This avoids calling an object's constructor and permits easy instantiation of classes that do not have a no-argument constructor. As the complete state of the object is restored from serialized data, processing of the constructor is not necessary.

Then, each field is restored with its primitive value or with a reference to an object:

```
for (Entry<String, ObjectState> entry
             : fields.entrySet()) {
  String fieldName[] =
         entry.getKey().split("#", 2);
  ObjectState fieldState = entry.getValue();

  Class clazz = Class.forName(fieldName[0]);
  Field field =
         clazz.getDeclaredField(fieldName[1]);

  field.setAccessible(true);
  fieldState.restoreInstanceToField(field,
                                    instance);
}
```

As a result, our implementation fulfills all five qualities that we expected from our solution.

However, our solution is still not ideal. Currently, it is only possible to save and restore application state as a whole. Moreover, serialization and deserialization using an XML file on the file system is relatively slow. This may result in a performance issue when emulating the Java Card transaction mechanism: Every time a transaction is started the whole application state needs to be captured. Similarly, rollback of a transaction requires implantation of the whole application state.

7. CONCLUSION

In this paper, we showed that there are significant differences between the Java Card virtual machine and other VMs. These differences cause problems with scenarios where Java Card applications are emulated on top of non-Java Card VMs. However, we found that it is possible to overcome the problems caused by different virtual machine lifecycles by adding a persistence framework to the Java and Dalvik virtual machines. Our solution can extract the state of Java Card applications and store it to persistent memory. Later, this persisted state can be re-implanted into the application (recreating all objects, references and primitive values). Nevertheless, our solution is not ideal. Currently, our solution is only capable of extracting and re-implanting application state as a whole, does not perform any caching and is not capable of storing or reverting only modified fields. This, however, has a big impact on performance when it comes to modeling Java Card's transaction mechanism: Whenever a transaction is aborted, the state of the whole application has to be reverted. Therefore, future research should focus on optimization for this scenario.

We implemented and tested our concept using a simple class hierarchy and object network to model our application state. In the future, we intend to integrate our implementation together with jCardSim in our emulator scenarios.

8. ACKNOWLEDGMENTS

This work is part of the project "High Speed RFID" within the EU program "Regionale Wettbewerbsfähigkeit OÖ 2007–2013 (Regio 13)" funded by the European regional development fund (ERDF) and the Province of Upper Austria (Land Oberösterreich).

Moreover, this work has been carried out in cooperation with "u'smile", the Josef Ressel Center for User-Friendly Secure Mobile Environments, funded by the Christian Doppler Gesellschaft, A1 Telekom Austria AG, Drei-Banken-EDV GmbH, LG Nexera Business Solutions AG, and NXP Semiconductors Austria GmbH.

9. REFERENCES

[1] D. Barry and T. Stanienda. Solving the Java Object Storage Problem. *Computer*, 31(11):33–40, Nov. 1998.

[2] R. G. G. Cattell and D. K. Barry, editors. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann Publishers, 1997.

[3] A. Rashid and R. Chitchyan. Persistance as an aspect. In *Proceedings of the 2nd International Conference on Aspect-oriented Software Development (AOSD)*, pages 120–129. Boston, MA, USA, 2003.

[4] M. Roland. Software Card Emulation in NFC-enabled Mobile Phones: Great Advantage or Security Nightmare? In *4th International Workshop on Security and Privacy in Spontaneous Interaction and Mobile Phone Use*. Newcastle, UK, June 2012.

[5] M. Roland. Debugging and Rapid Prototyping of NFC Secure Element Applications. In *Mobile Computing, Applications, and Services*, LNICST. Springer, Paris, France, Nov. 2013.

[6] SIMalliance. *Open Mobile API specification*, June 2012.

[7] Sun Microsystems, Inc. *Java Card Platform: Runtime Environment Specification, Version 2.2.2*, Mar. 2006.

[8] Sun Microsystems, Inc. *Java Card Platform: Virtual Machine Specification, Version 2.2.2*, Mar. 2006.

[9] G. Watson. *ORMLite Package, Version 4.45*, Mar. 2013.

[10] D. Yeager. Added NFC Reader support for two new tag types: ISO PCD type A and ISO PCD type B. Patches to the CyanogenMod aftermarket-firmware for Android devices, Jan. 2012.

DOI 10.1145/2536853.2536870