A Cost-Effective Approach to Software-in-the-Loop Simulation of Pervasive Systems and Applications

Giacomo Brambilla¹, Alessandro Grazioli¹, Marco Picone¹, Francesco Zanichelli¹, Michele Amoretti²

¹: Dept. of Information Engineering, 2: SITEIA.PARMA — University of Parma, Italy
giacomo.brambilla@studenti.unipr.it, grazioli@ce.unipr.it, {marco.picone, francesco.zanichelli, michele.amoretti}@unipr.it

Abstract—In this paper we illustrate a cost-effective software-in-the-loop simulation methodology, which is particularly suitable for testing large-scale pervasive systems and applications. The main advantage of such an approach is that real code has to be deployed once, and a general-purpose simulation engine replicates it over virtual devices and environments as often as we wish. Our implementation of the proposed methodology is illustrated by means of the software-in-the-loop simulation of a peer-to-peer information sharing system, with real code running on simulated mobile nodes.

I. INTRODUCTION

Pervasive systems and applications, due to their complexity, need careful analysis and test, before being deployed to target environments. Consider, for example, smart city applications [1], requiring the coordination of a huge number of networked software entities, interfaced with sensors, actuators, computational and storage facilities. Testing them in a controlled environment, like a laboratory, is not sufficient to find and evaluate the possible properties/issues of such complex systems — in particular, emergent ones, i.e., those that cannot be inferred from the analysis of single components, and appear when components interact. On the other hand, the simulation of large-scale systems is usually based on approximated models and simulation-specific code, which are not representative of all system details.

In this paper we propose a cost-effective software-in-the-loop simulation methodology, which allows to test deployment software on simulated devices and environments. Thus, code reuse is maximized, and the only effort is to create adapters to integrate the deployment code with the underlying platform, consisting of a general-purpose simulation engine, provided with more specific (still reusable) packages — e.g., to simulate mobility, communication, logging). The main advantage of such an approach is that real code has to be deployed once, and the simulation engine replicates it over virtual devices and environments as often as we wish.

The paper is organized as follows. In section II, we discuss some of the most relevant simulation approaches for pervasive systems and applications. In Section III, we present our software-in-the-loop simulation methodology. In Section IV we describe an implementation based on the general-purpose simulation environment called DEUS [2], and we illustrate the simulation of a vehicle-to-vehicle information sharing system, with deployment code running on simulated mobile nodes. There, we quantify the coding cost of the proposed approach, in terms of extra code lines specifically written for a large-scale simulation test. Finally, in Section V, we conclude the paper with a discussion of open issues and future work.

II. RELATED WORK

Various programming frameworks have been proposed to support the development of pervasive computing applications [3], [4], [5]. However, they require fully-equipped pervasive computing environments.

The development of software modules to be installed on resource-constrained devices, such as nodes of wireless sensor networks, is often supported by device emulators [6]. When the testing phase is completed, the application code can be installed on real devices as is, and its logic does not require further debugging. Unfortunately, the number of nodes that can be emulated at the same time is usually very limited. Thus, testing in the large requires simulation.

Bruneau and Consel have recently proposed the DiaSim simulator [7], which targets applications based on sensors and actuators, deployed in physical environments, involving users. DiaSim enables the simulation of the application logic of such applications, but does not simulate the other components of a pervasive computing environment. E.g., it does not provide any support to estimate physical aspects of an environment (such as the thermal modeling of a room), to simulate the network traffic between application components, or to model the behavior of application users. DiaSim is parameterized with respect to a high-level description of the target pervasive computing environment.

PerSim is an event-driven, interactive simulator, which allows to design a pervasive space and fit it with the desired sensors relevant to a specific scenario or application [8]. The simulator provides a web-based interface to enable incremental (long-live sessions) and collaborative design of pervasive space simulation projects. Researchers can design the space in terms of sensors, actuators, activities, and effectively generate data by changing and fine tuning simulation parameters. The authors have proposed an XML-based standard for Sensory Dataset Description Language (SDDL), alternative to the more common Resource Description Framework (RDF).
With respect to these solutions, our approach focuses on (1) seamless integration of deployment code over simulated devices and environments, (2) high modularity, as specific simulation packages can be plugged/unplugged over a general-purpose simulation engine, (3) scalability, as the number of replicated devices can be deterministically or stochastically defined and increased/decreased by the simulation engine, according to high-level system specifications.

III. PROPOSED METHODOLOGY

The software-in-the-loop simulation methodology we propose allows to test deployment software on simulated devices, immersed in simulated environments. On top of the computing host, a general-purpose simulation engine is installed. More specific simulation packages are installed, and integrated with deployment software for pervasive applications. The only effort that is required, for each different test scenario, is to develop lightweight adapters to integrate deployment code with simulation packages.

Figure 1 compares the proposed software-in-the-loop simulation methodology (illustrated by the layered diagram on the left) with on-field testing (shown by the diagram in middle) and traditional simulation (presented by the diagram on the right). The main advantage of software-in-the-loop simulation is that it requires just one deployment of the software to be tested, as the simulation engine generates and manages $N$ virtual nodes, providing the illusion of multiple replications. On-field testing, conversely, requires $N$ deployments of the code to be tested. Traditional simulation, instead, provides $N$ virtual nodes with virtual code, i.e., a simplified version of the deployment code. Usually, traditional simulation is convenient in terms of execution time, with respect to software-in-the-loop. However, its reliability is much lower.

For example, consider a point-to-point message transmission, requiring the execution of a `send(msg)` function provided by a specific socket-based API, such as Sip2Peer (http://code.google.com/p/sip2peer/). In this case, the adapter should expose the same function signature, but the related implementation would not be a socket-based message transmission. Instead, it should be an event scheduling on the simulation engine, with a timestamp in the future, computed according to a realistic delay model.

IV. IMPLEMENTATION

In our current implementation (http://code.google.com/p/deus/) of the proposed methodology, we adopt a discrete event simulation engine called DEUS [2], and three simulation packages, which cope respectively with node mobility, communication and logging. A reduced effort is needed to integrate deployment code with such packages, as it is sufficient to develop three adapters. Other packages could be included, e.g., to simulate the presence of sensors/actuators, or specific environmental events.

DEUS’ Java API allows developers to implement (by subclassing) (i) nodes, i.e., the entities which interact in a complex system, leading to emergent behaviors such as humans, pets, cells, robots or intelligent agents; (ii) events, e.g., node births and deaths, interactions among nodes, interactions with the environment, logs and so on; and (iii) processes, either stochastic or deterministic ones, constraining the timeliness of events. DEUS has been designed having in mind the three basic concepts listed above, and no specific modeling tool at all. However, DEUS nodes can be mapped to finite state machines, with transition functions that may be implemented either in the source code of the events that can be associated to the node, or in the source code of the node itself. Moreover, DEUS supports parallel (multicore and/or distributed) simulations [9].

The mobility package we have recently developed, called OSMobility, allows to simulate the motion of different entities, such as pedestrians and vehicles, in realistic geographical spaces. The main class of the OSMobility package is GeoPeer, which extends the basic Peer class provided by DEUS. A Peer is a network node, i.e., an entity (potentially) connected to other entities. A GeoPeer is characterized by a location, a way (e.g., road segment), and a set of neighbors,
i.e., GeoPeers on the same way or route. The class is extended by StationaryPeer and MobilePeer, which represent a static node (e.g., traffic light, roadside sensor) and a mobile node, respectively. A Way is a set of OSMNodes, which are points in the geographical space, and a Route is a sequence of Ways that must be traversed, in order to move from an OSMNode to another.

OSMobility is integrated with OpenStreetMap (OSM, http://www.openstreetmap.org) an open database which provides geographical data, such as road maps, for free. OSMobility uses such data to compute node trajectories, with a resolution degree which ranges from millimeters to kilometers. Thus, OSMobility allows to simulate vehicles running on highways or urban roads, pedestrians walking within Limited Traffic Zones, bicycles moving on cycling lanes, etc. The huge amount of information provided by OpenStreetMap allows also to take into account speed limits and to know where points of interest are placed, which is useful to create realistically located traffic jams. OSMobility uses PostgreSQL with PostGIS extension to store OpenStreetMap data, and pgRouting to generate routes. In practice, the OSMobility user has only to set endpoints, and routes are automatically generated. OSMobility's Router class, which provides an abstract method generateRoute(), has been specialized to DijkstraRouter, which uses Dijkstra's algorithm to compute the shortest route between two OSMNodes. In general, the best route may be defined according to any criteria.

The generic class for mobility models is SpeedModel, exposing get/set methods related to min/max/current speed. We have specialized such a class to implement the Fluid Traffic Model (FTM) [10], where speed is a monotonic decreasing function of vehicle density. The FTM is used to compute the next position of a vehicle, given its current position, speed, direction, and the density of surrounding vehicles. Such a computation is performed by the MovePeerEvent — whose instances are continuously generated and inserted in the event queue of the simulation engine, for every MobilePeer that is moving (as illustrated in Figure 2).

![Image](image_url)

Fig. 2. Sequence of MovePeerEvents in the event queue of the simulation engine.

The communication package provides several delay models which can be used to simulate message transmission between network nodes. The most simple model generates an exponential delay, whose expected value is computed from the message size and the nominal channel bandwidth. A more sophisticated delay model is the one we have obtained by means of highly detailed ns-3 simulations, assuming LTE communications [2]. In general, any delay model can be included in the communication package.

Deployment code running on a virtual node produces raw data (e.g., received request rate, service rate, success rate), which are placed in the shared memory or in a database. Scheduled by the simulation engine, the logging package picks such raw data and produces aggregated logs in machine-readable format — ready to be analyzed and used to generate graphs. The logging package is highly generic, providing basic primitives for storing and retrieving data in shared memory / database. Its specialized use is defined by the adapter that must be developed, in order to integrate logging package and deployment code.

A. Example

The methodology described in previous section can be applied to the evaluation of different pervasive applications. As an example, we illustrate the testing of a decentralized information sharing service, in the context of smart mobility. The related software can be installed on mobile devices, and used to produce, share and consume traffic information (such as accidents, traffic jams, detours). This is particularly useful for drivers, allowing to setup vehicular networks whose software is not embedded into vehicles — instead, it is portable from one vehicle to another, by the same user.

The software-in-the-loop simulation stack is illustrated in Figure 3 (left). On top of DEUS, we have set the OSMobility, communication and logging packages. To integrate them with the mobile application, we have developed three lightweight adapters:

- the first one allows the mobile app to access position, way, route of the simulated node, generated by the OSMobility package;
- the second one allows the mobile app to send/receive messages, by wrapping the communication package into a class which exposes the interface of a Sip2Peer node;
- the third one allows to pass mobile app state information (e.g., number of sent messages) to the logging package.

Long-term statistics are periodically logged to file. For live monitoring, OSMobility provides a visualization module (a related screenshot is shown in Figure 3 (right)).

To simulate a network with 1000 vehicles, with high message rate (1.5 messages per second, each message being 200 B large) and high mobility resolution (1 simulation event corresponds to a 20 m displacement), a single node — 2 GHz Xeon CPU, 16 GB RAM — of our cluster is sufficient. The duration of such a detailed simulation takes 10 hours, on average. Most of the time is devoted to the simulation of the message exchange between nodes. With the aforementioned hardware configuration, it is possible to simulate up to 4000 vehicles, in a reasonable time interval. For larger scenarios,
there are two options: either using a more powerful server, or adopting a parallel approach [9].

The main advantage of using OSMobility is that it supports a wide range of mobility models, including both traditional ones (such as FTM) and custom, user-defined ones. Compared to SUMO (http://sumo-sim.org/) a well-known microscopic traffic simulator, OSMobility provides dynamic vehicle routing. SUMO does not natively include such a feature, as routes have to be statically defined before the simulation is performed. Such a constraint can lead to unavoidable long trip times, or even to traffic blocks, for the low-priority roads which intersect high-priority ones. A number of frameworks exist to join SUMO and network simulators. Two notable examples are Veins (http://veins.car2x.org) and TraNS (http://www.isi.edu/nsnam/ns/) which respectively employ the OMNet++ and ns-2 network simulators. Both are based on the TraCI interface, which uses a TCP-based client/server architecture to allow the interaction with SUMO. Thereby, SUMO acts as a server and receives control messages to alter the simulator behavior at runtime. We plan to compare the performance of such a solution with our one.

Furthermore, the communication package exposes a Sip2Peer-like API, and frees developers from dealing with low-level communication issues, so that they can focus on the application-level functional requirements of the system of interest.

V. CONCLUSION

In this paper we have proposed a cost-effective software-in-the-loop simulation methodology, which is particularly suitable for testing large-scale pervasive systems and applications. Having a general-purpose simulation engine, and a set of task-specific simulation packages, the only effort that is required, for each different test scenario, is to integrate deployment code and simulation packages, by developing lightweight adapters. We have illustrated our current implementation of the proposed approach, based on the simulation environment called DEUS, and an example with real code for information sharing in vehicular networks.

Future directions of this research work are manyfold. We are interested in the definition of best practices (patterns) for the development of software adapters for the integration of deployment code and simulation packages. Moreover, we plan to extend our current implementation of the proposed software-in-the-loop simulation methodology, to use it for the evaluation of a wide range of information sharing protocols for pervasive systems, such as the highly scalable DGT [11], [12].

REFERENCES