**Research Part**

1A) Five other languages that can be executed in JVM are

1. Java Script
2. Python
3. Ruby
4. Prolog
5. Scheme

2A) Most programming languages today are hardware-platform specific. These programs are known as proprietary because they can only run on specific operating systems. The Java® virtual machine is the special element that makes the Java® programs platform independent. This JVM is an automated liaison process that manages communication between computer hardware and Java® programming code.

Using a Java® virtual machine has several benefits. The technical nuances of each operating system are hidden from the developer. This allows more versatility for the program because it can work with multiple computer hardware platforms. The JVM also allows the flexibility of using more then one operating system platform within the organization. There are several operating systems available today, including Windows®, MAC®, UNIX® and Linux®. A single Java® software program can run on multiple versions of operating systems. This is because Java® programs use the JVM to interact with operating system. Other types of software code typically require a separate compilation code base for each operating system.

References - <http://www.wisegeek.com/what-is-a-java-virtual-machine.htm>

<http://en.wikipedia.org/wiki/Cross-platform>

3A) The Java Virtual Machine is primarily designed for transporting Java programs. As a consequence, when JVM bytecodes are used to transport programs in other languages, the result becomes less acceptable the more the source language diverges from Java. Microsoft’s .NET transport format fares better in this respect because it has a more flexible type system and instruction set, but it is not extensible, and (for example) has no provision for supporting explicit programmer-specified parallelism. Both platforms have difficulty making transported programs run efficiently.

We present two techniques which provide parts of the envisioned language-agnostic functionality. Compressed abstract syntax trees as a wire format provide for a very dense encoding of programs at a high level of abstraction. We show how to parameterize the compression algorithm in a modular fashion with knowledge beyond the purely syntactical level. This leads to the notion of well-formedness by construction. The second technique defines the semantics of programs by mapping from abstract syntax trees to a typed core calculus representation. Based on this representation it becomes possible to use portable definitions of security policies and to execute programs written in different source languages, even if a more efficient trusted native compiler is not available on the target platform.

References - http://research.microsoft.com/en-us/um/people/nick/entcs/haldar.pdf