Compliance Checking in the Open Payments Ecosystem*

Shaun Azzopardi[1], Christian Colombo[1], Gordon J Pace[1], and Brian Vella[2]

[1] University of Malta
[2] Ixaris Ltd.

Abstract. Given the strict legal frameworks which regulate the movements and management of funds, building financial applications typically proves to be prohibitively expensive for small companies. Not only is it the case that understanding legal requirements and building a framework of compliance checks to ensure that such legislation is adhered to is a complex process, but also, service providers such as banks require certification and reporting before they are willing to take on the risks associated with the adoption of applications from small application developers. In this paper, we propose a solution which provides a centralised Open Payments Ecosystem which supports compliance checking and allows for the matching of financial applications with service providers and programme managers, automatically providing risk analysis and reporting. The solution proposed combines static and dynamic verification in a real-life use case, which can shed new insights on the use of formal methods on large complex systems. We also report on the software engineering challenges encountered when analysing formal requirements arising from the needs of compliance to applicable legislation.

1 Introduction

Businesses often find themselves needing diverse ways of affecting or enabling payments in various contexts. As an example, consider a business providing a payment service to a travel agency to purchase flights, hotel bookings, etc. Having several such purchases from a single corporate card, particularly if that same card is also used for other purchases, would make reconciliation non-straightforward at best. On the other hand, providing one shot cards for use by the travel agency, which are cards that can be used once and disabled after the first purchase, makes reconciliation easier as only one purchase will be associated with any given card. However, for a business to set up such a payment programme, it is quite complex (implement cards processes for provisioning, reconciliation, dispute management, as well as creating a compliant application) and the costs may be prohibitive. In addition, negotiating with a bank or payment service provider in order to use their services to perform the actual financial movements and the resulting investment required to guarantee compliance to national legislation can be daunting. Even understanding the legal requirements is a major task, let alone the building of the necessary infrastructure to ensure compliance and to perform the risk analysis required by law and the banks providing the services.

* The Open Payments Ecosystem has received funding from the European Union's Horizon 2020 research and innovation programme under grant number 666363.

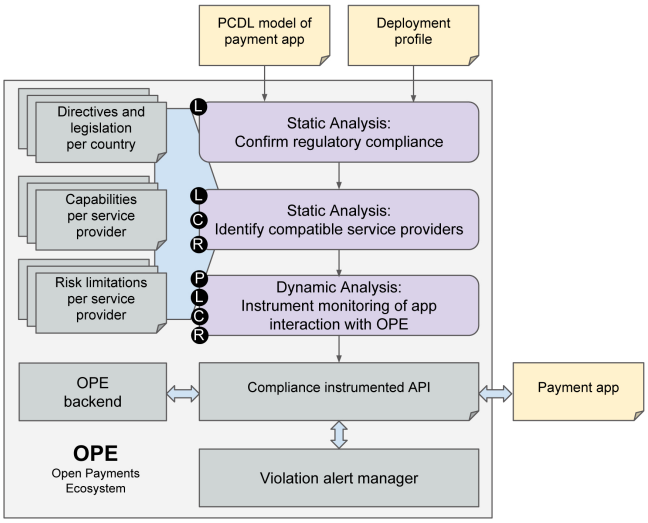

Fig. 1. The OPE compliance engine

In this paper, we present a proposed architecture which addresses these issues, and we outline the research challenges ahead in deploying such an architecture. It is of particular interest to the formal methods community in that it is a real-life challenge with a solution built around the possibilities opened by formal verification and analysis techniques. The solution is also a showcase of how formal methods are applicable to the challenging area of financial application compliance, ranging from risk monitoring and capabilities analysis to regulatory compliance.

Open Payments Ecosystem (OPE) aims at building an infrastructure to address this need by providing an execution environment for financial transactions. In order to support developers, OPE makes a *development environment* available with the necessary APIs for application development and service provider integration. The OPE itself does not hold funds (which legally, can only be held by a regulated institution). Therefore, payment applications are submitted to the OPE by developers together with a corresponding model. Programme managers, in turn, perform an automated *compliance check* on the model and, if successful, pair the adopted application with an integrated service provider. Since service providers are regulated institutions, this arrangement enables the application to be executed on the OPE platform. In this manner, the OPE brings together a number of players[3] including: (i) *service providers* (typically banks)

[3] In the rest of the paper, we will use the following terms:

Application: The artefact defining how instruments are issued and funds are moved.

Programme: An application deployed by a programme manager using a specific service provider(s).

Programme manager: An institution (not necessarily regulated) managing a number of programmes. Programme managers have contractual liability for the managed programmes.

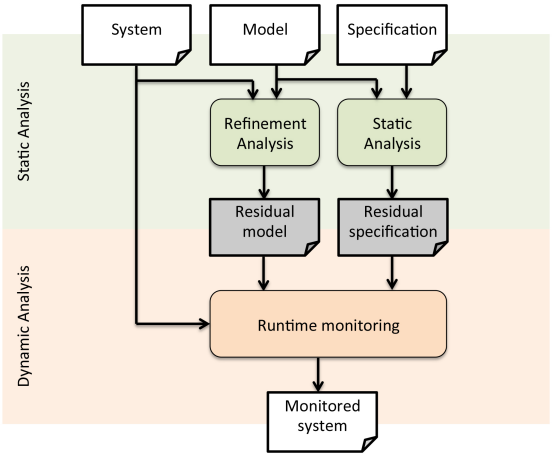

Fig. 2. The combination of static and dynamic analysis in the proposed architecture

which affect the underlying financial transactions; (ii) *developers* who create the payment applications; (iii) *corporate customers* (the travel agent in our example) who in turn provide the payment applications to their customers; and (iv) *programme managers* who take responsibility of putting end to end financial services (programmes) together — combining applications to service providers — and provide them to corporate customers.

2 Compliance Engine

The compliance subsystem at the core of OPE (shown in Fig. 1) has multiple roles: (i) it is used to support programme managers when matchmaking an application and a service provider; (ii) it ensures that a programme does not violate national legislation and regulations based on the location where it is planned to be deployed; and (iii) provides runtime monitoring on the running programme to continually check whether the monitored constraints are violated.

Since the OPE architecture envisages that the payment application is executed outside the platform (typically on the end user's device or a web server), only accessing the OPE platform through API calls, it is possible that the application submitted for validation and matched with an appropriate service provider is compromised or tampered with. Furthermore, providing compliance algorithms which support different programming languages and technologies which developers may adopt is not scalable in the long

Service Provider: A regulated institution providing some type of financial service to programme managers. Service providers have financial and regulatory liability for the services offered, some of which can be contractually transferred to programme managers while some would need to be evaluated against risk.

run. The solution to be adopted is that of having the developer submit a suitable model of the application behaviour, sufficiently detailed to enable verification and matching with the service providers. The *Payment Application Modelling Language* (PAML) is a domain-specific language being developed specifically to enable the description of a model of a payments application — its components and attributes, and constraints amongst them. This is used for the verification phase which will be performed when a developer submits an application, but in order to ensure that the application is, in fact, a faithful implementation of the PAML model submitted, the application's interaction with the OPE will be monitored at runtime to verify this. The compliance system thus combines static verification upon submission of an application with runtime or dynamic verification in a single framework as shown in Fig. 2. We envisage adopting techniques from recent work combining these two forms of verification [1, 2, 8, 4].

One of the major challenges is the diverse nature of compliance it attempts to address. We have identified four different types of constraints which need to be verified by the compliance engine:

Legal: In order to ensure interest from service providers, it is necessary that applications making use of payment accounts held by them, do not perform anything that is not allowed by regulations. For this reason, the compliance engine will require, in an encoded form, the definition and applicability of directives and legislation for different countries in which programmes may be deployed. For instance, UK e-money regulations require that funds on financial instruments must be redeemable at par value. This would require the compliance engine to (i) identify whether UK e-money legislation applies (confirm that the programme is regulated under UK law and that it deals with e-money), and if so; (ii) ensure that the application allows for redemption i.e. money remaining on a payment instrument (e.g. a card) after being closed can be withdrawn. However, checking that the amount of redemption is actually correct is not straightforward to check statically since the amount will depend on various runtime variables, thus splitting this compliance check into a static and a runtime verification component. The formalisation of the nuances of legislation is far from being straightforward, although we are considering techniques such as those developed in [3] to support this transformation, or validate it.

Capabilities: Service providers may have different capabilities e.g. in terms of what types of cards they can issue or amount of money they are able to handle, which means that when matching an application with potential service providers, the analysis must also cater for capabilities. These correspond more closely with traditional software engineering requirements, and we envisage that we can adopt existing techniques e.g. type systems, to enable their verification.

Risks: Service providers may also have guidelines in terms of risks they are willing to take. For instance, they may want a payment application not to issue more than a certain number of cards per person, or not to process more than a particular amount of money. As in the case of capabilities, this corresponds closely to business logic rules from financial transaction applications e.g. [6]. However, unlike regular properties, violation of which would require action to be taken, in this case, monitoring the properties as they approach thresholds, and giving the service providers the facility to act as they deem appropriate when these guidelines are not adhered to, is a

more appropriate approach. For this reason, the compliance engine will be split into two tiers, one layer which computes these statistics (using an approach similar to [5]), and another layer processes them, generating alerts, notifications and actions automatically.

PAML compliance: Finally already discussed, some of the compliance checking will happen statically against the PAML model of an application. To ensure that this compliance holds for the application, the engine will also need to monitor the applications' behaviour to ensure that is faithful to the model describing it.

Fig. 1 shows the architecture of this compliance engine, illustrating also at which stage the previous constraints are tackled (where L corresponds to legal constraints, C to capabilities, R to risks, and P to PAML compliance).

3 Formalising Regulatory Compliance

One of the major challenges of setting up the compliance engine is that of capturing what needs to be checked to ensure adherence to applicable regulations. To start with, the regulation come in the form of a number of regulatory texts, each consisting of a number of pages full of legal jargon. Secondly, regulations typically allow grey areas which leave room for interpretation. Furthermore, the lawyers who are familiar with the regulations are not technical people. In this context, we chose to apply an iterative process typical in software engineering practices [7] to ensure that the communication process between lawyers and developers does not leave room for misunderstandings. The steps we adopted were as follows:

Annotating regulations As a first step, lawyers provided the technical team with the regulatory texts annotated with comments and explanations. The comments mainly consisted of their opinion on whether the regulation falls within the scope of the compliance engine. The explanations were aimed at clarifying jargon or providing an interpretation when the regulations left was room for one.

Shortlisting of applicable regulations The technical team went through the annotated regulations, and for each regulation decided whether or not it falls within the scope of the compliance engine and in what way: which rules could be checked statically, which could be checked at runtime, etc. Once this list was compiled, it was discussed with the lawyers, having the technical team explaining the reasoning behind the adopted classification.

Summarising the shortlisted regulations Each shorlisted regulation was summarised in a few words by the technical team, trying as far as possible to avoid ambiguity. These summaries were once more communicated to the lawyers and any disagreements on the choice of words were discussed.

Formalising the regulations For each concisely described regulation approved by the lawyers, the technical team formulated the corresponding mathematics from which it is now straightforward to turn into code which automatically checks compliance. Once more, any questions which arose during formalisation process were discussed with the lawyers.

Following this iterative process, involving regular discussions based on evolving documentation, proved to be effective, leaving both the lawyers and technical team feeling confident the interpretation of the legal texts and the semantics of the formal specifications coincided.

4 Conclusions

Work on the OPE is still ongoing. We are currently completing the analysis of applicable legislation and incorporating these regulations within the compliance engine. Parts of this compliance process can be resolved statically, while other parts can only be verified dynamically at runtime. What we are currently looking into is how the two analysis approaches can be combined together in order to (i) enable the static checks to classify financial applications and match them with appropriate programme managers and service providers; (ii) ensure that none of the parts of the regulations which cannot be verified statically are violated at runtime through the use of runtime verification. Given the real-time nature and volume of transactions, it is crucial that runtime checks are reduced to a minimum, which can only be achieved by trying to verify as many of the properties as possible statically.

Compared to the current state of the art where banks have to manage risks by approving programs manually and requesting reports at particular intervals, the OPE will be offloading significant overheads off the risk takers, namely the banks, making the setting up and management of payment applications significantly more feasible. It is still be assessed to what degree the pre-deployment static analysis of the compliance process can alleviate the runtime overheads.

References

1. W. Ahrendt, J. M. Chimento, G. J. Pace, and G. Schneider. A specification language for static and runtime verification of data and control properties. In *FM'15*, volume 9109, pages 108–125. 2015.
2. W. Ahrendt, G. Pace, and G. Schneider. A Unified Approach for Static and Runtime Verification: Framework and Applications. In *ISOLA'12*, LNCS 7609. 2012.
3. S. Azzopardi, A. Gatt, and G. J. Pace. Formally analysing natural language contracts. In *Computer Science Annual Workshop 2015*, 2015.
4. E. Bodden, P. Lam, and L. J. Hendren. Clara: A framework for partially evaluating finite-state runtime monitors ahead of time. In *RV'10*, volume 6418 of *LNCS*, pages 183–197, 2010.
5. C. Colombo, A. Gauci, and G. J. Pace. Larvastat: Monitoring of statistical properties. In *Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, pages 480–484, 2010.
6. C. Colombo, G. J. Pace, and P. Abela. Safer asynchronous runtime monitoring using compensations. *Formal Methods in System Design*, 41(3):269–294, 2012.
7. C. Larman and V. R. Basili. Iterative and incremental developments. a brief history. *Computer*, 36(6):47–56, 2003.
8. D. Wonisch, A. Schremmer, and H. Wehrheim. Zero Overhead Runtime Monitoring. In *SEFM'13*, volume 8137 of *LNCS*, pages 244–258. Springer Berlin Heidelberg, 2013.