtering agent
  - Compression agent
- Configurations:
  - **Desktop/Laptop, LAN**
  - Laptop, WLAN
  - Laptop, WAN (e.g. GSM Data)
  - Palmtop, WAN / WLAN / LAN
  - Smartphone, WAN



In the first case, all the components are located at the terminal. Since a LAN is available, no compression agent is needed.

Use Case: Email

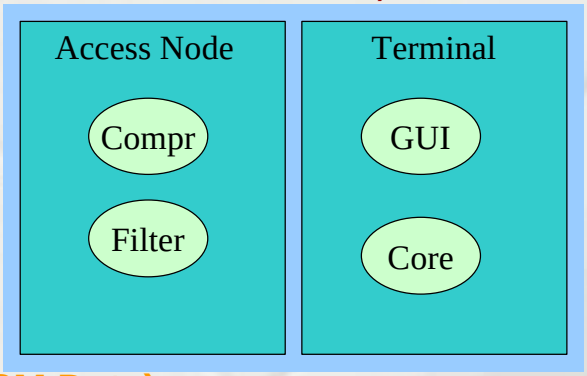
- Components:
  - User Interface agent (UDMA/Dedicated GUI)
  - Core Email agent
  - Filtering agent
  - Compression agent
- Configurations:
  - Desktop/Laptop, LAN
  - **Laptop, WLAN**
  - Laptop, WAN (e.g. GSM Data)
  - Palmtop, WAN / WLAN / LAN
  - Smartphone, WAN



In the second case, we still have the same class of terminal, but a slightly slower connection. Thus, in some cases we may want to compress data. An example would be a large attachment in a message from a mailing list. Also, the link speed may be down due to network load or bad radio conditions.

Use Case: Email

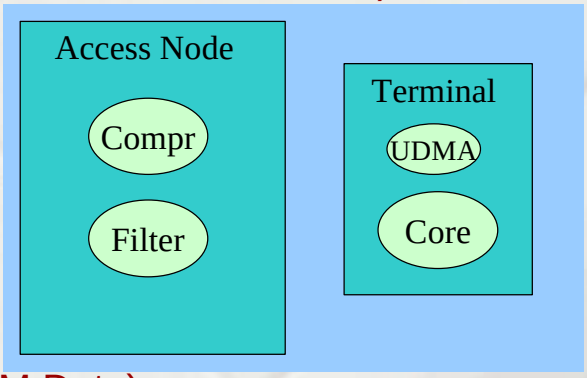
- Components:
  - User Interface agent (UDMA/Dedicated GUI)
  - Core Email agent
  - Filtering agent
  - Compression agent
- Configurations:
  - Desktop/Laptop, LAN
  - Laptop, WLAN
  - **Laptop, WAN (e.g. GSM Data)**
  - Palmtop, WAN / WLAN / LAN
  - Smartphone, WAN



With a very slow connection, we move the filtering agent from the terminal to the Access Node. While this makes it harder for the filtering agent to communicate with the user, disabling some interactive filtering strategies, it allows handling of some emails automatically, and when messages are prioritized on the network side the most important messages can be sent first.

Use Case: Email

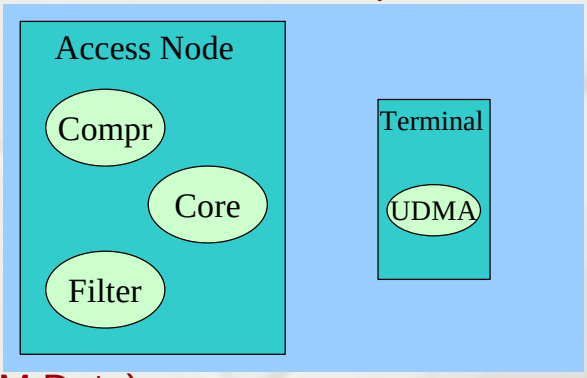
- Components:
  - User Interface agent (UDMA/Dedicated GUI)
  - Core Email agent
  - Filtering agent
  - Compression agent
- Configurations:
  - Desktop/Laptop, LAN
  - Laptop, WLAN
  - Laptop, WAN (e.g. GSM Data)
  - **Palmtop, WAN / WLAN / LAN**
  - Smartphone, WAN



With a palmtop we replace the dedicated GUI agent with a generic UDMA agent. Not only has the UDMA agent a smaller footprint, but it can also be shared by other applications, further reducing total memory requirements.

Use Case: Email

- Components:
  - User Interface agent (UDMA/Dedicated GUI)
  - Core Email agent
  - Filtering agent
  - Compression agent
- Configurations:
  - Desktop/Laptop, LAN
  - Laptop, WLAN
  - Laptop, WAN (e.g. GSM Data)
  - Palmtop, WAN / WLAN / LAN
  - **Smartphone, WAN**

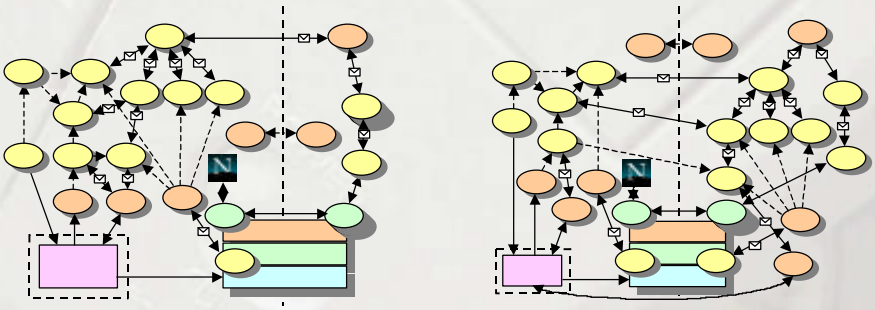


Finally, if we have a smartphone, we only run the user interface in the terminal. Effectively, we use the smartphone as an “almost-dumb” terminal.

Alternatively, we could move even the UDMA agent to the access node, and make it use WML forms for communicating with the user. This way, any WAP phone could use the application, at the cost of a more uncomfortable interface.

Partitioning in Monads

- Location of QoS prediction agents depends on terminal capabilities
- Configurations:
  - Minimal terminal resource use
  - Minimal bandwidth use
  - Compromises



As I said before, the Email example was constructed for illustrative purposes. Our project is not going to implement it. We have a better, from our point of view, use for partitioning.

In reality, QoS is not predicted by a single agent, although it looks like that for a client. Many agents collaborate to produce the predictions. It is not obvious how these agents should be located. One configuration is optimal from communication point of view, but places all agents at the terminal, stretching terminal resources. A configuration that reduces the number of agents at the terminal to a minimum wastes bandwidth. Between these two extremes, there are different compromises.

There is one problem, of course. In the initial partitioning, we do not have QoS predictions available, since the prediction system is not running yet! We have to select a configuration based on current QoS, and then repartition later if necessary. Fortunately, terminal resources is a more deciding factor here than QoS.

Summary: Benefits

- Adapt to terminal capabilities
  - Personal Mobility
- Adapt to available bandwidth
- Adapt to changing location of information sources (e.g. terminal location information)
- Load balancing

To summarize, the benefits of partitioning are as follows:

It allows adapting to terminal capabilities, including personal mobility

It allows adapting to available bandwidth

We can also adapt to changes in where information is available. For example, terminal location can be available either at the terminal (GPS), or at the fixed network (cell-based location). The part of an application that processes the information can be moved close to it.

Finally, we can do load balancing by repartitioning applications when a terminal starts to get short on resources.

Summary: Challenges

- Partitioning decisions require information about communication patterns
  - Communication profiles, learning
- Benefits depend on application being designed in a way that supports having different partitioning strategies for different environments

There are also some challenges. Making partitioning decisions requires detailed information about application communication patterns. While generic patterns can be stored into profiles, communication patterns are often partially user-specific. User-specific patterns would have to be learned by observing application behavior.

For an application to benefit from partitioning, it has to be designed in such a way that there are possible configurations for all the alternative environments.

While for simple applications it would be possible to make partitioning transparent, in the general case this is very hard or impossible. We have therefore chosen the approach that applications must be aware of partitioning. Our approach has the benefit of simplicity and increased application control, and the disadvantage of requiring specially-designed applications.

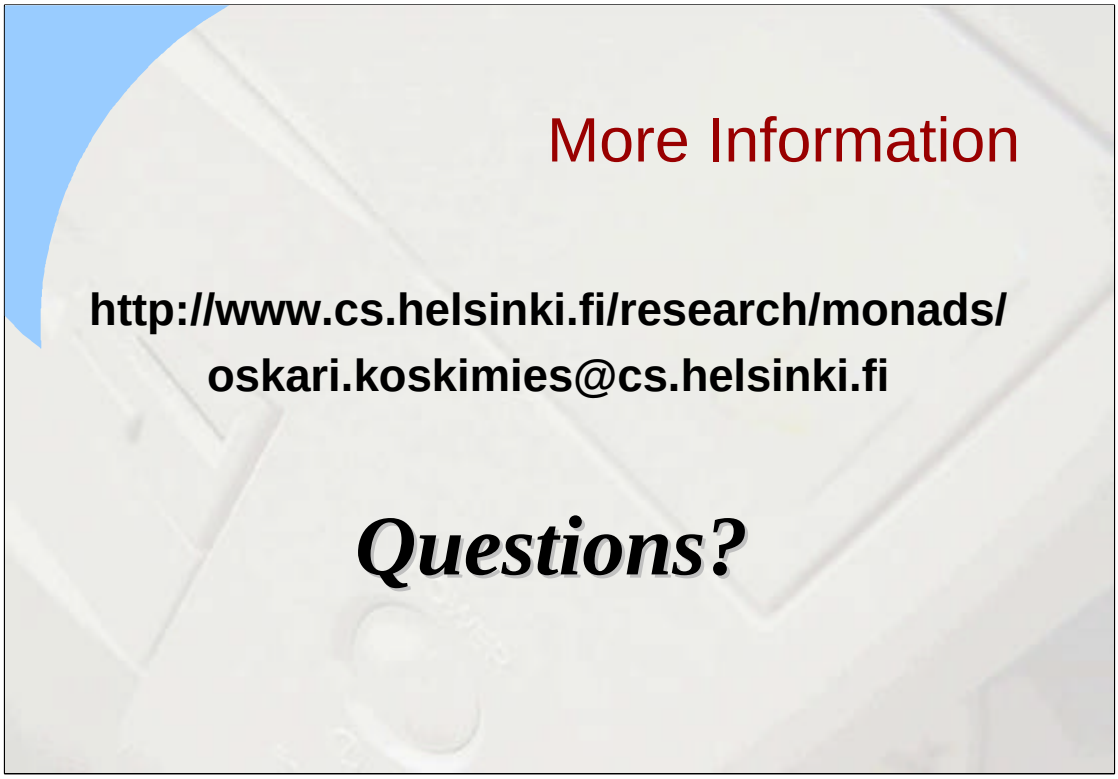
I must further stress that the component agents really must be agents, themselves capable of adaptation. For example, if the user interface of an application is switched from a dedicated GUI to a generic UI during repartitioning, agents probably have to change their behavior in non-trivial ways.

Current Status

- We are in the process of implementing Prototype II
  - Initial tests indicate that partitioning is a feasible method for adaptation
  - However, repartitioning requires agents to be very aware of partitioning
- Implementation and evaluation to be ready by the end of the year
- Papers on results to be written in early 2001

The implementation of prototype II is well underway - an early version has already been demonstrated, as you recall. However, it did not include partitioning. Initial tests indicate that we can use partitioning as planned. However, it also seems that repartitioning requires agents to be quite aware of partitioning. We are trying to reduce that.

According to schedule, implementation and evaluation of prototype II should be ready by the end of the year. Thus, you can expect papers on the results in conferences that have call for papers in early 2001.



For more information, you can consult the project website. It has many of our papers available. You can also email me.

Or, you can make a question now.