Open Platforms: New Challenges for Software Engineering

Emilie Balland, Charles Consel

Open Platforms: New Challenges for Software Engineering

Emilie Balland Charles Consel

INRIA Bordeaux Sud-Ouest / University of Bordeaux, France
emilie.balland@inria.fr/charles.consel@inria.fr

Abstract

Recently, platforms running third-party applications have become very popular, in particular due to the explosion of the smartphone market. These *open platforms* propose a rich stream of applications, targeting general-purpose as well as customized user needs. Developing applications for such target platforms introduces new requirements and raises new challenges.

This paper identifies key requirements for designing open platforms. After an overview of the existing approaches, we identify the challenges for the next generation of open platforms.

Keywords Open platforms, Networked Entities, Programming Frameworks, Middleware

1. Introduction

Open platforms can be characterized by three major features: (1) they offer *public programming interfaces* that third-party developers use to program new applications; (2) they provide *shared resources* to the applications that can either be hardware (camera, GPS. . .), software (email client, web browser. . .), or data (user profile, address book. . .); and, (3) they offer a *runtime environment* for applications hosted by a given device (*e.g.,* a smartphone) where the resources are local or networked.

In addition to these three characteristics, open platforms also provide *upload/download capabilities*, allowing applications to be disseminated: an application developer can upload his/her applications to make them visible and easily loadable by users.

From these characteristics, we identify three stakeholders: (1) the open platform provider, (2) the developer of third-party applications and (3) the user of the platform. The stakeholders have different expectations relative to the platform. Typically, the platform user is interested in controlling what resources are used by an application, while the application developer is concerned with the popularity of the platform and its ease of development, considering the low return on investment of application development, if any.

1.1 New Application Domains

Clearly, the concept of open platforms is not new. For example, operating systems kernels have been presented as open platforms where applications and resources can be viewed as system modules and hardware devices, respectively (*e.g.,* [6, 11]). However, system modules interact with the platform at a much lower level, preventing the use of software engineering approaches.

Over the last years, open platforms have been deployed in application domains where the end-user directly operates the platform (by loading new applications for example) and the resources are networked (*e.g.,*access to web resources). Let us examine prominent examples of such platforms.

Smartphones. Apple has developed an open platform for the iPhone. This open platform is based on a specific SDK [10], including public programming interfaces to the iPhone resources. The success of this platform is mainly due to the App Store that offers a tightly-controlled upload/download environment targeting a range of devices (*i.e.,* iPhone, iPad and iTouch). The Android SDK [22] developed by Google follows a similar approach. The main difference is that applications do not need to be validated by the provider to be accessible to users. Android has also proposed App Inventor [1], inspired by Scratch [21]; this environment allows non-programmers to develop their own Android applications.

Social networks. Websites such as Facebook or MySpace allow developers to create new applications for social networking. These applications can access the resources of the platform through specific APIs [2, 13]. Then, the users can deploy any of these applications in their own profiles and share them with their friends. When installing a new application, websites such as Facebook queries the user whether to give the application access to his/her private resources. Yet, as of now, the process is rather coarse-grained, raising concerns about privacy.

Video game consoles. The latest consoles have been designed as open platforms. They offer plenty of services such as game stores (*e.g.,* XBox live [5]) and toolkits for developing new games (*e.g.,* XNA Game Studio Express [7]). In fact, an emerging trend is games dedicated to the development of small games. For example, the Nintendo DS game named "WarioWare: Do it Yourself" [4] allows people to develop tiny games using a dedicated graphical language and to share them via a common platform.

Plug-ins. A plug-in is a set of software components that adds specific capabilities to a software system. Software systems with plug-in capabilities can be considered as open platforms. The most successful domain for software systems with plug-ins is certainly web browsers. Most web browsers such as Firefox offer a plug-in system through public APIs and a public download/upload web platform.

In these new application domains, the user customizes the platform by installing applications of interest in conformance with his/her resource usage policies. Another novelty of these application domains is that non-programmers are increasingly involved in application development, making open platforms more user-centric.

1.2 Requirements

Let us now identify and examine the key requirements for designing open platforms.

Safety. Most of the platform providers want third-party applications to be analyzable with sufficient accuracy to avoid malicious actions or to certify the conformance with their expected behaviors. For example, phone applications should not send SMS or access to paying services unless it is part of their nominal behaviour.

Security. Accesses to resources have to be controlled, to ensure the privacy of the platform users. To achieve this goal, the available resources have to be clearly identified and the user has to know the resources used by each application, and more precisely, how these resources are used. For example, the user is interested in the confinement of sensitive data.

Resource extensibility. To adapt open platforms to unanticipated users' demands, it is critical for the platform providers to allow resource extensibility. This requires support for the declaration and the discovery of new resources.

Programmability. The public programming interfaces should facilitate the development of new applications, even allowing non-programmers to develop their own applications. If the programming interfaces are high level enough, programming boils down to gluing operations.

Low-cost. In these new application domains, the majority of the applications are distributed at a small scale. Consequently, the open platform must enable application development to be low-cost.

Obviously, these requirements are not specific to open platforms but the openness makes them even more critical for the stakeholders. Section 2 makes an inventory of the existing development approaches and show their limits with regard to these requirements. Then, Section 3 examines the main challenges for the next generation of open platforms.

2. Overview of Existing Approaches

An open platform is characterized by three main elements: the programming interfaces, the shared resources and the runtime environment. For each of these elements, there exist different design strategies partially addressing the requirements discussed earlier.

2.1 APIs and Middlewares

Middlewares such as J2EE [20] or CORBA [14] have been used to support and simplify the development of complex distributed applications. They offer APIs to develop software components and a runtime environment to easily connect them over a distributed system.

Programming interfaces. Middlewares attempt to cover as much of their target application domain as possible in a single library. This strategy often leads to large APIs, providing little guidance to the application developer [15] and requiring a steep learning curve. This situation makes it difficult to achieve programmability and low-cost development.

Resource descriptions. Most middlewares include an Interface Description Language (IDL). IDLs are used to describe software components, offering a bridge between heterogeneous systems. For example, WSDL [9] is an IDL dedicated to Web Services and AIDL [22] is an IDL dedicated to the Android platform. IDLs facilitate the analysis of resource usage but are not accurate enough to lift the need for dynamic checks.

Runtime environment. Middlewares abstract over variations in hardware, operating systems or distribution systems technologies. Even though, as the components of an application might come from heterogeneous middlewares, the multiplication of distribution models still lead to interoperability issues; this situation is referred to as *the middleware paradox* [18].

2.2 Programming Frameworks

A programming framework is a software layer that organizes and supports applications. A framework may include APIs, a scripting language, or any supporting mechanism that helps to develop and glue together the different components of a software project. In comparison with libraries, the main characteristic of a programming framework is that it defines patterns of control flow. For example, the iPhone's SDK [10] allows the application developer to program code snippets that are then called by the runtime core of the framework. This approach is known as Inversion of Control [12] or Hollywood Principle [23] ("Don't call us, we'll call you").

Programming interfaces. The control-flow patterns underlying a framework facilitate the programming by guiding developers. To make open platforms accessible even to non-programmers, development frameworks have sometimes been built on top of conventional programming frameworks. For example, App Inventor [1] allows non-programmers to develop applications. They can visually design the way the application looks and specify the application's behavior using a visual-block programming language. Generally, the APIs offered by programming frameworks are higher level than the ones provided by middlewares, improving development support. The inversion of control facilitates code analysis by limiting the locations where the application logic is plugged in, and restricting the control flow.

Resource descriptions. In most programming frameworks, resource usage are solely characterized by API invocations [2, 10, 13]. Such an approach has a number of limitations. For example, it does not address dynamic resources as in pervasive computing environments. An exception is the Android platform that offers a dedicated IDL [22], allowing dynamic binding of resources.

Runtime environment. Programming frameworks impose a control-flow pattern on applications. As a consequence, the runtime environment can ensure strong security properties. For example, the multitasking in the iPhone runtime environment is moderate (*e.g.,* due to memory constraints, the runtime environment purges applications that have not been resumed recently) and each iPhone application is launched in a sandbox, limiting access to files, network resources or hardware subsystems [10].

3. Towards More Declarative and Customizable Open Platforms

The last generation of open platforms has shown that well-designed programming frameworks are an attractive development approach. They offer a better programming and analysis support than generic middlewares. However, they limit the kind of applications one can develop and the resources that can be accessed. The challenges for the next generation of open platforms is to lift these limitations.

3.1 Resource Declaration

In application domains such as pervasive computing, where resources can be dynamically discovered, languages dedicated to resource description are already used. For example, in service-oriented architectures such as the OSGi framework [19], both applications and resources are viewed as services accessible across a distributed computing environment. Networking protocols such as UpNP [17] provide native support for the discovery of new entities. When connected to a network, entities automatically broadcast their network address and supplied types of services. Another example of resource-specification languages in the web domain is

the W3C Resource Description Framework [3] (RDF) that allows to specify web data models. In pervasive computing, the domain-specific language Diaspec [8] offers a taxonomy layer dedicated to describing classes of entities that are relevant to the target application area. From this specification, the compiler generates customized programming support dedicated to the required resources of a class of applications. This declarative approach allows not only dynamic resource discovery but also accurate resource access analysis.

The next generation of open platforms need to offer such declarations for resources: they play a key role for dynamic resource extensibility. Furthermore, these declarations facilitate both safety and security analysis by providing information about the resource accesses.

3.2 Customized Programming Frameworks

The main drawback of the existing approaches is their genericity that results in large APIs. Additionally, generic approaches are limited in development and analysis support. To overcome this limitation without loosing extensibility, the future open platforms should offer programming frameworks that are automatically customized with respect to a given class of applications and required resources. These customization parameters would be expressed declaratively with domain-specific concepts. For example, the domain-specific language Diaspec [8] allows the specification of Sense/Compute/Control (SCC) applications based on a dedicated architectural pattern. From this specification, the compiler generates customized programming support dedicated to the application. Such approaches could be used for describing the resources of an open platform and for guiding the development of third-party applications.

To offer such customized programming frameworks, the platform providers have to characterize a range of applications and associated resources. Then, generated programming frameworks will guide application development via architectural patterns dedicated to the supplied declaration, improving programmability and reducing the development costs.

3.3 Static Verification of Third-Party Applications

In middlewares, most of the security verification is done at the level of the runtime environment by checking dynamically whether an access to a resource respects a given security policy. For example, Java midlets can be launched in a defensive manner, forcing the user to validate each resource access. To avoid this interaction with the users, platform providers can design security policies specific to applications. However, the main drawback of such dynamic verifications is the performance cost.

In programming frameworks such as that of Facebook, users can choose which part of their personal resources are shared. When installing a new application, they are informed whenever a resource is accessed by this application. Such security analysis can be carried out because of the restrictive API. However, existing programming frameworks are still too generic to allow fully static verification of the applications. For example, Facebook users are not only interested in the list of the resources required by an application but also how the application utilizes these resources. For example, it would be interesting to check data confinement: is the information sent to external systems or only used for statical purposes?

Using customized programming frameworks could allow a better verification support by restricting the control and data flows for a given application and thus enable the decidability of a larger set of properties. Some of these properties depend on the application (*e.g.,* behavioral invariants of the application). The application developer could declare the expected behaviors of his/her application by defining such properties in a manifest file associated to the application. Then, this behavioral specification would be readable by the platform user and the open platform could statically check whether the code is conform to these specifications due to the code restriction. This approach is illustrated by DiaSpec where architectural descriptions are translated into Promela specifcations, allowing the verification of LTL invariants [16]. Such invariants can be used by the application developer as behavioral specifications.

By making the developer declare more information about the application behavior, a better safety and security support can be offered. This extra task does not penalize the application developer as the generated programming framework also reduces his/her development costs.

4. Conclusion

We have identified the main requirements in the design of open platforms and shown how the current approaches and technologies fail to fulfill them. We have introduced the challenges open platforms raise to reconcile security and extensibility. Finally, we have sketched directions to address key requirements of open platforms.

References

[1] App inventor. URL `http://appinventor.googlelabs.com`.

[2] Myspace platform. URL `http://developerwiki.myspace.com`.

[3] Rdf specification. URL `http://www.w3.org/RDF`.

[4] WarioWare: Do it Yourself. URL `http://www.wariowarediy.com`.

[5] Xbox-live website. URL `http://www.xbox.com/live`.

[6] B. N. Bershad, S. Savage, E. G. Sirer, M. Fiuczynski, D. Becker, S. Eggers, and C. Chambers. Extensibility, safety and performance in the spin operating system. pages 267–284, 1995.

[7] C. Carter. *Microsoft XNA game studio 3.0 unleashed*. Sams, 2009.

[8] D. Cassou, B. Bertran, N. Loriant, and C. Consel. A generative programming approach to developing pervasive computing systems. In *GPCE'09: Proceedings of the 8th International Conference on Generative Programming and Component Engineering*, pages 137–146, Denver, CO, USA, 2009. ACM.

[9] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web service definition language (wsdl). Technical report, March 2001. URL `http://www.w3.org/TR/wsdl`.

[10] J. L. Dave Mark. *Beginning iPhone Development: Exploring the iPhone SDK*. Apress, 2009.

[11] D. R. Engler, M. F. Kaashoek, and J. O'Toole. Exokernel: an operating system architecture for application-level resource management. In *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 251–266, New York, NY, USA, 1995. ACM. ISBN 0-89791-715-4.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Abstraction and Reuse of Object-Oriented Design*. Addison-Wesley, Boston, MA, January 1995. ISBN 0201633612.

[13] J. Goldman. *Facebook Cookbook: Building Applications to Grow Your Facebook Empire*. O'Reilly Media, Inc., 2008. ISBN 059651817X, 9780596518172.

[14] O. M. Group. Corba component model 4.0 specification. Specification Version 4.0, Object Management Group, April 2006. URL `http://www.omg.org/docs/formal/06-04-01.pdf`.

[15] M. Henning. The Rise and Fall of CORBA. *ACM Queue*, 4(5), 2006.

[16] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[17] M. Jeronimo and J. Weast. *UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play*. Intel Press, 2003. ISBN 0971786119.

[18] F. Kordon and L. Pautet. Toward nex-generation middleware? *IEEE Distributed Systems Online*, 6(3):2, 2005. ISSN 1541-4922.

[19] OSGi. OSGi alliance. URL `http://www.osgi.org`.

[20] P. J. Perrone and K. Chaganti. *J2EE: Developer's Handbook*. Pearson Education, 2003. ISBN 0672323486.

[21] M. Resnick, J. Maloney, A. Monroy-Hernndez, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: Programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[22] R. Rogers, J. Lombardo, Z. Mednieks, and B. Meike. *Android Application Development: Programming with the Google SDK*. O'Reilly, Beijing, 2009. ISBN 978-0-596-52147-9.

[23] R. E. Sweet. The Mesa programming environment. In *Proceedings of the ACM SIGPLAN 85 symposium on Language issues in programming environments*, pages 216–229, New York, NY, USA, 1985. ACM. ISBN 0-89791-165-2. doi: http://doi.acm.org/10.1145/800225.806843.