Towards Introducing Code Mobility on J2ME

Laurentiu Lucian Petrea and Dan Grigoras
Computer Science Department
UCC
Cork, Ireland
www.mccg.ucc.ie
Mobile Ad Hoc Networks

- Heterogeneous mobile devices
- No fixed infrastructure
- Transitory participation of members
- Limited bandwidth, high latency
- Difficulties in providing services, sharing resources and networks’ applications
Mobile Code Platform for Ad-Hoc Networks

- We reappraise the mobility
  - an ad hoc network will have 3 mobile elements: users, devices and software
Mobile Code – Features

- Naturally heterogeneous
- Execute asynchronously and autonomously
- Reduce network load
- Overcome network latency
- Encapsulate protocols
- Adapt dynamically to the environment
Systems Using Mobile Code for Mobile Devices

- LEAP (Lightweight Extensible Agent Platform)
  - a FIPA platform that can be deployed into a heterogeneous network of mobile and fixed devices, ranging from cellular phones to enterprise servers

- MIA (Mobile Information Agents)
  - develops an intelligent information system, which puts information of local relevance from the World Wide Web into the hands of a mobile user

- MAE (The Mobile Agent Environment)
  - is a mobile agent platform developed for resource-limited devices targeting m-commerce applications

- kSacI
  - is the transformed version of the SACI platform for the kVM covering the J2ME restrictions
Conclusions about Mobile Agent Platforms

- Diversity of the devices - from desktops and laptops to PDAs and mobile phones
- Fragmentation of software offering very different services
- Java is commonly used for developing these distributed services – J2ME
- Currently, there is no mobile code platform that achieves code mobility for light devices
No Mobile Code Platform on Light Mobile Devices – Why?

- The CLDC/J2ME limitations
  - No user-defined, Java-level class loaders
    - the CLDC implementation must have a built-in class loader that cannot be overridden or replaced by the user
  - No reflection features
    - no support for RMI
    - no object serialization
Mobile Code Platform for Ad-Hoc Networks – Our Design

- Consider a J2ME-based implementation of the mobile code platform
- The platform provides adaptability to heterogeneous systems using the mobile code
- The platform provides mobile code for information management
- The platform provides an API for developing services based on mobile code
Achieve code mobility in CLDC/J2ME

- CLDC is a platform for devices with limited resources
- Data are inherently scattered in the network of such devices
- Devices have a limited amount of storage space
- Network services request automatic deployment or update features
- New remote class loader for the CLDC
Remote class loading raises the question of trusting the code

Security Model evolved from JDK1.0 to Java 2

The Java 2 security model - a system level policy defines access permissions for executing code grouped into protection domains
Access Control

- Based on stack inspection
  - real systems such as JVM or CLR using it
- Based on execution history
  - to the best of our knowledge there is no real system that uses it
- Base on information-flow control
  - proved to be too restrictive and not practical
Access control based on stack inspection

- Method calls are recorded in program stack
- Each call has associated a set of permissions
- The algorithm inspects the stack and checks that the requested permission is contained in the associated set of permissions
Access control based on stack inspection

- Could be introduced to JDK 1.0 and JDK 1.1 without changing the underlying JVM
- The algorithm is difficult to be evaluated
- Brings limitations on the optimizations techniques
  - method inline
  - tail recursion
  - inter-procedural optimization
Access control based on stack inspection - Implementation

- Security passing style
  - transformation of code - adds an extra argument to every method of the application
  - cost of $O(1)$, but an overhead is introduced for each method call for passing the extra argument
  - difficult to implement due to native methods, reflection, bootstrapping sequence, inheritance
  - performance is roughly similar to the Sun’s implementation

- Using multiple access control matrices
  - checks involved in algorithm have been proven to be an NP complete problem
  - introduce dynamically changing for security policies
Access control based on stack inspection - Implementation

- Using an in-lined reference monitor
  - transformation of code – the monitor is merged into the Java applications to enforce the security policy

- First implementation
  - uses security passing style approach
  - performance is worse than of the Java resident implementation for most of the cases
  - performs better only in the case of having many permissions checks relative to the number of method calls

- Second implementation
  - uses the access to the JVM call stack
  - performance is close to Java resident implementation
  - brings more flexibility about what policies can be enforced
Access control based on execution history

- Permissions of the code are determined by inspecting the attributes of the code that has run before.
- Static permissions are associated with objects at loading time and represent the maximal rights of that code.
- Current permissions are associated with each execution unit during the run time.
- When a method is executed the current permissions are updated by intersecting the current permissions with the static current permissions.
- Updates are performed at calls and returns of methods.
- Advantages:
  - enabling optimizations currently not available for the stack inspection model
  - caller is protected from the callee as well
  - permissions could be represented as a variable
Access control mechanism for the CLDC

- Number of valuable resources is limited
- Minimum number of remotely loaded classes
- Limited number of protection domains to deal with
- Changing policies on light devices is a seldom event
Proposed solution

- Drop the security passing style model
  - code is transformed and it brings an overhead time in all code execution
  - did not perform better than the Java resident implementation
- Drop in-lined reference monitor
  - transformation of the code does not worth because there are not many permission checks relative to the number of method calls
  - security checks are seldom events on the CLDC
- Consider history-based algorithm
  - a smaller amount of permissions of a simpler security policy is CLDC case - can be efficiently expressed as one global variable
  - easier way of being formulated
  - the callers are protected from the callees as well
Automatic update mechanism suited for the CLDC

- First solution is to introduce in the virtual machine a check whenever a method is called
  - disadvantage - constantly check the identity of the method calls
- The second solution is to perform a static analysis of the code
  - disadvantage - the code has to be available before execution
Automatic update mechanism suited for the CLDC

- For each class an ID for code identity is provided
- For each method of the class, all calls inside the method are checked to see if they belong to a class of which identity can be determined
- If the callee’s identity is different from that of the caller, an update of current rights is introduced
Automatic update mechanism suited for the CLDC

- Each class can have a list named *update method list*, where other classes can be registered.
- If the callee’s identity cannot be determined, a *special mark* is introduced; the current class is registered in the *update method list* of the callee class for later verification.
- For all the classes registered in the *update method list*, the *special mark* previously introduced will be replaced; if the identities of the two classes are different, then an update of the current rights will be introduced. Otherwise, the *special mark* will be removed.
Future work

- Performance evaluation for the Java virtual machine using the remote class loading algorithm
- Define and evaluate various security policies for the CLDC
- Design an efficient way of representing the rights of the CLDC system resources
End of presentation

Thank you!

Questions?