Mobile code architecture

Definition

- Mobile code style can be defined informally as the capability to dynamically change the bindings between code fragments and the location where they are executed. In general, the mobile code style provides mobility to transfer code inside networks and to allow resource relocation inside the network
- Vocabulary for components and connectors:
    - Components (code interpreter, execution dock)
    - Connectors (network protocol, code/data packaging for transmission)
    - Data (Code, program state, data for the code)

Topological constraints

- Remote evaluation
    - This covers any code that is executed on a remote machine, with the results being sent back to the client. The execution can be moderated, meaning it is observed by the client, or unmoderated, without the client's supervision. An example is a coding challenge website like Leetcode, where you write a program locally in the browser and then it is sent to a remote server for execution. The server then returns the results which indicate how many of the test cases your program executed correctly.
- Code on demand
    - This is a commonly used technology where executable code is sent from a server and run on the client's computer. Most people will encounter this kind of style when browsing the internet, where javascript code is downloaded from a server and then run locally.
- Mobile agents
    - This is some software and data that is able to move from one computer to another autonomously, while continuing its execution. It is able to copy a saved state of itself to a new machine and resume execution at will. Some benefits are reduced network load from not having to transfer files, asynchronous execution, flexibility, reduced compilation time, and increased portability. A common mobile agent is a computer virus, that is able to spread and run itself over a network.

Applicable problems

- System has resources but not the code:
    - The first type is the case where one system has the resources needed to do a task, but not the code. In this case Code on demand is the solution, as it can request code from the server and execute it locally. This saves space on the server by not storing and sending those resources everytime, making it much more scalable. An example of this is a web server sending the html & javascript code to a home computer, which have access to resources like cookies and the users screen.

- System has code but no resources:
    - Another type is when a system has the code it wants to execute, but not the resources to run it. This is a good use for remote evaluation. The client can delegate the execution to a remote server which has the resources to run the code. This allows code to be worked on and created on many different systems, but be executed from anywhere on the same environment.

Resilience to change

- The use of the mobile code style often results in an architecture that is very resilient .
- Both the code on demand and remote evaluation archetypes allow for the client and the server to treat each other as a black box.
- As long as the interpreter/compiler is working correctly, the client/server can be switched for another with ease. With the mobile agents, new nodes can be added and removed at will, as long as they have the ability to run the mobile agent.

Negative behaviours

- Inconsistency in the execution environment:
    - One negative behaviour of this architecture sprouts from the fact that it can't always know what environment it will be run on. As such it needs to define what kind of environments it should be run on, and make sure that it can support all of them. Take browsers as an example, different browsers interpret and run HTML & JS differently, so if a website wants to support all browsers, then they need to add special cases and test on all browsers to ensure expected behaviour.
- Uncertainty about security:
    - Another problem that arises from this is the security issue mentioned before. Depending on the implementation and use, the sending system might not be able to know whether or not the remote system is safe or not. This forces developers to develop as if the remote system is unsafe by default, which limits certain features.

Supported NFPs

- Supports scalability:
    - One of the biggest benefits of this style is scalability. Especially in the context of remote evaluation. It allows a central main controller to quickly delegate tasks to a large number of general (non specific) computers. This can be useful for large calculations as different parts can be run in parallel on different machines. It is also useful for code on demand, since it allows the client to take a small part of the computational load, which would add up to a huge load for the server if it had to do it itself.

Inhibited NFPs

- Inhibits security:
    - One issue that this style might face is in the realm of security, especially when the system remotely running the code is not closely monitored. Since the server is sending the remote system code to be executed, this code will be, at some stage, visible to the remote system.This could allow a bad agent to gain insight into how the system functions, or even intercept the code at the endpoint and make changes that the sending system doesn't expect. As such this code cannot contain sensitive information, and any information sent back from this code must be checked again to ensure it is not invalid.

Comparison with other architectures

- In Mobile code, plug in and black board architectures, the components are not coupled.
- In Mobile code, plug in and black board architectures, the data is in one place.

Credit:

Students:

Super Sirius Team (Zhuoxi Li, Ricky Zhao, Jiaming Zhang, Tong Li)

AAT (Adam Kenneweg, Alexander Douglas Bourqe, Tom Taubkin)

Instructor and TAs:

Mei Nagappan, Achyudh Ram Keshav Ram, Aswin Vayiravan, Wenhan Zhu