JAIST Repository
https://dspace.jaist.ac.jp/

Title	大規模シミュレーションのためのスマートデバイスアプリケーションの構成法
Author(s)	中川, 颯馬
Citation	
Issue Date	2025-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/19832
Rights	
Description	Supervisor: 篠田 陽一, 先端科学技術研究科, 修士 (情報科学)

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

A Structure for Smart Device Applications for Large Scale Simulation

2210126 Soma Nakagawa

In times of disasters or emergencies, the number of people using smart device applications (App) can increase dramatically. For example, in the case of massive Nankai Trough earthquake, as many as 8.8 million people are expected to be affected. If a large number of these people simultaneously use a disaster-response App, it is essential for App to continue functioning as usual. To ensure this, performance testing and large-scale deployment simulations must be conducted in advance. This need is especially critical for mission-critical App designed for use during disasters. Although existing load testing tools can be used for large-scale simulations, the loads generated by these tools are typically based on predefined and static scenarios. This makes it difficult to replicate the complex conditions of actual usage. By using App themselves to generate loads, it becomes possible to create more complex scenarios that better reflect real-life situations. This approach enables a more realistic evaluation of App performance under diverse conditions.

Existing methods for simulating large-scale App deployment include approaches that use physical devices and approaches that use emulators. When relying on physical devices, either on-premise devices or cloud-based services can be used. However, this tends to be impractical from a cost or resource perspective. Although emulators can accommodate larger scale than physical devices, they require significant overhead to emulate hardware features such as displays, input devices, and sensors. As a result, simulations on the order of millions of instances become difficult.

To address these challenges, this research proposes converting an App into a headless App (vApp). This allows the App to operate without depending on physical devices or emulators, thereby simplifying large-scale simulation. One straightforward way to remove the frontend entirely would leave only the core logic. However, this approach makes it impossible to test user input logic. Therefore, this study proposes rebuilding the frontend using a design pattern commonly employed in App with GUI. This allows for generalization of the elements that need to be rebuilt. MVC (Model-View-Controller) model is adopted here as an example for frontend rebuilding. During the process of making App headless, three major points were considered.

The first point involves an automated control mechanism for vApp. Since vApp has no GUI, existing methods such as touch operations or screen transitions are unusable. Moreover, manual operation is impractical for large-scale simulations. Consequently, an agent-oriented simulator (Agent) was introduced to simulate user operations and behavior scenarios. This enables vApp to be controlled automatically.

The second point concerns the design of I/O model for vApp. In a large-scale simulation, each Agent is paired with one vApp. However, since vApp has no GUI, conventional input and output mechanisms, such as touch operations or screen transitions, cannot be used. To address this limitation, operations and information must be exchanged through message communication between Agent and vApp. This communication mechanism is defined here as “I/O model”. Two models are proposed to facilitate this interaction. The first is vScreen Model, which focuses on screen structure and transitions. The second is vOps Model, which places more emphasis on the services provided by App.

The third point addresses a sensor information exchange model. When App runs on a physical device or emulator, it can directly obtain information such as GPS data or battery status. However, because a vApp does not run on an actual device or emulator, these types of sensor information do not physically exist. To overcome this limitation, this study leverages the fact that Agent simulates user behavior. Agent pseudo-generates various sensor data and sends it to the vApp. This mechanism allows vApp to behave as if it were operating on a physical device. Two models are proposed for exchanging sensor information between the vApp and Agent. One model handles data asynchronously (Asynchronous Model), while the other exchanges data synchronously (Synchronous Model).

To verify the effectiveness of the proposed approach, an example App was implemented as vApp using MVC pattern. Partial rebuilding of GUI in that vApp resulted in App variant. This made it possible to clarify the implementation differences and confirm the reproducibility of the method.

A comparative evaluation was then conducted using five different methods that combine various execution environments, forms of App, and operation methods. These methods were evaluated based on computing resources, required time, execution costs, ease of automation, ease of debugging, scalability, fidelity, and implementation overhead. The results show that the proposed method demonstrates clear advantages in many of these areas. Regarding computing resources and scalability, executing lightweight vApp on general-purpose servers enables simulations on the order of one million instances at reasonable cost and within reasonable time. In terms of execution costs and time, substantial savings were observed compared to other methods. Notably, the proposed method achieved approximately twenty-four times better cost efficiency than emulator-based approaches. Removing GUI also simplifies automation. Since there is no need to track screen transitions or user input, the control program becomes less complex than existing automated tools.

On the other hand, there are indications that the method may be inferior

in terms of fidelity, as it is difficult to reproduce all user interaction patterns or device-specific behaviors. Additionally, implementation overhead remains an issue, particularly due to the need to restore GUI if vApp is converted back into a fully functional App. Future work includes developing an automatic mechanism for generating vApp from existing App and refining Agent to simulate more detailed user behaviors. Further research will also involve conducting large-scale experiments with millions or tens of millions of simulations and clarifying which types of App and requirements are best suited for this method.