SOLVING EXPERIMENT REPRODUCIBILITY IN AMBIENT INTELLIGENCE

Andrei-Adnan Ismail[1], Adina-Magda Florea[2]

Ambient Intelligence is an active research field that studies embedding computing elements into the environment - Weiser's concept of disappearing computer.

Practical experiments are complex to orchestrate, because of the technical expertise required in hardware, software engineering and machine learning. The entry barrier for running such an experiment is very high, requiring coordinated efforts at all levels in a research team, and an actual laboratory with sensors in order to collect the data.

In this work, we show how the power of Amazon's EC2 cloud can be leveraged in conjunction with a record and replay system for sensor data in order to allow researchers from anywhere in the world to run such experiments. The software infrastructure used for the experiments is open-source and freely available at `https://github.com/ami-lab/AmI-Platform`*. This platform is used in the AmI Laboratory (EF 210) of the University Politehnica of Bucharest.*

Keywords: ambient intelligence, cloud computing, software framework, experiment reproducibility

1. Introduction

Experiment reproducibility is a requirement when publishing an article in areas of research such as databases or operating systems. Research results are vetted by the community and further extended because there is a clear sequence of steps for anyone who wishes to execute it. In Ambient Intelligence, the situation is quite the opposite: most experiments are based on simulations and theoretical scenarios are one of the main ways to showcase importance of the results. In this work, we aim to improve this by proposing an open-source

[1]PhD Student, Department of Computer Science and Engineering, Faculty of Automatic Control and Computers, University Politehnica of Bucharest, e-mail: `iandrei@gmail.com`

[2]Professor, Department of Computer Science and Engineering, Faculty of Automatic Control and Computers, University Politehnica of Bucharest, e-mail: `adina.florea@cs.pub.ro`

software infrastructure[1] that allows researchers to record sensor data and to easily run experiments by using such recordings on top of the Amazon EC2 cloud.

One of the major obstacles in rapidly prototyping practical AmI applications is not being able to reuse the work done in other consacrated systems such as [20] or [4], leading to an impractical effort necessary for setting up even the most basic AmI experiment [8]. Several of the steps of this effort include:

S1 having access to a dedicated laboratory - a big administrative problem in itself, having to go through an approval and maintenance procedure at the university level

S2 having a sensor array and a scalable software infrastructure for consuming the data in a simple manner

S3 staying up-to-date with the latest trends in sensing platforms, buying them and integrating them into the system

S4 developing algorithms on top of this distributed system in order to implement different scenarios such as distributed tracking or context-aware communications

This can easily be a full-time job for several researchers, without obtaining too many results, as it requires multi-disciplinary skills ranging from identifying the important research topics at the moment to software engineering of robust components. The problem is even worse due to the limited ability of reusing software from other projects, as it is usually too specific to the scenario implemented [8]. Moreover, not all scenarios can be implemented with all types of sensors; since some scenarios require specialized sensors that take months to order and to install, the need for such a sensor can delay an ambient intelligence project significantly. And the need of such a sensor can come as the unexpected conclusion of months of experimenting with other sensors that fail to deliver on the expected results.

We propose a software infrastructure that has three basic capabilities: it can record and replay sensor data, it offers simple primitivies for consuming it in real-time and it is able to run experiments based on recorded data in the cloud, by automatically renting computing power from Amazon EC2[2].

2. Previous Work in Software Infrastructures for AmI

The existing platforms for developing Ambient Intelligence applications are focused on the following topics:

[1] `https://github.com/ami-lab/AmI-Platform`

- publishing software services in a marketplace structure, where end users who already own a deployed Ambient Intelligence system can discover and buy such services
- dissemination of information in a context-aware manner; subscribers to information buses should only receive information that is relevant to their context
- modelling interactions between users, sensors and the environment in a semantically rich way that is query-able through standard methods such as SPARQL or graph database queries

Most of these platforms are written in Java. What makes Java a good choice for this kind of task is the large variety of interpreters (JVMs), available on different hardware platforms and its proven stability.

OSAmI Commons is an open source, universal networking platform enabling vertical application domains. It is built around the concept of SOA (Service Oriented Architecture), with a focus on interoperability and open-source[4]. OSGi[1] is one of its core technologies. Its philosophy is that "one of the ways to reduce the development cost is to reuse existing software as much as possible" [11]. The physical architecture of an OSAmI system is structured into four types of entities: nodes, systems, support infrastructure elements and background systems (OSGi wrappers for in house legacy systems). A system is modelled like a dynamically changing set of nodes, who self-organize into groups acting together as a single service provider.

One important aspect of OSAmI Commons is its toolchain for assisting developers in writing software on top of it: tools for new service creation, editing code in an OSAmI specific environment (a plugin for Eclipse) and for packaging applications in the deployment stage.

The main player in the Ambient Assisted Living (AAL) software framework market is the **OpenAAL framework**[20], the byproduct of a series of European projects such as SOPRANO-IP[2], universAAL[3], and PERSONA AAL [10]. The framework has a holistic approach, offering software components for all stages in the lifecycle of an AAL project, from developing the software in an integrated editor (it has an Eclipse plugin as well), to a marketplace to connect possible customers with OSGi services to be deployed by professionals in their homes. It provides a smart distributed services layer on top of the following concepts:

[2] `http://www.soprano-ip.org`

[3] `http://www.universaal.org/`

- AAL space - the environment in which users interact with the embedded computer artifacts, and are supported by their reasoning and context awareness
- channels - abstract communication pipes between system components that are used to deliver semantically annotated messages between them
- actuator - a device that can provoke changes in the physical world, similar to the artifact concept from the agents and artifacts model[15]
- sensor
- context - data that can be shared between components in order to increase their situational awareness

The communication between the components is done by using ontologies specially developed for the purpose of this project, that provide an uniform semantic annotation, allowing reasoning components to provide uniform conclusions regardless of the source and destination of the message.

Just as the OSAmI Commons project, the code is open source and publicly available for reuse, however, it is in its early beta phases.

2.1. **AmICiTy**

AmICiTy[13] is an interesting hybrid system between an ambiental intelligence solution and a multi-agent system. Even though compared to the previous two frameworks it was only deployed in academic settings, it is a very interesting model of an ambiental intelligence interaction. Environmental resources (artifacts) are translated into local contextual graphs managed by resource-manager agents, who use an information dissemination middleware in order to share information with each other; moreover, certain agents have preferences for certain types of facts (represented as subgraphs). Goals of a person interacting with this kind of environment are realized by having the person's associated personal agent constantly interact with the environmental agent and other personal agents.

3. **The UPB AmI Laboratory**

We will continue by describing the functionality of the experiment running in the UPB AmI Laboratory, powered by the platform proposed here. In this laboratory, there are more than 50 audio, video, proximity, humidity and temperature sensors, laid out in 9 keypoints (figure 1, page 21). The keypoints are spread out throughout the room at the same height in order to optimally cover its viewing angles (figure 2, page 21).

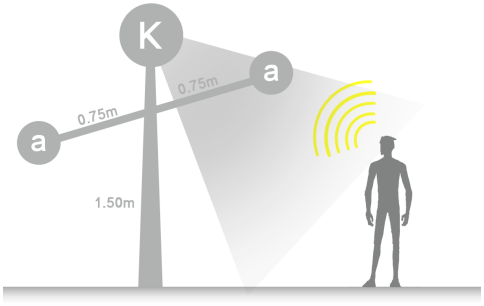

FIGURE 1. Keypoint - 1 Kinect and 2 Arduino boards with sensors

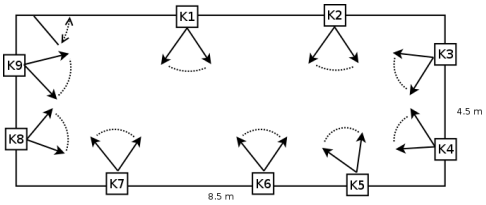

FIGURE 2. Layout of the keypoints in the laboratory

On top of this array of sensors, formed mainly from Microsoft Kinect[16] depth cameras and Arduino[14] boards, the AmI-Platform software framework[4] is running. The currently deployed experiment performs real-time person tracking with a precision of 10 cm and aggregates multiple measurements of different types for the same person in real time. Developing this kind of experiment required us to try multiple versions of the algorithms on the same dataset, both from the laboratory and from the cloud, and has led us to solving the experiment reproducibility problem in AmI.

4. Why Work on Real Data Instead of Simulations While Developing Algorithms?

Why cannot a researcher work on simulations and needs to access real sensory data in order to create new algorithms, architectures and concepts? While building our system, we have encountered twice the situation where we developed an algorithm on synthetic data, and it needed major modifications in order to run on real-world data:

- when we were implementing AndroAR [5], the system for augmenting the camera image on the smartphone screen with information about the real-world buildings recognized using GPS location and visual features of the artifacts. In this case, standard feature-matching algorithms from computer vision literature did not perform well on buildings that were quite different, but did so on official benchmarks with predictable lighting and viewing angles. Even when matching the same building against a slightly changed image of itself, windows from the top floor were mistaken for windows on the bottom floor. We employed geometric reasoning in

[4] https://github.com/ami-lab/AmI-Platform

order to ensure that groups features matched belonged to the same region of the objects.

- when implementing AndroRemote[5], we tested the object detection service on the COIL-100 database [12]. Even though when we initially developed the algorithms, we devised complex scenarios on top of this dataset, they could not discern between two very different real-world objects: a computer monitor, and an air conditioning system. The elaborate training scheme for the algorithms turned out to work on the COIL dataset because the amount of data we had used for training was unrealistic: the dataset had pictures from tens of angles, with normalized lighting and image size. We realised that this is an unreasonable requirement with respect to the deployment of an AmI system, just like the authors warn in [18].

The conclusion is simple: the researcher needs to work on real data as soon as possible, because more problems than initially planned will arise anyway. In order to be able to do that, a researcher needs a simple way to record and consume such recorded data.

5. A Record and Replay Component

In AmI-Platform[6], the acquisition and processing of sensor data in real-time is done using a loosely coupled infrastructure based on the concept of PDU (Processing Data Unit). The PDU is an independent entity which processes messages delivered to it by an omni-present queue system. A computation is represented by a directed graph of PDUs running independently, sending messages to one another. One example of such a graph can be found in figure 3 (page 23). This concept is called in the literature *stream processing*[19].

In order to allow the system to record sensor data and replay it at a later time, the whole architecture was split into two conceptual halves: the data acquisition and the data processing (figure 4, page 24). According to this scheme, all data enters the system through a single entry point (the *measurements* queue) and can be therefore recorded by a separate system before entering it. We decided to record the sensor data in files, with one measurement per line, in JSON format. The full format of these measurements is described in[7]. Measurements are recorded according to a set of *active* experiments, which specify a set of filtering criteria (such as sensor type or measurement type).

[5]Implementation by Mircea Trăichioiu, a BSc. student, as part of his AI-MAS (http://aimas.cs.pub.ro) internship in the summer of 2011

[6]`https://github.com/ami-lab/AmI-Platform`

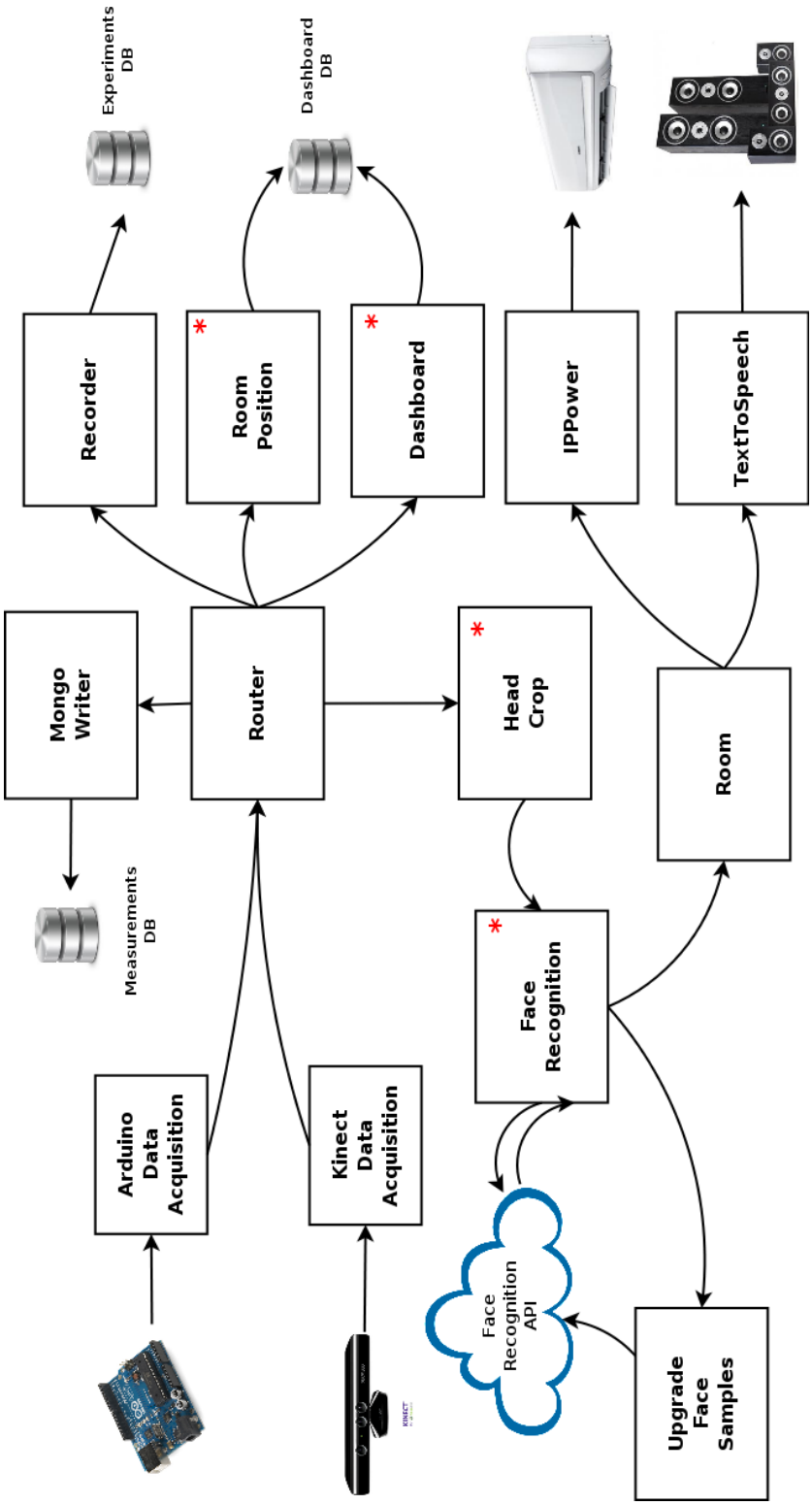

FIGURE 3. Graph of PDUs used for the deployed UPB AmI Lab experiments

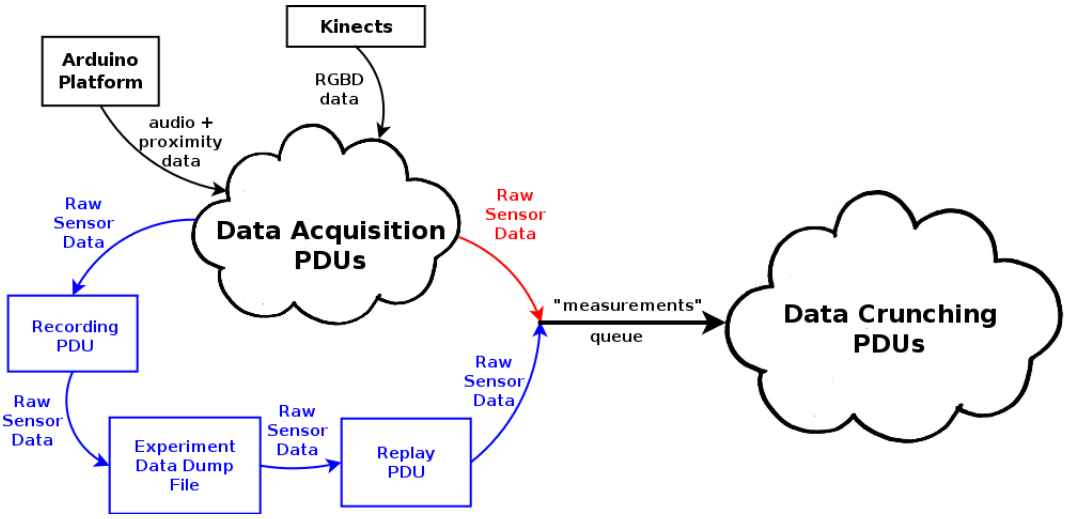

FIGURE 4. Record and Playback System

In order to replay the recorded data, the data acquisition circuit (represented in red in figure 4 on page 24) must be deactivated first. The replay system (represented in blue in the same figure) will afterwards feed input data to the graph of PDUs by using a playback algorithm and an input file with recorded data. The main challenge is to feed the measurements at the exact same rate as they were recorded in the first place. Two types of problems can appear for the replay circuitry:

- it cannot process the data file quickly enough in order to feed measurements at the requested rate. There are 2 solutions to this: scale the replay system by allocating more resources to it (impractical in most cases, since the replay system will usually run on the researcher's working machine), or skip some measurements until the replay system has caught up. Depending on the algorithm to be tested, the researcher should be able to choose whether real-time or data completeness is more important.
- it can feed data to the data crunching system processes data too quickly. This means that the sensors generated the data values at a slower rate than the replay system can feed them. Therefore, it has to slow down in order to adapt to the sensor data generation rate. An artificial delay will be introduced, equivalent to the time left until the next measurement should arrive. One interesting aspect related to waiting for this time is that system calls for waiting for a certain amount of time (called *sleep* in POSIX[9] operating systems) guarantee that the current process will sleep for *at least* the requested amount of time, not exactly it. The cause

for this is the nedeterminism found in process schedulers of the operating systems. While in general it should not be a problem, it might become one for sensor recordings where measurements are very close to one another.

When replaying the measurements, we cannot expect the processing system to work in relative time (it is possible, but very hard to achieve). This means that the timestamps of the measurements must be modified to the moment when they enter the system (as opposed to the moment when they were initially recorded). While data fed to the system will be purposely inaccurate, the relative time differences between measurements fed into it will remain almost the same.

6. **Using a Cloud Infrastructure to Replay an Experiment**

Due to the resource-intensive nature of our system (and probably of other AmI systems as well, in a similar setting), it is not reasonable to assume that the whole system can be ran on a researcher's computer. Moreover, both [6] and [8] have noticed the exagerated assumptions the creators of AmI systems make about people trying to use them in their own experiments - no good guidelines on how to install and configure the systems. If they have to install our own infrastructure and the numerous components needed to make it work, it might be a very difficult task. Therefore, we have opted for an approach that is very easy to use for another researcher but costs them money: renting servers automatically from a cloud provider and provisioning the cluster of servers to set-up everything needed for experimenting with our own platform and sensor recordings.

We have chosen Amazon's EC2 (Elastic Compute Cloud) as our provider for virtual servers. It "is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers"[7]. It offers a wide range of machine types that can be rented for an hourly cost. These differ in storage capabilities, CPU cores and RAM memory. In Table 1, on page 26 we give a few examples of machine types, their capabilities and hourly costs as of September 2013.

We have created an automated system for renting these machines, installing the software needed on them and playing back an experiment (figure 5, page 28). The number of machines, types of modules and their role in the system is specified in an *experiment profile* (one such example profile can be found in listing 1, page 27). The explanation for the fields present in the experiment profile:

[7] http://aws.amazon.com/ec2/

Machine Type	Hourly Cost	Compute Units	Memory
t1.micro	$0.02	2 (only for short bursts)	0.60 GB
m1.small	$0.06	1 (1 core × 1 unit)	1.70 GB
m1.medium	$0.12	2 (1 core × 2 units)	3.50 GB
c1.medium	$0.14	5 (2 cores × 2.5 units)	1.70 GB
m1.large	$0.24	4 (2 cores × 2 units)	7.50 GB
c1.xlarge	$0.58	20 (8 cores × 2.5 units)	7.00 GB

TABLE 1. Amazon EC2 machine types

- *Name* is the name of the machine, and it is used to uniquely identify it in the group of started machines. This name will be displayed in the web interface next to the machine capabilities in the researcher's Amazon account.
- *machine_type* refers to the type of machine, several valid values are provided in table 1
- *ami_id* refers to the Amazon Machine Image from the marketplace for such images: `https://aws.amazon.com/marketplace`. It is a file containing the disk image of a virtual machine with all the necessary software installed for the purpose it declares. We are currently using a stock Ubuntu 12.04 image, and plan to create our own AMI images for Ambiental Intelligence Applications.
- *security_group* is the group of machines to which this new machine belongs to, from a security point of view. Amazon Web Services (AWS) offers an interface for managing such machine groups together, by specifying firewall rules that apply to all of them as a whole. For example, the default security group that comes with an EC2 account will give access only to port 22 (SSH) for the opened machines.
- *manifest* refers to the Puppet[17] manifest to be applied to that machine, a file containing descriptions of resources such as files, packages and services written in a domain-specific language. Such files are executed by an interpreter which is functional in nature (so one describes the desired state of the machine, rather than the steps necessary to reach that state) in order to provision the machine correctly.
- *type* refers to the type of machine. In our Ambient Intelligence platform, valid types are: *crunch* (generic data processing node), *mongo* (a MongoDB[3] storage node for measurements), *redis* (a Redis cache node for sharing global state between PDUs) or *kestrel* (a queue server node).
- *modules* refers to the data crunching modules that should be ran on that node. Notice, how, in our example, the queue system node does not run

any module, even though it might be beneficial to do so (being close to the queue system would yield a faster interaction with it).

```
[
    {
          "Name": "crunch_01",
          "meta": {
                 "machine_type": "c1.xlarge",
                 "ami_id": "ami-d0f89fb9",
                 "security_group": "sg-82df60e9",
                 "keypair": "ami-keypair",
          },
          "manifest": "crunch_01.pp",
          "type": "crunch",
          "modules": "ami-router,ami-room-position"
         },

         {
          "Name": "queues",
          "meta": {
                 "machine_type": "m1.large",
                 "ami_id": "ami-d0f89fb9",
                 "security_group": "sg-82df60e9",
                 "keypair": "ami-keypair",
          },
          "manifest": "kestrel.pp",
          "type": "kestrel",
          "modules": ""
         }
]
```

Listing 1: Experiment profile with 2 machines

Once the machines are rented from the cloud provider, the workflow from figure 5 (page 28) will be executed by our provisioning system in order to ensure that the servers are ready for running an experiment. At the end of the workflow, the servers will have the correct software installed and will know the addresses of other servers they need, and will be ready for executing an experiment.

The first experiment run takes 15 minutes (time which includes starting up the machines and provisioning them correctly), while the subsequent ones take only 1-2 minutes (the time necessary to download the experiment file and load it into memory).

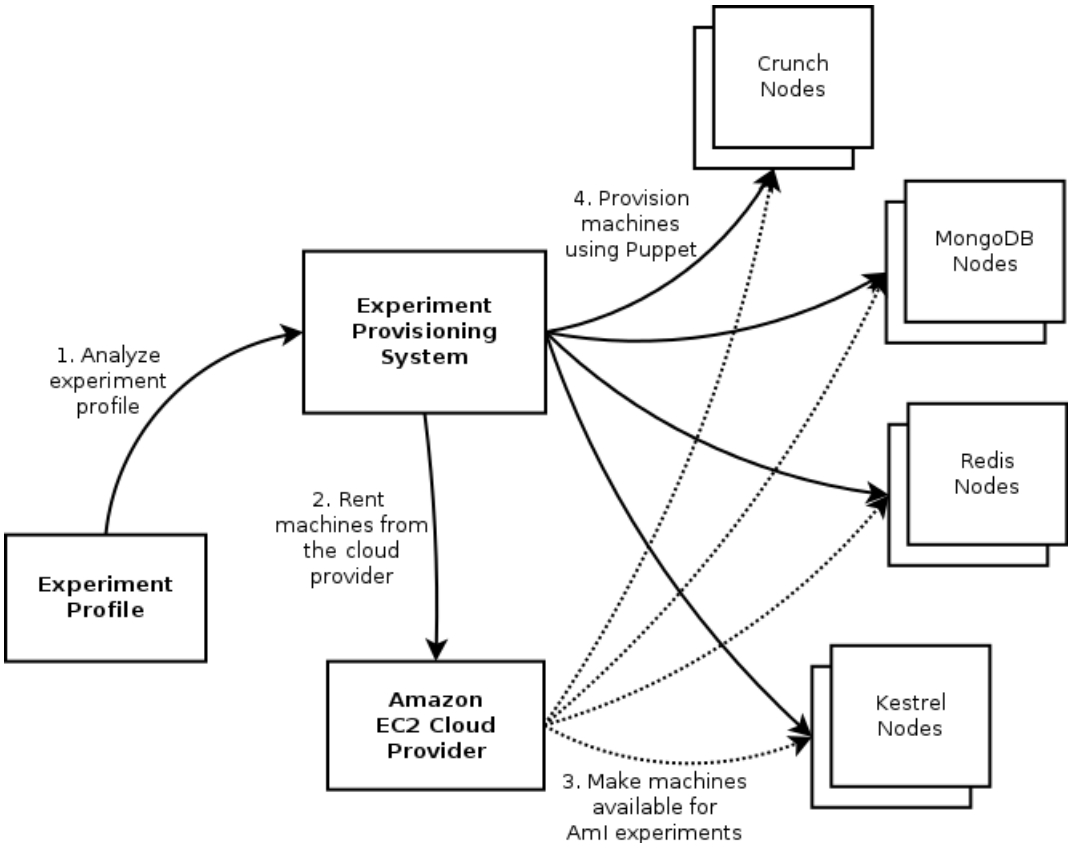

FIGURE 5. Experiment Provisioning System flow from renting servers to experimenting

7. Conclusions and Future Work

We have identified a major problem preventing researchers from across the world to develop robust Ambient Intelligence systems: experiment reproducibility. This manifests itself in many ways, ranging from slow and costly development to lack of reusable components for building such applications and to publications with experiments that cannot be verified (in contrast to other areas of research in computer science).

In order to solve this problem, we created a cloud infrastructure able to replay experiments previously recorded with another tool for recording. We have presented architectural diagrams and details about how we implemented all these functionalities. The end result is an open-source platform[8] which

[8] `https://github.com/ami-lab/AmI-Platform`

manages to open a set of 8 servers and install everything correctly on them in less than 15 minutes, at the cost of 0.50$ per hour.

We argue that this system in combination with a platform for efficiently disseminating sensor recordings from laboratories using it around the world is a complete solution to the problem of experiment reproducibility in Ambient Intelligence.

8. **Acknowledgements**

This paper has been funded by:

- Sectorial Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU 76813.
- ERRIC FP7 Project, number 264207, FP7-REGPOT-2010-1, led by the University Politehnica of Bucharest

REFERENCES

[1] OSGi Alliance. Osgi service platform, core specification, release 4, version 4.1. *OSGi Specification*, 2007.

[2] EC Amazon. Amazon elastic compute cloud (amazon ec2). *Amazon Elastic Compute Cloud (Amazon EC2)*, 2010.

[3] Kyle Banker. *MongoDB in action*. Manning Publications Co., 2011.

[4] Naci Dai, Wolfgang Thronicke, Alejandra Ruiz López, Félix Cuadrado Latasa, Elmar Zeeb, Christoph Fiehe, Anna Litvina, Jan Krueger, Oliver Dohndorf, Isaac Agudo, et al. Osami commonsan open dynamic services platform for ambient intelligence. In *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1–10. IEEE, 2011.

[5] Alexandru Damian. Android-based crowd sourcing of landmark metadata, 2012.

[6] Adrian Friday, Manuel Roman, Christian Becker, and Jalal Al-Muhtadi. Guidelines and open issues in systems support for ubicomp: reflections on ubisys 2003 and 2004. *Personal and Ubiquitous Computing*, 10(1):1–3, 2006.

[7] Andrei-Adnan Ismail, Cosmin Marian, and Adina-Magda Florea. A framework for consistent experimentation in ami. In *The International Workshop on Agent Technology for Ambient Intelligence*, 2013.

[8] Rui José, Helena Rodrigues, and Nuno Otero. Ambient intelligence: Beyond the inspiring vision. *J. UCS*, 16(12):1480–1499, 2010.

[9] Donald Lewine. *POSIX programmer's guide: writing portable UNIX programs with the POSIX. 1 standard*. O'Reilly, 1991.

[10] Guido Matrella Ferdinando Grossi Stefania Baratta Michele Amoretti, Giorgia Copelli. The persona aal platform: Deployment in the italian pilot site of bard. In *AALIANCE conference*, Malaga, Spain, 2010.

[11] Tatsuo Nakajima. Case study of middleware infrastructure for ambient intelligence environments. In *Handbook of Ambient Intelligence and Smart Environments*, pages 229–256. Springer, 2010.

[12] SK Nayar, SA Nene, and H. Murase. Columbia object image library (coil 100). Technical report, Tech. Report No. CUCS-006-96. Department of Comp. Science, Columbia University, 1996.

[13] Andrei Olaru. A context-aware multi-agent system for ami environments. Technical report, Technical report, University Politehnica of Bucharest, University Pierre et Marie Curie Paris, 2010.

[14] E. Ramos. Arduino basics. *Arduino and Kinect Projects*, pages 1–22, 2012.

[15] A. Ricci, M. Viroli, and A. Omicini. Give agents their artifacts: the a&a approach for engineering working environments in mas. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 150. ACM, 2007.

[16] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1297–1304. IEEE, 2011.

[17] Jason A Smith, John S De Stefano Jr, John Fetzko, Christopher Hollowell, Hironori Ito, Mizuki Karasawa, James Pryor, Tejas Rao, and William Strecker-Kellogg. Centralized fabric management using puppet, git, and glpi. In *Journal of Physics: Conference Series*, volume 396, page 042056. IOP Publishing, 2012.

[18] Pallavi Kaushik Stephen S. Intille and Randy Rockinson. Deploying context-aware health technology at home: Human-centric challenges. In *Human-Centric Interfaces for Ambient Intelligence*, pages 479–503. Elsevier, 2010.

[19] Michael Stonebraker, Uur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. *ACM SIGMOD Record*, 34(4):42–47, 2005.

[20] P. Wolf, A. Schmidt, J.P. Otte, M. Klein, S. Rollwage, B. König-Ries, T. Dettborn, and A. Gabdulkhakova. openaal-the open source middleware for ambient-assisted living (aal). In *AALIANCE conference, Malaga, Spain*, 2010.