Feature Interaction Problems in Smart Cards with Dynamic Application Lifecycle and their Countermeasures

***Abstract*—Smart cards, in their traditional deployment architecture referred to as the Issuer Centric Smart Card Ownership Model (ICOM), have a restricted application lifecycle. In this model, an application is installed onto a smart card by the relevant card issuer. In most cases, the card issuer is also the centralised controlling authority for different lifecycle stages of the application. The installed application might not be deleted for the whole duration of the smart card's operation. The interaction of the application is controlled and closely monitored by the card issuer. ICOM-based smart cards have only one application, in the majority of deployments. Therefore, the likelihood of Feature Interaction Problems (FIPs) is minimal. By contrast, in open and dynamic smart card models like the GlobalPlatform Consumer-Centric Model (GP-CCM), and the User Centric Smart Card Ownership Model (UCOM) the probability of FIPs is substantially higher. The nature of these models allows users to install and delete applications as they require. This change in the application landscape might create a situation in which features (functions) of an application do not execute properly due to their reliance on an application that might be removed by the user or updated/modified by the Service Provider (SP). In this paper, we will focus on the problems related to application deletion that can potentially cause an FIP on the smart card platform. The paper proposes a framework to minimise such problems so the users can gain maximum service from their device and "freedom of choice" without concerns about application interdependencies.**

I. Introduction

On any computing device, individual applications consist of multiple functions. These functions have intrinsic dependence on each other in the context of the application, whereas different applications might rely on each other or on different services provided by the platform (Operating System: OS and Hardware). These dependencies are the result of modular designs that have significantly helped in enabling easy development and deployment of various applications to a wide range of computing devices. However, such dependencies in any context, including application-to-application, application to Application Programming Interfaces (APIs), and application-to-OS, can cause Feature Interaction Problems (FIPs).

The narrow definition of an FIP is where "two or more features (functions, applications, and/or Application Programming Interfaces: APIs, etc.) in a close coupling (interaction) reach a stage when change, modification or interruption to one of the features adversely affect the overall stability and/or reliable execution of others" [1–4]. In any dynamic environment, applications are constantly being installed or removed in addition to being updated and modified. Occurrence of FIPs in such an environment is a consequence of the openness and dynamism of that environment.

Traditional smart cards are under centralised control, usually in the hands of the card issuer who acquires them from the manufacturer and issues them to individual users [5]. The card issuer has complete control of the smart card, including the hardware, Operating System (OS) and application(s). In this model, also referred to as ICOM [5] applications are not deleted or installed very often. Furthermore, the card issuer can vet off-card the compatibilities and dependencies between applications and the OS. Finally, a substantial number of smart cards deployed under this model only have one application, whose lifetime is directly related to the lifetime of the respective smart card. Therefore, consideration of FIPs is not necessary; however, this cannot be assumed for future smart card models [6]. The rationale behind this is based on the growing trend in the smart card industry towards multitenancy on multi-application smart cards — multitenancy by different non-related organisations.

Smart card models like GP-CCM [7] and UCOM [5] allow users to install and delete applications to varying degrees. Where applications from varied providers can be installed there is the potential for compatibility issues. Furthermore, different applications might have off-card relationships between their providers so that that they share resources with each other on a smart card. Such a mechanism is referred to as application sharing [8]. If one of the applications is modified or updated, it might have an adverse effect on the functioning of the dependent applications. Similarly, a Smart Card Operating System (SCOS) or application update might make certain functions incompatible with each other. Such issues arise due to the openness and dynamism of the proposed models and are categorised as FIPs for smart cards. Resolving these issues would be an important step towards a fully dynamic multitenancy multi-application smart cards.

A. Contribution of Paper

In this paper, we focus on only one aspect of FIPs for smart cards: application deletion. The application deletion process has the potential to introduce interdependency issues that might create FIPs for the remaining applications on the card. While other FIP-related scenarios are also valid and require in-depth analysis, they are beyond the scope of this paper. To minimise any FIP-based issues related to application deletion, we propose a potential solution for smart cards (regardless of their use of ICOM, GP-CCM or UCOM). We also compare our proposed framework with the existing frameworks that deals with application deletion.

B. Structure of Paper

In section II, we briefly discuss different smart card-based models. Section III then defines the term "FIP" in the context of this paper, assessing what actions can cause this, and their relevance to the smart card architecture. In section IV, we present a short survey of how this problem is dealt with existing architectures. In section V, we propose a framework as a potential solution for application deletion in a manner that avoids FIPs. Furthermore, in the same section, we analyse the proposal and compare it with existing solutions. Finally, in section VI we discuss future research directions and conclude the paper.

II. Smart Card Ecosystems

From their inception at the end of the 1970s up to the creation of multiapplication platforms at the end of the 1990s, smart cards generally supported only one application, issued for a specific purpose by the card issuer. However, multiapplication technologies [9], like Java Card [10], Multos [11] and a few others no longer in use, have enabled the emergence of new species in the smart card ecosystem, including application providers (also refered as Service Provider: SP) being able to develop applications to load onto cards post-issuance. New ownership models have been proposed ranging from those in which the smart cards' issuer can share the smart cards with partners, to those in which the cardholder is the owner of her card and as a result administrate it according to her needs and wishes.

A. Issuer Centric Smart Cards

Issuer Centric Smart Cards [5] are the most commonly deployed smart cards for historical, branding, business and technical reasons. As discussed, historically, smart cards supported only one purpose-specific application issued by the card issuer.

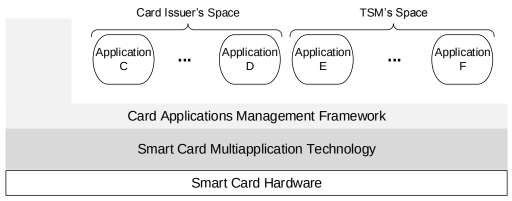


Figure 1. Issuer Centric Smart Card Architecture

On the branding side, even if new technologies have enabled the support of several applications, smart card issuers still wanted to keep their brands on their cards and did not want to share this space on the plastic card's body with other actors.

However, with the development of Near Field Communication (NFC) technology, cards are now more often embedded in terminals (e.g. phones) and the branding aspect is less of a problem since the cards are hidden.

For business reasons, smart card issuers want to keep control of the markets where smart cards are used. Since they are looking for new sources of revenue, they sign partnership agreements with application providers to lease them space on their smart cards. However, the key expertise of smart card issuers is not the management of post-issuance deployment of applications. To leverage the use of multiapplication technologies, NFC technology has offered a technical solution in the form of the Trusted Service Manager (TSM) architecture [15].

This model for post-issuance deployment of applications does not stipulate who must be the TSM and it provides two main features: security of deployed cards is maintained as long as the TSM is a trusted party for the smart card issuers; and smart card issuers can still get revenue according to the terms of agreement signed with the TSM.

As illustrated in Figure 1, Issuer Centric Smart Cards are thus owned by an issuer who controls them and may authorise some partners, through the TSM's authority, to install applications. In this context, the cardholder has no say on the contents of her smart card. The on-card application management framework depends on the underlying smart card multiapplication technology; for example, GlobalPlatform [22].

On these smart cards, the trust relationships are the following:

- smart card issuers trust their cards;
- the TSM trusts the smart cards (i.e. it can be sure that the card is a real card and not a simulation [12]) because of the agreement they have signed with the smart card issuers;
- smart card issuers trust the applications installed under the authority of the TSM because of the agreement they have signed, since the TSM has also signed an agreement with the application providers.

B. User Centric Smart Cards

User Centric Smart Cards [5] are not yet commonly deployed but they are very promising, since they empower cardholders with the ability to control the contents of their cards whilst still ensuring a high level of security for the TSMs and application providers installing their applications on these cards.

As illustrated in Figure 2, the Trusted Environment & Execution Manager (TEM) [13] is the key component of this ownership model in coping with the threats associated with a more open platform and in guaranteeing trust at the cardholder, TSM and application provider levels.

These cards can be integrated with any models of deployment, although the CO-TSM [14] is one of the most appropriate in terms of maximal flexibility, scalability and choices of applications for the user.

In addition the UCOM can accommodate a coopetitive architecture [15], which can be seen as a compromise between the desire for control that the users aspire to obtain and that

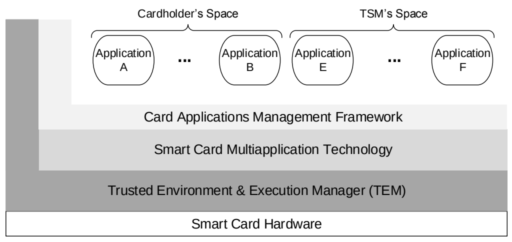

Cardholder's Space
TSM's Space
Application A
...
Application B
Application E
...
Application F
Card Applications Management Framework
Smart Card Multiapplication Technology
Trusted Environment & Execution Manager (TEM)
Smart Card Hardware

Figure 2. User Centric Smart Card Architecture

the issuers want to keep. Moreover, as illustrated in Figure 3, compared to the traditional ICOM, the TEM reinforces trust for each stakeholder; extending the present trust frameworks.

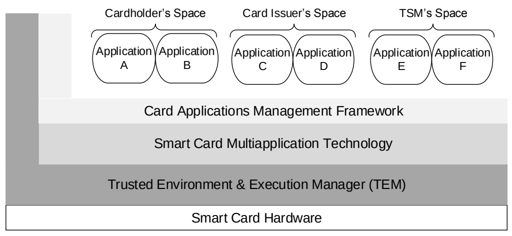


Figure 3. Coopetitive Smart Card Architecture

C. GlobalPlatform Consumer-Centric Smart Cards

Due to the commercial interests involved in empowering cardholders to choose applications they want on their cards, GlobalPlatform has produced a white paper [7] to indicate the main intent of their future Consumer-Centric Model. Based on this white paper, the architecture of GP-CCM Smart Card is presented in Figure 4. It can be seen that the cardholder's space is under the supervision of the card issuer's space since there is some direct communication between Application Providers (or their TSM) and the card issuer (called the Trusted Token Provider). In a similar way, installation can be authorised if there is some pre-established business relationship between the Application Provider and the card issuer.

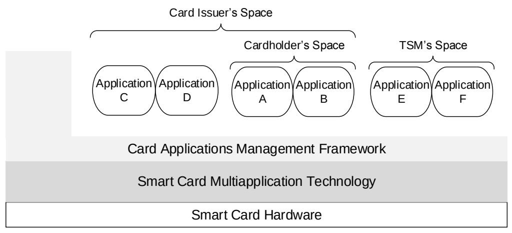


Figure 4. GP-CCM Smart Card Architecture

III. Feature Interaction Problem

In this section, we discuss the FIP, and why it has become a crucial issue for smart cards.

An FIP arises when multiple features in a system depend on each other. Change or modification to one of these features has the potential to affect the overall system. Early work on FIPs can be traced to the telecommunication industry [16] where services from different providers interacted with each other.

In the early days of computers, these devices were built from scratch for specific purposes. Therefore, we can assume that as the entire development of the system was carried out by the same organisation, the likelihood of FIPs being a crucial issue would be minimal. However, over time, the modular design paradigm became the centrepiece of computer architecture. From hardware to OS to user applications, all of them were built to be as modular as possible.

This urge for modularity created dependencies between different features, where each of these features might be developed and managed by different organisations that might not even have a relationship between them. As stated by Apel et al. in [1], an FIP occurs when the behaviour of one feature is influenced by the presence (or lack) of another feature (or set of other features). Therefore, on one hand we would like to build a computer system that has a high level of modularity to foster innovation and enable a user to gain access to a wide range of services, but on the other hand such modularity increases the potential of FIPs in the overall system [17–19].

As discussed previously, early smart card [9, 20, 21] architectures were based on direct integration of an application to the core of the SCOS. The term SCOS is used in a loose sense as these operating systems on early smart cards were built to the requirements of the application that was to be hosted on them [20]. However, this changed with the advent of SCOSs like Multos [11] and Java Card [10]. These SCOSs allowed multiple applications on a smart card; however, in actual deployment such smart cards have been always under the control of a centralised authority. This authority can vet individual applications prior to installation and can also intrinsically resolve any potential FIP concerns.

This changed with the advent of NFC and proposals like GP-CCM and UCOM that put forward the idea of an open and dynamic smart card architecture. In such an architecture, we have the same situation as in general purpose computing, in which users can download and install applications as they desire. These applications have a set of their own features; some depend on the underlying OS, libraries, and potentially other applications on the respective platform. Changes to any of these features have the potential to create negative effects on the overall performance, reliability and security of the application and/or platform [4].

The smart card industry and its deployment environment is very sensitive to security, reliability and performance issues. Therefore, to make GP-CCM, UCOM and even ICOM (with or without TSM) robust solutions, we need to build an architecture that can withstand any potential concerns in relation to FIPs. However, this paper is restricted solely to the FIP due to the application deletion process. To allow a user to install any application as she desires, she might wish the same for application deletion. Therefore, a situation in which a user cannot delete an application because deleting it might affect other applications might be not prefereable.

For example, application developers might try to restrict users from deleting their applications by making the applications they heavily depend on each other. In this situation, existing architectures discussed in section IV will not delete these applications.

Both of the proposed models, GP-CCM and UCOM, do not place on any restriction on user's choices on the platform in any way by any application. Therefore, in this proposal we try to minimise such possibilities. However, this cannot be assumed for the existing architectures, as discussed in the next section.

IV. Survey of Countermeasures to Feature Interaction Problems

In this section, we will discuss how existing frameworks like GlobalPlatform, Multos and Java Card propose to resolve any FIP issues arising due to the application deletion process.

A. GlobalPlatform's Approach

The application deletion process in the GlobalPlatform card specification can be initiated by any entity that has the right to execute a delete command. An application provider or a card issuer (TSM) can issue a delete command that is accompanied by mandatory authorisation parameters to authenticate to the respective smart card(s). The deletion process is handled by the OPEN framework of the GlobalPlatform specification and it performs several checks before proceeding with the deletion of an application. The checks include verifying the deletion request (e.g. delete token [22]), confirming whether the application requested for deletion is referenced by another application, and other optional housekeeping checks. If these verifications fail, the GlobalPlatform specification states that the deletion process should be terminated. We have two concerns with the GlobalPlatform deletion process. Firstly, how it can determine that an application is referenced, which is generally part of the application sharing mechanism? As stated in the GlobalPlatform card specification [22], the specification relies on the underlying platform implementation for the application sharing mechanism. Therefore, this test for the deletion process requires the support of the underlying platform's application sharing mechanism [8]. Secondly, if an application is referenced, then why terminate the deletion process? Instead, one could resolve the interdependencies and then proceed with the deletion process. Unfortunately, the GlobalPlatform card specification does not detail the resolution of interdependencies among different applications on a smart card.

B. Multos' Approach

In Multos, the application deletion process is the same as the application loading depicted in Figure 5 and described in detail in [14]. The only differences are that in the deletion process, there is no application load unit generator, and an application provider or a card issuer requests the Multos Certification Authority for the application deletion certificate instead of the application load certificate [23]. The on-card deletion process simply deletes the application data and code. As applications on a Multos card do not have interdependencies, the deletion process does not need to be concerned about FIPs. In the application sharing mechanism on Multos cards [8], a client application may still make a delegation request. However, because it is just an APDU message, the delegation mechanism can return an error that should be handled by the client application.

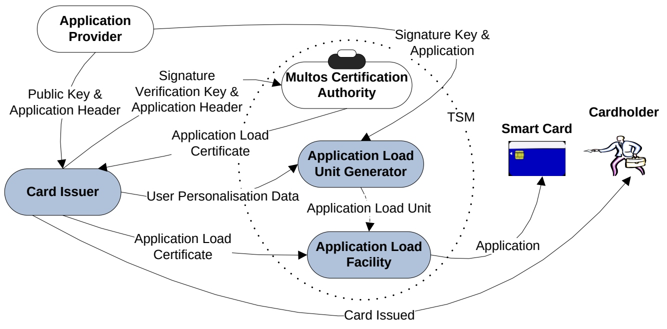

Application Provider
Signature Key & Application
Public Key & Application Header
Signature Verification Key & Application Header
Multos Certification Authority
TSM
Cardholder
Smart Card
Application Load Certificate
Card Issuer
Application Load Unit Generator
User Personalisation Data
Application Load Unit
Application Load Certificate
Application Load Facility
Application
Card Issued

Figure 5. Multos Application Loading Framework [14]

C. Java Card's Approach

The Java Card 2.x and 3.x classic editions have similar schemes, as detailed in the GlobalPlatform card specification. The Java Card specification [24] stipulates that the Java Card Runtime Environment (JCRE) should not attempt to delete an application if it is being referenced another application. However, the Java Card 3.x connected edition extends the deletion framework and attempts to resolve the interdependencies among different applications. The Java Card 3.x connected edition's application deletion mechanism is based on events and associated listeners. The events mechanism enables an application to register/un-register itself for events generated by other applications, and also enables it to generate similar events. The connected edition defines an event for application deletion as an "application instance deletion request" event (`event:///standard/app/deleted`) [10]. By doing so, a client application can register itself to the deletion events of a server application. Therefore, when the server application is requested to be deleted by an authorised entity, the card manager of a Java Card will instruct the server application regarding the deletion request that in return can signal the deletion event. The client application, on receipt of such event, can perform the tasks needed to remove the dependencies on the server application. The card manager will then proceed with checking whether there are any applications that still have dependencies on the server application. If the dependencies of such applications cannot be removed, the card manager will terminate the deletion process. One thing to note is that it is optional for server and client applications to register, signal and manage any events.

The deletion mechanism for the Java Card 3.x connected edition is a positive step towards providing an architecture where application installation and deletion will be more common than before. The deletion mechanism for the UCOM

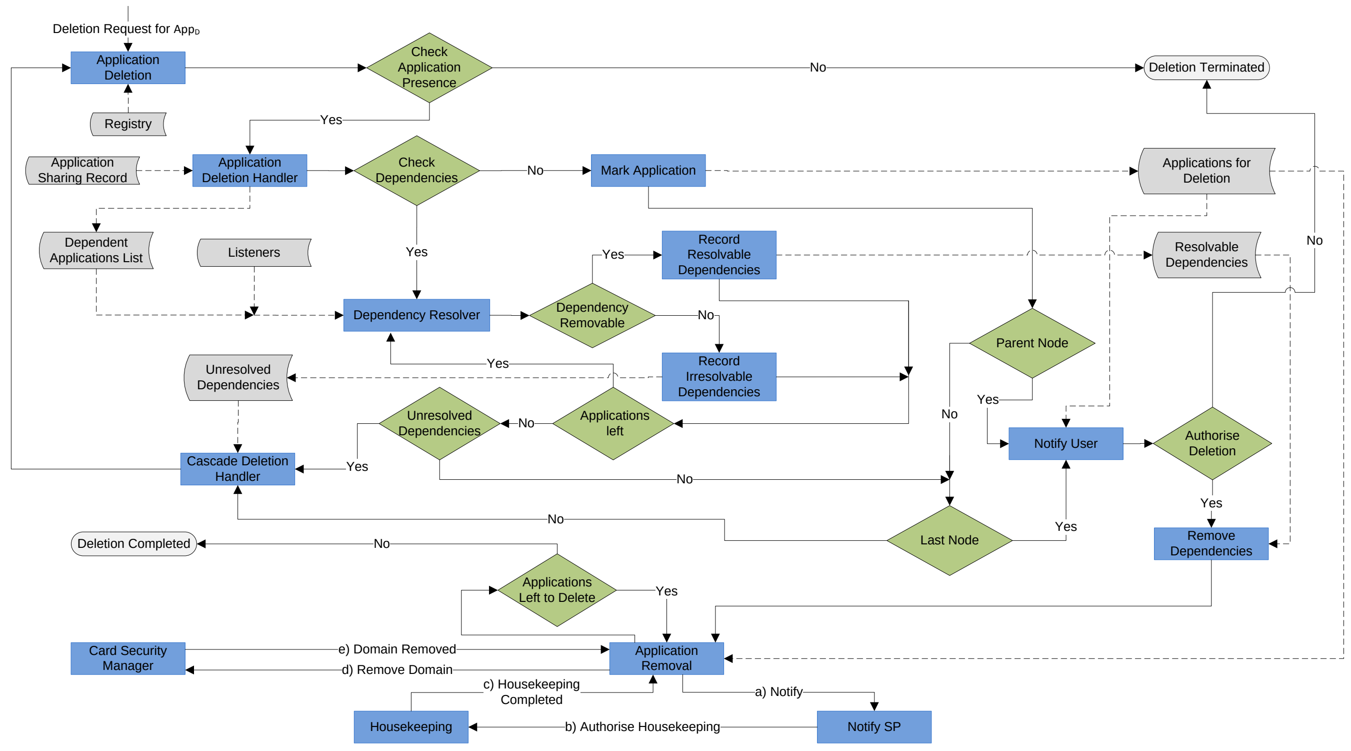


Figure 6. Proposed Application Deletion Process to avoid FIPs in Smart Cards with Dynamic Application Lifecycle (e.g. GP-CCM and UCOM)

architecture is based on the Java Card 3.x connected edition with compulsory components and includes provision for a cascade deletion process.

V. Feature Interaction Resolution Architecture

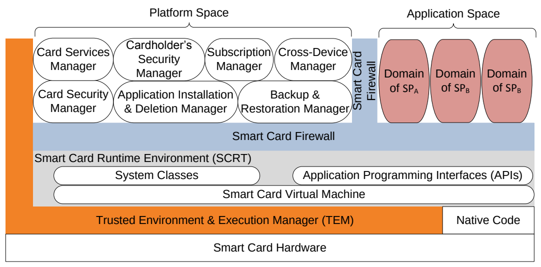
In this section, we discuss the proposed architecture to minimise (if not completely remove) the probability of FIP due to application deletion processes on GP-CCM, UCOM and ICOM (with or without TSM) supported smart cards.

A. Proposed Architecture

The deletion process in UCOM [25] is based on the Java Card 3.x connected edition specification, which is extended by a cascade deletion mechanism. Cascade deletion enables a smart card to proceed with the deletion of any dependent applications if their dependencies cannot be resolved in a satisfactory manner. A smart card can only proceed with cascade deletion if the cardholder explicitly sanctions it. The deletion process for the UCOM is illustrated in Figure 6 and described below.

In Figure 6, the hard-bordered rectangles represent operations and the rhombus shapes represent the if-then-else conditional statement that is part of the operation that precedes it. The quadrilaterals with curved vertical sides represent the data structures (e.g. files). The dotted lines represent data read or write operations: if the arrowhead points to the data structure then it is a write operation; otherwise, it is a read operation. During the description of the deletion process, we italicise individual operations (e.g. *operation*) and refer to a data structure with double quotes (e.g. “data structure”). Most of the processes represented in the Figure 6 are part of the application installation & deletion manager illustrated in Figure 7.

On receipt of the application deletion request from either the user or the SP, the *application deletion* will first check whether the application is installed on the smart card. For illustration, we call the application that is requested to be deleted App$_D$. If App$_D$ is present on the card, then it will be registered as an installed application in the “registry” maintained by the application installation & deletion manager. In this case, when App$_D$ is present, the request is forwarded to the *application deletion handler*, which will retrieve the “application sharing record” maintained by the smart card firewall and check whether App$_D$ has any dependent applications.



Platform Space
Application Space
Card Services Manager
Cardholder's Security Manager
Subscription Manager
Cross-Device Manager
Smart Card Firewall
Domain of SP$_A$
Domain of SP$_B$
Domain of SP$_B$
Card Security Manager
Application Installation & Deletion Manager
Backup & Restoration Manager
Smart Card Firewall
Smart Card Runtime Environment (SCRT)
System Classes
Application Programming Interfaces (APIs)
Smart Card Virtual Machine
Trusted Environment & Execution Manager (TEM)
Native Code
Smart Card Hardware

Figure 7. UCOM Supported Smart Card Architecture [13–15, 26]

If the application does not have any dependent applications, the *mark application* gathers the application-related information and records it in the file “applications for deletion”. Next, it checks that the application is not part of any application-sharing tree — meaning there are no application dependencies to resolve. In this scenario, it might seem a redundant check but this step will become necessary when App_D has dependencies. In the next step, the user is notified that the smart card is ready to delete the application App_D. If the user authorises it, first the *remove dependencies* removes any dependencies, which is followed by the *application removal*. The *application removal* will first notify the SP (*notify SP*), which might perform some *housekeeping* tasks like depersonalisation of the application or/and transfer of any log files. Depersonalisation of the application involves removal of all user-related data along with any cryptographic material. Furthermore, transfer of the application log files [27] to the SP. We have to retain this as it might contain the usage information of the application, which might be necessary for fraud prevention or detection along with evidence of certain activities. After the housekeeping is completed, the *application removal* requests the *card security manager* (Figure 7) to delete the domain credentials and reclaim the memory. After a successful outcome, the *application removal* checks whether there are any more applications to delete. If not, then the deletion process will terminate.

In the second case, if the *application deletion handler* finds dependencies, then it will generate the "dependent application list". The *dependency resolver* will take the list of applications that are dependent and also ones that are registered as "listeners" to the deletion request for App_D. The *dependency resolver* generates the deletion event for App_D and notifies all applications that are registered to this event. If the applications can resolve the dependency then it will record them as "resolvable dependencies". For this step, we rely on the dependent application's response, which might be malicious. If a dependent application App_C signals that it can remove the dependency, but does not take any action regarding this, its aim might be to use the reference to App_D for some malicious purpose. However, we protect the platform from such eventualities: the firewall mechanism will also remove the record that App_C is authorised to access App_D reference (a memory reference to App_D's application resource manager: Figure 8). The firewall mechanism can effectively prevent the memory access; however, the main aim of dependency resolution is to avoid any eventualities in which a dependent application might not be able to execute reliably in the future.

The *dependency resolver* will keep on iterating through the "dependent applications list" until it reaches the end of it. It will then check whether there are any "unresolved dependencies". If yes, then it moves to the *cascade deletion handler*; otherwise, it will check whether it is at the last application in the "applications for deletion". The *cascade deletion handler* takes the "unresolved dependencies" list and iterates through it, signalling the deletion of the respective applications. For each application, a dependency analysis is performed. This process is iterated until the list of all applications required to be deleted is generated: "applications for deletion". At this stage, the user is notified by the *notify user* and the "applications for deletion" is communicated. If the user authorises the card to go ahead with the deletion of the applications, the *remove dependencies* will first remove any dependencies. A point to note is that we leave the dependencies removal process to the end: if the user does not authorise the deletion, at least we will not delete any application sharing instances. Therefore, before the user authorises the deletion, the entire process attempts to find dependent applications and points out to the user the list of applications that cannot resolve their dependencies on App_D. The *application removal* process will then iterate through the "applications for deletion" and delete them one at a time.

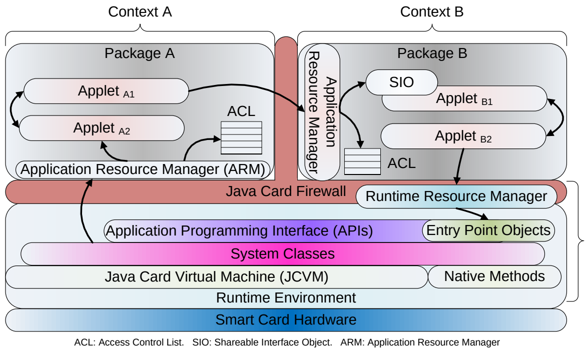

Context A
Context B
Package A
Package B
Applet A1
Applet A2
ACL
Application Resource Manager (ARM)
Application Resource Manager
SIO
Applet B1
Applet B2
ACL
Java Card Firewall
Runtime Resource Manager
Application Programming Interface (APIs)
Entry Point Objects
System Classes
Java Card Virtual Machine (JCVM)
Native Methods
Runtime Environment
Smart Card Hardware
System Context
ACL: Access Control List. SIO: Shareable Interface Object. ARM: Application Resource Manager

Figure 8. UCOM Supported Smart Card Application Architecture and Application Sharing Framework (for details please see [8])

In cases where the deletion request was initiated by the SP of App_D, and it requires deletion of other applications that do not belong to the SP, the user will still be notified. If the user opts for not deleting it, the SP can then proceed with blocking App_D. In the blocked state, an application is not accessible to the user; however, dependent applications can still access it through the application sharing mechanism.

As discussed before, the UCOM deletion process only provides dependent applications with an opportunity to gracefully resolve their dependencies. If an application does not have such a mechanism, the UCOM deletion process marks that application for deletion. Furthermore, during the deletion process App_D's resource manager, which maintains access to the application via the smart card firewall, is removed. Thus, if a dependent application tries to access App_D's resources, the firewall mechanism will reject that request. If the application does not gracefully proceed after the firewall rejects its request, the card security manager can either block the application or mark it for deletion. Therefore, any application that affects the reliability of the smart card platform will be removed or at least blocked by the card security manager.

B. Analysis of the Architecture

The basic ethos of the user- or consumer- centric design is to give the right of decision-making to individual users. This is the same in both the GP-CCM and UCOM proposals. In our proposal, the user has complete freedom to choose any application she wishes to be removed from her smart card.

There can be other applications that might be reliant on the application selected for deletion. In this situation, the proposed architecture follows the same ideology, empowering the user and other applications to make their decisions. By empowering

Table I
COMPARISON BETWEEN EXISTING AND PROPOSED FRAMEWORK.

Features	**Multos**	**Java Card 3.x**	**GlobalPlatform**	**Proposed Framework**
Inter-dependencies	No	Yes	Yes	Yes
Application Deletion	Yes	Yes	Yes	Yes
Dependencies Discovery	No	Yes	Yes	Yes
Dependencies Resolution	No	No	No	Yes
Application Deletion with Dependencies	No	No	No	Yes
Dependent Application Notification	No	Yes	No	Yes
Dependent Application Protection	No	No	No	Yes

other applications, we mean informing the dependent applications that some of their features that might be dependent on the application selected for deletion may not function properly. Note that it is the responsibility of the application developers of the dependent application to have a contingency plan for such situations. Furthermore, if dependent applications do not have such a mechanism and inform the SCOS that they cannot manage this feature change, then the decision is given to the smart card user. The user can either choose not to delete the relevant application or to select the deletion of all dependent applications that have notified the SCOS that they cannot guarantee their safe and reliable execution.

C. Comparison with Existing Proposals

In this section, we compare the exiting proposals application deletion and how to resolve the application inter-dependencies with our proposal in this paper. The comparison is shown in the table I.

Except Multos, all other proposals allow applications to have inter-dependencies with each other. Where application deletion is permitted in all of the platforms listed in table I. However, only the proposed framework in this paper facilitates the discovery and resolution of such dependencies. Java Card and GlobalPlatform perform the discovery but if an application has inter-dependencies then such an application is not deleted. Therefore, there is no dependence resolutions for the successful deletion of the respective application. Except our proposed framework, no other existing framework provides application deletion mechanism for inter-dependent applications. Furthermore, the proposal in this paper also notifies the dependent application about the deletion, so it can adequate manage its dependencies. If the dependent application cannot manage its dependencies then it can inform the application deletion handler, which might proceed with its deletion. However, for instance if a dependent application informs the application deletion handler that it can reliably function even after it deletes the application on which it depends. Then later on tries to access it (using application sharing mechanism) the UCOM proposed firewall will prohibit such requests and also mark it for non-conformation to the platform's security policies. Depending upon the security policies the application might then be deleted from the smart card. The firewall permissions that overseas the application sharing mechanism are updated by the application deletion handler while deleting an application and/or resolving dependencies.

VI. FUTURE RESEARCH DIRECTIONS AND CONCLUSION

In this paper, we looked at the FIP in the context of smart card technology. Upto now the FIP has not received substantial attention from either the smart card industry or academia. There is a valid reason for this and it is entrenched in the traditional deployment architecture of smart cards: ICOM. However, with the changes introduced by GP-CCM and UCOM, concerns in relation to FIPs have the potential to increase. In this paper, we have only dealt with a particular aspect of the FIP in connection with the application deletion process on a smart card. The proposed architecture for application deletion tries to resolve all potential feature dependencies to ensure that after the relevant application is deleted, other applications and SCOS are not affected, especially in terms of reliability, and security. In future work, we would like to explore additional aspects of FIPs in an open and dynamic environment like GP-CCM and UCOM. This is considered to be pivotal for the success of these models.

REFERENCES

[1] S. Apel, S. Kolesnikov, N. Siegmund, C. Kästner, and B. Garvin, "Exploring Feature Interactions in the Wild: The New Feature-interaction Challenge," in *Proceedings of the 5th International Workshop on Feature-Oriented Software Development*, NY, USA: ACM, 2013.

[2] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec, "Feature Interaction: A Critical Review and Considered Forecast," *Comput. Netw.*, vol. 41, no. 1, pp. 115–141, Jan. 2003.

[3] Q. Zhao, J. Huang, X. Chen, and G. Huang, "Feature Interaction Problems in Web-based Service Composition - (position paper)," in *Feature Interactions in Software and Communication Systems X,*, M. Nakamura and S. Reiff-Marganiec, Eds. Lisbon, Portugal: IOS Press, June 2009.

[4] R. L. A. Nhlabatsi and B. Nuseibeh, "Feature interaction: The Security Threat from within Software Systems," *Progress in Informatics*, no. 5, pp. 75–89, 2008.

[5] R. N. Akram, K. Markantonakis, and K. Mayes, "A Paradigm Shift in Smart Card Ownership Model," in *Proceedings of the 2010 International Conference on Computational Science and Its Applications*, Bernady O. Apduhan, Osvaldo Gervasi, Andres Iglesias, D. Taniar, and M. Gavrilova, Eds. Fukuoka, Japan: IEEE CS, March 2010.

[6] R. N. Akram and K. Markantonakis, "Smart Cards: State-of-the-Art to Future Directions, Invited Paper," in *IEEE International Symposium on Signal Processing and Information Technology*, C. Douligeris and D. Serpanos, Eds. Athens, Greece: IEEE CS, December 2013.

[7] "GlobalPlatform A New Model: The Consumer-Centric Model and How It Applies to the Mobile Ecosystem," Whitepaper, March 2012.

[8] R. N. Akram, K. Markantonakis, and K. Mayes, "Firewall Mechanism in a User Centric Smart Card Ownership Model," in *Smart Card Research and Advanced Application, 9th IFIP WG 8.8/11.2 International Conference*, D. Gollmann, J.-L. Lanet, and J. Iguchi-Cartigny, Eds., vol. 6035/2010. Passau, Germany: Springer, April 2010, pp. 118–132.

[9] D. Sauveron, "Multiapplication Smart Card: Towards an Open Smart Card?" *Inf. Secur. Tech. Rep.*, vol. 14, no. 2, pp. 70–78, 2009.

[10] *Java Card Platform Specifications*, Oracle Std. V3.0.1, May 2009.

[11] *Multos: The Multos Specification,*, Online, Std.

[12] R. N. Akram, K. Markantonakis, and K. Mayes, "Simulator Problem in User Centric Smart Card Ownership Model," in *6th IEEE/IFIP International Symposium on Trusted Computing and Communications*, H. Y. Tang and X. Fu, Eds. HK, China: IEEE CS, December 2010.

[13] ——, "Trusted Platform Module for Smart Cards," in *6th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, O. Alfandi, Ed. Dubai, UAE: IEEE CS, March 2014.

[14] R. N. Akram, K. Markantonakis, and D. Sauveron, "Collaborative and Ubiquitous Consumer Oriented Trusted Service Manager," in *13th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, Y. Liu, Ed. IEEE CS, September 2014.

[15] R. N. Akram, K. Markantonakis, and K. Mayes, "Coopetitive Architecture to Support a Dynamic and Scalable NFC based Mobile Services Architecture," in *2012 Inter. Conf. on Information and Communications Security*, K. Chow and L. C. Hui, Eds. China: Springer, Oct 2012.

[16] T. F. Bowen, F. Dworack, C.-H. Chow, N. Griffeth, G. E. Herman, and Y.-J. Lin, "The Feature Interaction Problem in Telecommunications Systems," in *Software Engineering for Telecommunication Switching Systems, Seventh International Conference on*. IET, 1989, pp. 59–62.

[17] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec, "Feature Interaction: A Critical Review and Considered Forecast," *Computer Networks*, vol. 41, no. 1, pp. 115–141, 2003.

[18] P. Zave, "Faq Sheet on Feature Interaction," *Link: http://www. research. att. com/~ pamela/faq. html*, 1999.

[19] S. Apel, W. Scholz, C. Lengauer, and C. Kastner, "Detecting Dependences and Interactions in Feature-Oriented Design," in *Software Reliability Engineering, IEEE 21st International Symposium on*, Nov 2010.

[20] K. Markantonakis, "The Case for a Secure Multi-Application Smart Card Operating System," in *the First International Workshop on Information Security*. London, UK: Springer, 1998, pp. 188–197.

[21] P. Girard, "Which Security Policy for Multiplication Smart Cards?" in *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*. Berkeley, CA, USA: USENIX Association, 1999, pp. 3–3.

[22] *GlobalPlatform: GlobalPlatform Card Specification, Version 2.2,*, GlobalPlatform Std., March 2006.

[23] "Multos: Guide to Loading and Deleting Applications," MAOSCO, Tech. Rep. MAO-DOC-TEC-008 v2.21, 2006.

[24] *Java Card Platform Specification*, Sun Microsystem Inc Std. V2.2.2, March 2006.

[25] R. N. Akram, K. Markantonakis, and K. Mayes, "Application Management Framework in User Centric Smart Card Ownership Model," in *The 10th International Workshop on Information Security Applications*, H. Y. YOUM and M. Yung, Eds., Korea: Springer, August 2009.

[26] ——, "A Secure and Trusted Channel Protocol for the User Centric Smart Card Ownership Model," in *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. Melbourne, Australia: IEEE CS, July 2013.

[27] K. Markantonakis, "Secure Logging Mechanisms for Smart Cards," Ph.D. dissertation, Royal Holloway, University of London, Egham, United Kingdom, December 1999.