

Politecnico di Milano

Polo Territoriale di Como

Faculty of Engineering Department of Electronics, Information and Bioengineering



Master of Science in Computer Engineering

# INTER APPLICATION COMMUNICATION IN MOBILE ENVIRONMENT

Supervisor:

Prof. Marco Brambilla

Andrea Mauri

Author:

Ognen Kostovski 814551

Academic year 2015/2016

## SOMMARIO

Nell'ambiente mobile, la possibilità di permettere alle applicazioni di collaborare fra di loro è stato, ed è ancora una caratteristica molto importante per noi. Il presente lavoro offrirà un modello di comunicazione app – to – app. Trasferire le informazioni da modelli di alto livello indipendenti in soluzioni specifici nella piattaforma appropriata. La costante crescita delle applicazioni software e i cambiamenti rapidi delle tecnologie sono una seria sfida per i metodi e gli strumenti tradizionali per lo sviluppo del software, i quali cominciano ad avere problemi per far fronte con i nuovi requisiti. Nello stesso tempo, tante compagnie affrontano difficoltà molto serie con i loro IT sistemi, in particolare nel modo in quale sono sviluppati, acquisiti e percepiti dagli utenti.

La moltitudine di piattaforme e di dispositivi ha creato problemi nel campo delle applicazioni mobili che sono molto diffuse, e globalmente per la software ingegneria, l'industria. Con il loro progresso una cosa diventa chiara o meno chiara quando si pensa a essa. Più piattaforme si producono, più applicazioni specifici si sviluppano, più il campo diventa fangoso. Uno di questi problemi è Inter App Communication (IaC) e nel presente saggio cercheremo di offrire soluzione di alto livello (basato su interazione con l'utente) che è guidato dal modello per lo sviluppo di modelli di modellazione riutilizzabili.

L'approccio dell'Ingegneria Guidata dal Modello - Model Driven Engineering (MDE)- proponendo una metodologia della piattaforma indipendente del design per quanto riguarda le dette difficoltà, richiede modelli migliorati e tecniche sofisticate di generazione codici. In questo studio con l'obiettivo centrato sui modelli, l'intenzione è introdurre nuove scheme di design aggiungendo concetti di Interazione con l'Utente e la logica software business: scheme di inter app comunicazioni connessi con l'interazione. La natura trasversale di artefatti ci permette di configurarli in modelli di design front-end, aumentando il loro potere espressivo e fornendo continuità con la logica back-end. Le piattaforme, per il problema della Comunicazione Inter App che noi cerchiamo, sono i sistemi operativi commerciali dai nostri dispositivi preferiti. Per il desktop: Windows, Macintosh, Unix e per i dispositivi mobile: Android, iOS, Windows 10, ecc.

La nostra Ricerca si basa su queste piattaforme e sul loro modo di risolvere il problema. Fino a poco tempo fa questo è stato ritenuto il lavoro dei sistemi operative, che agivano da broker per la comunicazione App – to – App. Ora che i fornitori delle piattaforme sono più generosi permettendo ai sviluppatori di gestirla senza la loro assistenza diretta (e il nostro approccio può fornire estensibilità), offriamo il presente saggio che tratta il detto problema con modello guidato, approccio indipendente dalla prospettiva dell'Utente.

La ricerca inizia con l'analisi della comunicazione dell'inter applicazione, con una valutazione dell'estensibilità e la generazione codici, e con scenari reali per dedurre le dinamiche che bloccano la comunicazione inter app, sottolineando i loro implicazioni riguardo l'interazione con gli utenti. Infine, presentiamo alcune scheme centrate sull'interazione derivate dai risultati dall'inchiesta effettuata. Successivamente, il filo fondamentale del nostro lavoro devia alla comunicazione inter app in applicazioni mobili, introducendo gli aspetti centrali, specializzazioni diversi e i loro criteri per adattamento. Gli artefatti che risultano da questa fase vengono mostrati in un modello spiegando il modellare delle applicazioni, seguito da spiegazioni dettagliate sulle scelte del design, l'impatto della forza espressiva del modello e i problemi affrontati o potenziali.

## *ABSTRACT*

In the mobile environment, the possibility of allowing applications to seamlessly collaborate with each other has been, and still is an important feature worth looking into. This work will provide a model driven approach to app – to – app communication. Transferring the knowledge from high level platform independent models to appropriate platform specific solutions.

The ever-increasing complexity of software applications and the rapid changes in the technologies are posing serious challenges to the traditional software development methods and tools, which are having problems to cope with the new requirements. At the same time, most companies are now facing serious difficulties with their IT systems, in particular in the way in which they are developed, acquired and perceived by users.

The multitude of platforms and devices has created problems for the widespread field of mobile application, and globally the software engineering, industry. As the advancements keep piling up, one thing is becoming clear--or less clear when you think about it. The more platforms are built, developers active and platform specific applications are developed, the more muddled the field becomes. One of these problems is Inter App Communication (IaC) and in this paper we are trying to provide model-driven high level (user interaction based) solution on development of reusable modeling patterns.

The Model Driven Engineering (MDE) approach, proposing a platform independent design methodology taking on these difficulties, requires enhanced models and sophisticated code generation techniques. In this model-centered study, we aim at introducing new design patterns joining concepts of User Interaction and software business logic: inter app communication patterns associated to interaction. The transverse nature of artifacts allows us to feature them in front-end design models, augmenting their expressive power and providing continuity with the back-end logic.

The platforms, for the Inter App Communication problem that we are looking into, are the commercial operating systems of our favorite Devices. For desktop: Windows, Macintosh, Unix and for the mobile devices: Android, iOS, Windows 10, etc.

We base our Research on these platforms and their way of solving the problem. Until a while ago this was deemed as the job of the operating system, who acted as a request broker for the App – to – App communication. Now that the platform vendors are more generous allowing developers to handle this without their direct assistance (and our approach can provide extensibility), we provide this paper that takes on this problem with a model-driven, platform independent approach from the User perspective.

The research starts by analyzing platform specific take on inter application communication, with a view for extensibility and code generation, and some real-world scenarios to infer the triggering dynamics of inter app communication, underscoring their implications on user interaction.

Eventually, we present some interaction-centered patterns derived from the result of this inquiry. Subsequently, the main thread of our work deviates to inter app communication in mobile applications, introducing its core aspects, different specializations and their adaptation criteria. The resulting artifacts of this stage are finally shown at work into an application modeling scenario, followed by a detailed explanation of design choices, impact on expressive power of the model and encountered or potential issues.

**Contents:**

CHAPTER 1	Introduction.....	1
CHAPTER 2	Background.....	4
CHAPTER 2.1	Model Driven Software Engineering (MDSE).....	4
CHAPTER 2.2	Interaction Flow Modeling Language (IFML).....	5
CHAPTER 2.3	IFML and Mobile Applications Extensions.....	6
CHAPTER 2.4	Design Patterns in Software Development.....	7
CHAPTER 3	Inter App Communication Technology.....	8
CHAPTER 3.1	Inter App Communication Mechanism.....	10
CHAPTER 3.2	Android.....	13
CHAPTER 3.3	Windows.....	18
Windows 8.1	App-to-App.....	18
Windows 10	UWP App-to-App.....	20
CHAPTER 3.4	Web of Apps.....	23
CHAPTER 3.5	Related Work.....	26
CHAPTER 3.6	Summary.....	27
CHAPTER 4	Inter application Communication.....	28
CHAPTER 4.1	Interactional Perspective.....	28
CHAPTER 4.2	Patterns Identification and Analysis.....	29
CHAPTER 4.3	Model driven design.....	30
CHAPTER 5	Implementation of app – to – app communication.....	43
CHAPTER 5.1	Web Ratio Mobile Custom Components.....	44
CHAPTER 5.2	Apache Cordova Plugin.....	45
CHAPTER 5.3	Web Ratio Mobile Application.....	48
CHAPTER 6	Future Improvements.....	50
CHAPTER 7	Conclusion.....	53
<b>Bibliography</b>	.....	<b>54</b>

# List of Figures

<b>Figure 1</b> Inter application Communication – proof of concept. . . . .	10
<b>Figure 2</b> Illustration of how an implicit intent is deliver through the system to start another activity. . . . .	16
<b>Figure 3</b> Launching target application based on chooser dialog. . . . .	18
<b>Figure 4</b> Sharing data with other applications based on chooser dialog. . . . .	19
<b>Figure 5</b> Windows 10 upgrades: launching specific app, for results and app service. . . . .	20
<b>Figure 6</b> Launching a specific target application . . . . .	21
<b>Figure 7</b> Launching a specific target application with a file token to be delivered . . . . .	21
<b>Figure 8</b> Launching a specific target application for results. . . . .	22
<b>Figure 9</b> Query the OS for laC support. . . . .	24
<b>Figure 10a</b> Query the OS to enumerate target applications. . . . .	25
<b>Figure 10b</b> Launching an application service. . . . .	25
<b>Figure 11</b> Launching target application using enumerate API. . . . .	30
<b>Figure 12</b> Enumerating applications able to handle the application call. . . . .	31
<b>Figure 13</b> Checking if the default value has been set for the application call. . . . .	31
<b>Figure 14</b> Save selected application as default. . . . .	31
<b>Figure 15</b> Launch specific application, web ratio custom component. . . . .	31
<b>Figure 16</b> Launching a specific target application without chooser dialog. . . . .	33
<b>Figure 17</b> Checking if the target application has been installed on the device . . . . .	34
<b>Figure 18</b> Launching specific application, web ratio custom component. . . . .	34
<b>Figure 19</b> Launching a specific target application for results . . . . .	36
<b>Figure 20</b> Launching specific application for results if it has been installed. . . . .	37
<b>Figure 21</b> Saving the input data from the target application. . . . .	37
<b>Figure 22</b> Target application able to handle launching calls from other Web Ratio apps. . . . .	39
<b>Figure 23</b> Updating the input data from the launching application . . . . .	40
<b>Figure 24</b> Return results and user control back to the launching application. . . . .	40
<b>Figure 25</b> laC using plugin xml file to update the application’s configuration files. . . . .	45
<b>Figure 26</b> JavaScript interface for Apache Cordova Plugin. . . . .	45
<b>Figure 27</b> laC Cordova Plugin Android implementation execute method. . . . .	45
<b>Figure 28</b> Alarm application IFML diagram. . . . .	47
<b>Figure 29</b> Send SMS App Service Web Ratio Custom Operation component. . . . .	48
<b>Figure 30</b> Direct SMS Messages application navigation WCO component. . . . .	48
<b>Figure 31</b> Navigation to chooser dialog presenting all messaging apps WCO. . . . .	48
<b>Figure 32 - 37</b> Alarm application Screenshots. . . . .	53

# List of Tables

<b>Table 1</b> Basic core IFML concepts: description, notation and platform-specific cases . . . . 5
<b>Table 2</b> Inter application Communication - Protocol . . . . . 11
<b>Table 3</b> App – to – App communication use cases. . . . . 26
<b>Table 4</b> Android implementation of the Inter app Communication protocol. . . . . 46
<b>Table 5</b> Application Services use case scenarios. . . . . 51

# CHAPTER 1 INTRODUCTION

Today there are so many mobile devices running on different platforms and platform versions, with even bigger number of different displays (differing in screen size, aspect ratio, PPI, resolution and various technologies).

The reality is most of the phones are smartphones, and with that they become less of phones, and more of cameras or navigation devices. This extraordinary change in the way people communicate and access information is happening all the time. Each day mobile devices are used more extensively. It seems like everything is evolving so fast, that is hard to keep up with the ever-changing mobile industry. But if we just take a small step back and try to see the big picture, the only important thing we can see is information. Information is the one thing that matters, because people need to communicate. In order to live, survive and prosper, people must exchange information. The only thing that changes with time is the medium through which people communicate. Therefore we need to bring consistent information and allow seamless way of communication throughout all modern devices.

The mobile ecosystem is undergoing an outstanding expansion, affecting not only existing industry segments but planting the seeds for new realities. Enabled by the overcoming of constraints related to computational power, reduced memory and storage of mobile devices, the enterprises exploring the mobile world are challenging the idea of ameliorating users' lives exploiting the possibilities provided by the pervasiveness of mobile devices. New disciplines and trends are rapidly growing on top of this idea, intersecting multiple areas of interests: many leading hardware and software companies are extending, reviewing their research scope to deepen their knowledge and provide better integrated solutions for users in mobility. This enthusiastic impulse has been introducing a huge amount of novelties reshaping the existing ecosystem: let us mention, for instance, the spread of wearable devices and the vision of the Internet of Everything.

On the other hand, the general demand over software quality and complexity has been increased as well. Today applications designed to run on mobile devices feature rich interfaces implying convoluted interaction mechanisms and impressive business logic implementations, allowed by the technological evolution of the target devices. Additionally, thinking to all of the latest trends characterizing the mobile software environment, like Connected Living, Advertising, Digital Commerce and Security, the relevance of data-intensive services is undeniable. Critical tasks like Inter app Communication (IaC), data synchronization, contextual information retrieval and the activation of communication paths to access external service providers are difficult to orchestrate, and having to consider them at design time is not a trivial software engineering problem.

Keeping app's user interface (UI) and user experience (UX) consistent in a cross-platform environment and throughout different displays out there represents a huge challenge. Fundamentally, we have two words that do not quite fit together: diversity and consistency. The bigger the diversity, the harder to keep the consistency. And what matters the most while keeping this cross-platform consistency are time and cost. How long it will take, and how much it will cost? Every application developer (whether an individual or a company) out there tries to keep these two values as low as possible.

In this problematic context, computing with functional and non-functional requirements of mobile applications serving multiple platforms on different devices results in the emergence of the need of systematic approach. Matter of fact, in the typical front-end development of mobile applications manual coding still remains a predominant practice, with all the liabilities it involves: low reuse, inefficient maintainability, problems of portability, risks of inconsistencies are just some of them.

Approaches like Model Driven Engineering (MDE), conversely, leverage the introduction of the abstraction relying on their core discipline, software modeling. However, using a model driven approach to produce quality software introduces controversies related to the sophistication level of models and the complexity of their implementation through code generation techniques (exploited to produce actual code).

Developers need tools that will help them build applications faster while keeping them consistent throughout different devices. The gap that existed for complete automation of the process of developing cross platform applications has been bridged by using the Interaction Flow Modeling Language (IFML). A front-end modeling language used to create models for the system to be developed. It provides a conceptual modeling consisting of formally defined graphical and textual representation. With the introduction of IFML, we are moving into the field of Model-Driven Development (MDD), where MDD applications are automatically generated from the models, allowing for more flexibility, faster prototyping, validation in the early phases of a project and shorter time to market.

Having in mind, developers need tools that will help them build apps faster while keeping the apps consistent through different devices. Fortunately, there are many innovating tools that are focused on designing cross-platform mobile development patterns and principles. Based on the comparisons of these tools we have chosen to make implementation for Phonegap cross-platform mobile development framework using tool called Web Ratio (Mobile version). It is a tool based on model-driven engineering of data-intensive applications. To achieve this it uses the IFML modeling language, provides possibilities to extend it with UML and BPMN diagrams and maps the designed diagrams of the system into cross-platform mobile development concepts. This allows the developers to generate fully automated Apache Cordova applications from the system designed models.



In this thesis, we are focusing on the front-end modeling perspective, trying to introduce some novel artifacts to cope with the requirements over data. In particular, Inter app Communication is a recurring process, abstracting a huge amount of scenarios that must be covered by applications relying on remote or local data access and processing. The approach we adopt is purposely pattern-based, given the correlation between the problem-solving nature of Inter app Communication and the problem-solution structure of design patterns. Since patterns proposal requires an accurate understanding of the dynamics to represent, we move towards app to app communication with a top-down analysis of the state of the art, resulting from the study based on understanding the way platform specific technologies tackle the IaC problem and gaining global understanding from high level user interaction point of view. The solution that our patterns are meant to provide combines aspects of user interaction with the business logic implied by inter app communication. Given the heterogeneous nature of these artifacts, they are expressed using two different languages: Interaction Flow Modeling Language (IFML), representing front-end concepts, and Unified Modeling Language (UML), modeling the behavior of the application components. Both IFML and UML have been adopted as standards by the Object Management Group (OMG). The most valuable goal we are pursuing with this experimental work is to demonstrate the effectiveness of our introductions in terms of expressive power of front-end models.

The thesis is structured as follows:

**CHAPTER 2. *Background***

Informs the reader of the key concepts and their use.

**CHAPTER 3. *Inter application Communication Technology***

Top-down Research conducted through analysis of the two mechanisms for IaC.

**CHAPTER 4. *Inter application Communication Models***

Model driven solution for the IaC problem based on conclusions from the Research.

**CHAPTER 5. *Implementation of App - to - App communication***

Implementation of the Model driven solution in a real world Application.

**CHAPTER 6. *Future Improvements***

Possible deviations of this work that will improve the idea of Web of Apps.

**CHAPTER 7. *Conclusion***

Asks and answers the very basic questions that motivate this work.

## CHAPTER 2 BACKGROUND

Trying to solve the app – to – app communication problem with a model driven approach by introducing software design patterns for the mobile environment. This is accomplished using the IFML modeling diagrams and carefully and clearly selecting the problem at hand, later presented as pattern solutions. These and several other key concepts will be introduced to understand this platform independent approach for solving IaC problem.

### CHAPTER 2.1 MODEL DRIVEN SOFTWARE ENGINEERING (MDSE)

Besides the notion of pattern, the other pillar of our study is representation by the scope in which the exploitation of patterns takes place: the modeling environment.

The MDE approach. Model Driven Engineering is an approach to development using models as main artifacts. This translates to the exploitation of models during the whole process of development, in contrast with a normal approach that typically takes advantage of their features only in the analysis and design phases.

The relevance of MDE in software engineering [1] has known a notable growth in the past five years thanks to rising forces like application pervasiveness and their multi-platform, distributed nature. The adaptation of an MDE approach eludes the weaknesses of a trial-and-error typical of adaptively fixed systems, fostering the focus change on gathering abstract representations of the knowledge governing application domains and their re-use.

On the other hand, code generation from software models is a hard task, since it requires a complete understanding of the model to code transformation semantics. As a consequence, end-to-end generation of application code matching the level of quality of a hand-crafted solution remains an ambitious goal. Despite these difficulties, some impressive outcomes have been achieved, like in the case of Web Ratio, company in the MDE tool market as reported in [2]

The central role of models in MDE products and services made the industry agree on the idea of setting some standards to produce platform-independent models. As a consequence, the OMG (Object Management Group) adopted the Interaction Flow Modeling Language (IFML) as a standard in July 2014.

## CHAPTER 2.2

## INTERACTION FLOW MODELING LANGUAGE (IFML)

As we have been stating so far, we want to express the patterns we are going to present in the most abstract way possible, to preserve the platform-independence of our study. Under this perspective, the Interaction Flow Modeling Language (IFML) appears to be an excellent technology, being a modeling language that “supports the specification of the front-end of applications, independently of the technological details of their realization” [3]. In fact, IFML addresses several questions related to front-end modeling, such as view composition and content, commands, actions, effects of interaction and parameter binding. IFML syntax is visual, based on OMG standards and very close to the one characterizing Unified Modeling Language (UML), thus it results familiar to developers.


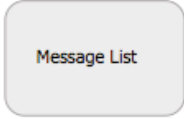

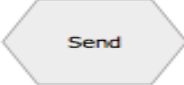


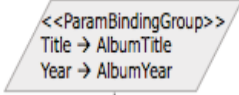
Concept	Meaning	IFML Notation	PSM Example
<b>View Container</b>	An element of the interface that comprises elements displaying content and supporting interaction and/or other View Containers		Web page, Window, Pane
<b>View Component</b>	An element of the interface that displays content or accepts input		An HTML list, A JavaScript image gallery, An input form
<b>Event</b>	An occurrence that affects the state of the application		
<b>Action</b>	A piece of business logic triggered by an event		A database update, The sending of an email
<b>Navigation Flow</b>	An input-output dependency. The source of the link has some output that is associated with the input of the target of the link		Sending and receiving of parameters in the HTTP request
<b>Data Flow</b>	Data passing between View Components or Action as consequence of a previous user interaction		
<b>Parameter Binding Group</b>	Set of Parameter Bindings associated to an Interaction Flow (being it navigation or data flow)		

Table 1 Basic core IFML concepts: description, notation and platform-specific cases

Its expressive power is empowered by extensibility, supported by the language thanks to its incorporated standard means for defining new concepts. Practically, this quality is being exploited to propose a model-driven approach for Mobile Applications development, as portrayed in [3] and witnessed by innovative commercial solutions like Web Ratio Mobile Platform [4]. Considering all of the abovementioned advantages, the use of IFML to satisfy our inquiry is a logical consequence. In the Inter application Communication (IaC) chapter, we are introducing the patterns depicting the front-end implications of app-to-app communication triggering using the IFML notation, fully referenced in [3].

## CHAPTER 2.3

## IFML AND MOBILE APPLICATIONS EXTENSIONS

Let us now explain how IFML contributes to create suitable platform-independent models (PIM) of graphical user interfaces for application deployed on multiple systems. As explained in the details in [3], an IFML PIM provides:

- The specifications of the view composition and content, respectively illustrating the schema of containers and their components
- The specifications of events (commands) affecting the state of the User Interface and of the transitions (effects of the interaction) portraying the effects implied by the state change
- The parameter binding specification, listing the dependencies between containers, components and actions
- The actions reference, specified by interaction flows connecting events to affected view containers or components

The PIM should be designed for change and its design should adhere to a set of “golden rules” of the language:

- It should be concise, conveying the front-end model in a single diagram and avoiding redundancy by exploiting inference from the model
- It should encourage extensibility, allowing the adaptation to novel requirements, interaction modalities
- It should be implementable, to ensure that models can be mapped easily into executable applications
- Not everything should be in the model, meaning that, for instance, presentation aspects should not be covered

In particular, the extensibility property of the language is what we relied on to conduct our research work. In fact, IFML provides tools for defining novel concepts and interaction paradigms, possibly substantially different from the ones for which the language has been designed. Under this perspective, our study demonstrated this concept by integrating in models information on Inter app Communication, and aspect that is based on front-end design, but in some cases is implemented with back-end logic.

To conclude this brief overview of the features of IFML that enabled this work, we mention some extensions – widely adopted for the representation of the patterns presented in the next chapters – of the language conceived for the design of mobile applications.

The class Screen, representing a screen of the application, has been defined by extending View Container, as well as Tool Bar, which may contain other containers and have on its boundary a set of events. The Mobile System stereotype has been introduced to distinguish View Containers that in mobile interfaces are devoted to specific functionalities (like Notifications, Settings). Mobile Components extends the classic IFML class View Components and denotes mobile view components, such as buttons or icons.

## CHAPTER 2.4

## DESIGN PATTERNS IN SOFTWARE DEVELOPMENT

The concept of “Design Pattern” has been playing a central role in the study presented in this research work. Despite of this fact, so far we have never deepened the concept of pattern within the environment for which it was intended, that is software design.

In simple words, a pattern highlights a problem occurring multiple times in a given environment and describes the solution to that problem, in such a way that the solution can be re-used over and over. More specifically, in software industry a design pattern is described by a written formal document detailing the elements characterizing it. According to [13], authored by the so called “Gang of Four”, which in this discipline is a fundamental reference document, the headings of the specification must include Intent, Motivation, Applicability, Structure, Participants, Collaborations, Consequences, Implementation, Sample Code, Known Uses and Related Patterns. Eventually, as design patterns have been growing in popularity, alternative pattern forms have been defined: the most common are Portland, Coplien (adopted to introduce patterns in this thesis), Posa and P of EAA.

In the upcoming paragraphs (Naming, Force, Issues) we are reporting some remarks on patterns creation, taking inspiration from the considerations articulated by the author on enterprise software Martin Fowler in [5].

**Naming.** A great benefit resulting by the identification of patterns is given by the possibility to organize, name and catalog the problem-solution scenarios, so that their reusing potential is facilitated. For this reason, choosing an effective, evocative name is a valuable task, keeping in mind that it should be part of a vocabulary of software techniques.

**Forces.** A common misunderstanding about patterns results from thinking that they should introduce something new in software engineering. This is not true at all, as their goal is to capture knowledge, rather than invent it. Precisely because of this, when presenting a pattern, writers should strongly motivate their need, emphasizing the forces behind them. Under this perspective, trying to think about occasions in which the pattern usage would not be recommended can be useful, at least to identify an alternative pattern.

**Issues.** Granularity expresses the surface of conceptual ground covered by a pattern. Typically, selecting granularity opens a debate not allowing absolute solutions: in fact, deciding where to place boundaries between different patterns operating on nuances of the same problem is a hard task. Another critical aspect is the task-oriented nature of patterns. Tool orientations in software leads to simplifications, and pattern writing does not represent an exception: observing a framework and identifying tools is intuitive. But patterns should stay task-oriented, because of the natural answer that task orientation provides to designers adopting the pattern.

**The Behavioral Approach.** By definition, “behavioral patterns are concerned with algorithms and the assignment of responsibilities between objects” [13, p. 249]. They focus not just on objects and classes, zooming also on the existing communication mechanisms between them. This kind of approach broadly resembles the one used to create the patterns illustrated in our research: even though it may appear tough to visualize through the IFML renditions, this similarity is emphasized by the UML sequence diagrams... (primer) Processes of this nature are similar to the scenarios portrayed by the “Chain of Responsibility” classic pattern [13, p. 251], in which requests are forwarded and handled with flexibility, and objects collaborating are loosely coupled.

## CHAPTER 3 INTER APP COMMUNICATION TECHNOLOGY

When it's said that two applications communicate with each other in the IaC context, that stands for the interaction that happens between the last screen of one and appropriate starting screen of the other application. This dynamic is highly dependent on the platform constraints in the mobile environment.

Looking at Platform Specific solutions with a view on extensibility for our Model Driven Approach and possible code generation to specific model use cases that we will present.

In order to provide platform indifferent solution we need first to analyze and then to adapt to the knowledge and constraints imposed by the platforms. This is not easy and there are methods in order to provide a well-defined solution.

That is why, in the following chapters we will introduce the concept of IaC based on an example which will deliberately pinpoint the difference between the platform specific solutions. Detailed look into the most significant factors that represent the common features relative to the problem of IaC. The result of this research will not only help bring us closer to platform indifference, but the conclusions based on it will be the introduction and bases of Chapter 4, the model driven approach to Inter app Communication. First lets try to understand what IaC really means. Inter app Communication (IaC), also stands for App – to – App communication. Or in other words, the different ways that two applications are able manipulate user experience, with or without exchange of information between one another. These dynamics of the UX are leading the project to common features and platform independent IaC.

Platform independence is achieved through common knowledge representation using model driven approach and bridging the gap to targeted platforms using appropriate implementation techniques. That is because the common features are presented in an abstract approach. Transforming that to platform specific solution, even with quality code generation, depends on specific native implementations. After carefully evaluating the research and taking into account the results from it, the decisions on which platform has been chosen as a reference point from the gathered knowledge of the research will be discussed and evaluated.

This represents top-down approach to analyzing Inter application Communication based on the dynamics of existing platform technologies and building a high level understanding applicable to PIM models.

Two mechanisms encountered: Android Intent system and URL Custom Schemas (all other platforms). Basic example is the best way to highlight the **limitations** and **constraints** implied by these mechanisms of IaC. These differences in philosophy, security and user control over the process that are imposed by the commercial platforms are the exceptions that need to be strategically analyzed and resolved with clarity. Because most of them will be part of the platform specific adaptation, which is essential for the idea of platform indifference. But Intent and URL mechanisms, the way they work and the way they are used in a platform specific way, also hold the knowledge for the protocols required in order to provide our conclusions. These conclusions are the result of this top-down research which will guide us to build PIM models that capture these dynamics.

The IaC problem has been evolving in different ways and with different implied constraints by each platform, so it was easy to get lost in this uncertainty until now when platforms are more open to allowing developers orchestrate it by themselves. This freedom is allowing developers to have more control over inter application user experience which until now has been deemed the job of the operating system. Although still constrained, and whether or not it will remain like that in the near future, it is worth pointing out that these mechanisms have been here for a long time.

Interesting to note is that the Android Intent system uses the same type of messages for both possible types of communication, internal and external. On the other hand, iOS uses different messages types for each of these communication types. Windows Phone 7 at the beginning was using only a small number of messages to provide security and stability to their application.

## CHAPTER 3.1

## INTER APP COMMUNICATION MECHANISM

Comparison of **two mechanisms**. Through this basic example we will try to highlight their **limitations** and **constraints**. The first mechanism we will be looking into is Android Intent System (system characteristics are defined in chapter 3.2). Custom URL Schemas are a well-known concept from the time of the first version of Internet Explorer and represents the second mechanism for IaC. In the following example we will compare these two mechanisms, but in order to fully understand the possibilities of this mechanism, in the research we will explain the characteristics of the Windows platform which also operates on Custom URL schemas (chapter 3.3).

### **Android Intent System** and **Custom URL Schema** (iOS, Windows).

To provide a proof of concept for Inter-App Communication protocol, we will introduce a simple interaction scenario, seen in figure 1. Two applications will be needed to do the tests. On one hand we will have the Gate-keeper App developed for the Android platform. On the other hand, we will have a demo application generated with Web Ratio which uses a set of Services like WCOs (Web Ratio Custom Operations).

Since we initially want to demonstrate the technical feasibility of this feature, we will start with a rather simple use case. In it, the user will start by opening a demo application generated with Web Ratio in which he will be able to navigate through the places and events contained in the KB (Knowledge Base). Every time the user taps on a snippet that contains a summary description of one of the items in the KB, he will be redirected to the Gate-keeper app where he will be able to continue his exploration, by leveraging the enhanced search capabilities and other functionalities offered by the application.

Under the hood, the scenario will exercise one of the app to app interactions documented in the Inter App Communication Protocol. Particularly, the OPEN\_EVENT message will be used to enable the seamless transition of the user from a Web Ratio generated app to the Android version of the Gate-keeper application.

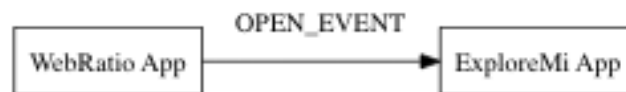


Figure 1 Inter app communication - proof of concept

The **scope** of the proof of concept.

The interaction scenario will be established between an application generated with Web Ratio and the Target application.

Due to some technical constraints imposed by Web Ratio, the communication to be implemented will flow in one direction and will start in the Web Ratio application.

Since the Android version of the Target app will be the first one to implement the inter-app communication protocol, the first version of the proof of concept will only work in this platform. Future developments may target iOS, keeping in mind the **limitations** imposed by the platform and which we will try to understand during this research, to provide better IaC dynamics.



## Inter-App Communication Protocol

One of the dimensions of growth of a platform depends on the number of mobile applications that rely on its Knowledge base to offer value to their users. In this scenario, having a standardized way in which these applications can communicate with each other becomes an important concern.

Supporting IaC type of interactions in a mobile context is equal to defining a set of messages and responses that can be sent back and forth between applications. We call this collection of messages the Inter App Communication protocol, and we present in the following table some messages that are deemed interesting to compare between the mechanisms.

Description	Android Intent	iOS URL Scheme
Open Screen	OPEN_SCREEN	screen/open?id=<screen-id>
View other application	ACTION_VIEW	screen/open
Obtain Access Token	OAUTH	oauth?key=<app-key>
Send message	ACTION_SEND	tel/message=<msg-id>
	<b>(optional) abstract-name:</b>	<b>URL-Schema:</b>
	iac.polimi.alarm	alarm://www.alarm.com/
	General or Specific Intent	Always known Target app

Table 2 Inter app communication - Protocol

Mobile platforms offer different and uneven support for inter-app communication. On the one hand, Android offers a very robust system based on Intents that allows for almost any kind of inter-app communication to occur. On the other hand, iOS offers a more restrictive model based on URL Schemes, which mostly allow for simple communications that follow only in one direction

### Cross-Platform Considerations

The idea could work in Android. In this platform, developers can declare that their applications are able to respond to specific intents. In the case when the Web Ratio app could have a button that triggers the publication of an intent (ACTION\_SEND). The OS will show the user a list with all the applications that can handle the ACTION\_SEND intent. In our example, a ticketing application that has registered its ability to respond to the ACTION\_SEND intent will appear in the list.

In the case of iOS, the use case is not easy to replicate. In this platform, developers express the messages that their application is able to receive. The format of these messages should be unique. In the case of the ticketing application, the developer will register his own scheme, and will specify the messages that other applications can send to it – e.g. www.ticketing.com/event/:id. The problem with this implementation, is that to invoke the ticketing application, the Web Ratio App or any gate-keeper Target application will have to know before hand, the specific messages that it wants to broadcast. That is, to send a message to the ticketing app it will have to know the www.ticketing.com/event/:id URL.

In summary, the use case that had been proposed can only be implemented in Android. Whereas in iOS, the protocol can only be supported by the official apps.

## **System level messaging**

Only a subset of general Intents / Actions can be defined. This is because, we cannot predict what will be the intents the other applications will be able to manage. This means, that Target app, will be able to publish a particular set of, somewhat general, Intents – ACTION\_VIEW, so that the Operating System can show the user all the applications installed that are able to respond to the given event.

This strategy will require application developers to familiarize themselves with the Intents that are published by the official apps, so that their applications can respond to them.

## **Opportunities**

While the communications that flow from the Web Ratio apps to third-party applications, need to be somewhat generic, and in the case of iOS, they are rather hard to implement, communications that flow in the opposite direction offer a larger set of possibilities. From this perspective, the Web Ratio apps “publish” a list of messages / Intents that is able to manage. This will allow third-party applications to request the execution of specific actions from the Web Ratio app. Coming back to the example of the ticketing application, in this case the developer could issue an intent –ACTION\_SEND, to request the Web Ratio app to add a new event to the user’s wish-list.

Using this communication flow the Web Ratio app becomes a sort of bridge that makes it even easier for developers to integrate their apps with the platform. That is, instead of having to implement the communications with the servers to add an event to the user’s wish-list, a developer could simply issue an intent, so that the Target app takes care of this operation.

## CHAPTER 3.2

## ANDROID

The Android Intent System [6] is similar to Shell execute on Windows for the principles involved, which takes care of inter process communication. It can be viewed as a black box that behind the scenes works with naming conventions similar to the custom URL schemas. But with advanced protocols it manages to provide robust Inter app Communication. Also the way that applications communicate is the same way, no matter whether they want to launch a screen from the same or from another applications, only the protocols are adapted. In this chapter we will analyze the messages of this protocol, called intents, and the dynamics of the way they are used which is the protocol itself.

Sending an Intent has been possible for quite a long time in Android. At first, the only way possible to react to an intent call was by handling it through a push notification. Notifications and the status bar of the phone are still an integral part of today's smartphone experience.

Receiving an Intent, via Push Notification:

Receive a notification in the status bar can alert the user that some background action has been completed. If the user is interested in this announcement he can interact with the status bar and the notifications in it. These user interactions can be handled and some are used to launch the Application from the intent information stored inside the notification.

But after a while it was realized that it doesn't have to be a Push Notification, and variants of this system were developed for its adaptation to better the inter and intra app communication between the screens provided in an application.

### *Introduction to Intents*

Intent is a messaging object you can use to request an Action from another Application, and there are two type of intents:

- Explicit Intent – sent to a Specific Application, Service, Activity. Specify the Component to start by a fully qualified class name.
- Implicit Intent – broadcast to All Applications with matching Intent Filter in their manifest. Declares a general action to perform.

An intent is an abstract description of an operation to be performed. It can be used with start activity (screen), to launch an activity, broadcast intent to send it to any interested broadcast receiver components, and start service(Intent) or bind service(Intent, Service Connection) to communicate with background Service.

An Intent provides a facility for performing late runtime binding between the code in different applications. Its most significant use is in launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed.

The primary pieces of information in an intent are:

- action – The general action to be performed, such as ACTION\_VIEW, ACTION\_EDIT, ACTION\_MAIN
- data – The data to operate on, such as a person record in the contacts database, expressed as a Uri.

In addition to these primary attributes, there are a number of secondary attributes that you can also include with an intent:

- category – Gives additional information about the action to execute. For example, CATEGORY\_LAUNCHER means it should appear in the Launcher as a top-level application, while CATEGORY\_ALTERNATIVE means it should be included in a list of alternative actions the user can perform on a piece of data.
- type – Specifies an explicit type (a MIME type) of the intent data. Normally the type is inferred from the data itself. By setting this attribute, you disable that evaluation and force an explicit type.
- component – Specifies an explicit name of a component class to use for the intent. Normally this is determined by looking at the other information in the intent (the action, data/type, and categories) and matching that with a component that can handle it. If this attribute is set then none of the evaluation is performed, and this component is used exactly as is. By specifying this attribute, all of the other Intent attributes become optional.
- extras – This is a Bundle of any additional information. This can be used to provide extended information to the component. For example, if we have an action to send an e-mail message, we could also include extra pieces of data here to supply a subject, body, etc.

There are a variety of standard Intent action and category constraints defined in the Intent class, but applications can also define their own. These strings use Java-style scoping to ensure they are unique – for example, the standard ACTION\_VIEW is called “android.intent.action.VIEW”.

Put together, the set of actions, data types, categories, and extra data defines a language for the system allowing for the expression of phrases such as “call john smith’s cell”. As applications are added to the system, they can extend this language by adding new actions, types and categories, or they can modify the behavior of existing phrases by supplying their own activities that handle them.

Three primary uses:

An Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are **three fundamental use-cases**:

**To start an activity:**

An Activity represents a single screen in an application. You can start a new instance of an Activity by passing an Intent to start Activity method. The Intent describes the activity to start and carries any necessary data.

If you want to receive a result from the activity when it finishes, call start Activity for result method. Your activity receives the result as a separate Intent object in your activity's on Activity Result handling method callback.

**To start a service:**

A Service is a component that performs operations in the background without a user interface. You can start a service to perform a one-time operation (such as download a file) by passing an Intent to start Service method. The Intent describes the service to start and carries any necessary data.

If the service is designed with a client-server interface, you can bind to the service from another component by passing an Intent to bind service method.

**To deliver a broadcast:**

A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an Intent to send broadcast method, send ordered broadcast method, or send sticky broadcast method.

These use cases trigger the start of new application, but how does the target application register these calls? The intent messaging object is the source information that we want to send, while the target application registers the possible resolutions of intents in manifest xml file. There are two types of intents:

**Explicit intents** specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, start a new activity in response to a user action or start a service to download a file in the background.

**Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

When you create an explicit intent to start an activity or service, the system immediately starts the app component specified in the Intent object.

When you create an implicit intent, the Android system finds the appropriate component to start by comparing the contents of the intent to the intent filters declared in the manifest file of other apps on the device. If the intent matches an intent filter, the system starts that component and delivers it the Intent object. If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use. Illustrated in figure 2.

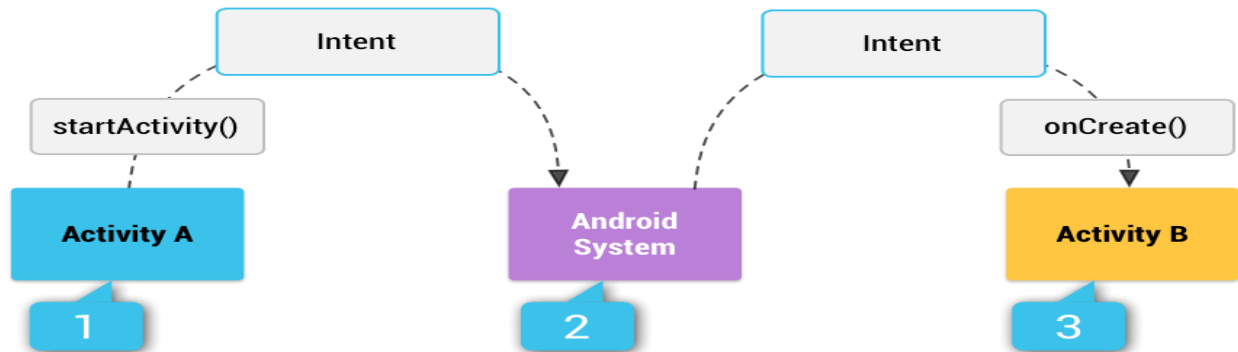


Figure 2 Illustration of how an implicit intent is delivered through the system to start another activity

An intent filter is an expression in an app's manifest file that specifies the type of intents that the component would like to receive. For instance, by declaring an intent filter for an activity, you make it possible for other apps to directly start your activity with a certain kind of intent. Likewise, if you do not declare any intent filters for an activity, then it can be started only with an explicit intent.

As a step of caution. To be secure, always use an explicit intent when starting a Service and do not declare intent filters for your services. Using an implicit intent to start a service is security hazard because you cannot be certain what service will respond to the intent, and the user cannot see which service starts. (bind Service with implicit intents throws exception!)

### Intent Resolution

There are two primary forms of intents to be resolved:

- Resolution of explicit Intents, have specified a component which provides the **exact class to be run**. Often these will not include any other information, simply being a way for an application to launch various internal activities it has as the user interacts with the application.
- Resolution of implicit Intents, have not specified a component, instead, they must include enough information for the **system to determine** which of the available components is best to run for that intent.

When using implicit intents, given such an arbitrary intent we need to know what to do with it. This is handled by the process of **Intent Resolution** [7], which maps an Intent to an Activity, Broadcast Receiver or Service (or sometimes two or more activities/receivers) that can handle it.

The intent resolution mechanism basically revolves around matching an Intent against all of the <intent-filter> descriptions in the installed application packages. (Plus, in the case of broadcast, any Broadcast Receiver object explicitly registered with register Receiver(Broadcast Receiver, Intent Filter)). Details on this can be found in the documentation on the Intent Filter class.

There are three pieces of information in the Intent that are used for resolution: the action, type and category. Using this information, a query is done on the Package Manager for a component that can handle the intent. The appropriate component is determined based on the intent information supplied in the AndroidManifest.xml file as follows:

The **action**, if given, must be listed by the component as one it handles.

The **type** is retrieved from the Intent's data, if not already supplied in the Intent. Like the action, if a type is included in the intent (either explicitly or implicitly in its data), then this must be listed by the component as one it handles.

For data that is not a content: **URI** and where no explicit type is included in the intent, instead the **scheme** of the intent data (such as http: or mailto:) is considered. Again like the action, if we are matching a scheme it must be listed by the components as one it can handle.

The **categories**, if supplied, must *all* be listed by the activity as categories it handles. This is, if you include the categories CATEGORY\_LAUNCHER and CATEGORY\_ALTERNATIVE, then you will only resolve to components with an intent that lists *both* of those categories. Activities will very often need to support the CATEGORY\_DEFAULT so that they can be found by Context.startActivity

## CHAPTER 3.3

## WINDOWS

Inter App Communication enables the developers application to communicate with the built-in (native) Applications or other Developer Applications.

As a representative of the Custom URL Schema mechanism, used as protocol for providing inter application communication, we have chosen the Windows mobile platform. In order to build high level understanding of the common features for platform indifference it is best that we base the research on knowledge and information that gathers the most about the mechanism of interest. Windows have used and provided a lot of different ways to use the Custom URL Schemas. That clearly states that some other platforms like iOS can and maybe will release these extended range of use cases in the platform. But until now in the iOS platform most of the advanced features are not allowed because of application security and simplicity reasons.

We will first start with Windows 8.1 and then Windows 10 to address the possibilities of Custom Url Schemas mechanism [8] and how it evolves.

### WINDOWS 8.1 APP-TO-APP

#### Uri and Protocol Activation

```
Launcher.LaunchUriAsync(new Uri("sampleapp:?ID=aea6"));
```

```
Launcher.LaunchFileAsync(file);
```



Figure 3 Launching target application based on chooser dialog

Ability to call the application based on a Custom Protocol that is set up inside the developers Application. The protocol is registered whenever the starting Application is installed. Calling the starting application Protocol or in other words will be calling the Application is seen in the first line of figure 3. Similar to regular Uri, possibility to add variables / Value Sets. Limitation: Not sure whose Application I am calling, if other user installed same Application – by design made like this. User/OS chooses the Target Application, which makes a point that it was meant to be Decoupled (with the Chooser Dialog).

Register the Application to have a File extension (Adobe -> “.pdf”) is the second line of figure 3. The way that the developers Application registers to Handle specific Application calls is through the Application Manifest. If your application is using files, there you can register that your application is able to respond to call from a file invocation with a certain extension. This is usually more common with Desktop platforms.



## Share Contract

Another way that Applications in Windows 8.1 can “share” stuff is Share Contract seen in figure 4.



Figure 4 Sharing data with other applications based on chooser dialog

The sending (sharing) Application creates a Data Package of stuff (files, text, images, ...) that is able to share. This is the safest way that two Windows applications can communicate with each other. The principles are strict and the opportunities are limited. But unlike the previous approaches, here different applications communicate with each other without third-party user interaction interference. This makes them independent applications that can use limited information between them.

The Shell (OS) puts up a picker User Interface, so the user chooses the Applications that he wants to share it with. After choosing these applications, the referred data packages will be mutually available.

Share Contracts is a decoupled way of connecting the sender and the receiver. It solves the problem of two Applications that can't talk to one-another. But of main interest are APIs that allow transferring user control from one application to another without third-party interferences. This was, and still is the philosophy and security opinion of the iOS platform. Where the previous rule of Windows 8 still stands today, which is that two applications cannot talk directly between each other. Although this rule doesn't stand for the platform applications coming with the operating system itself.

The reason for that, is that you don't want to make another Application unstable, so the operating system (Windows) works as a broker between them. Application passes its payload to the broker, and after the right protocols, the broker pass payload to the Target Application.

Sharing Data is too accomplished with choosing the collaborating applications, only this is done before the communication even started. Or in other words sharing through a share Broker.

## WINDOWS 10 UWP APP-TO-APP

Figure 5 shows all the enhancements in App-to-App communication on Windows 10:

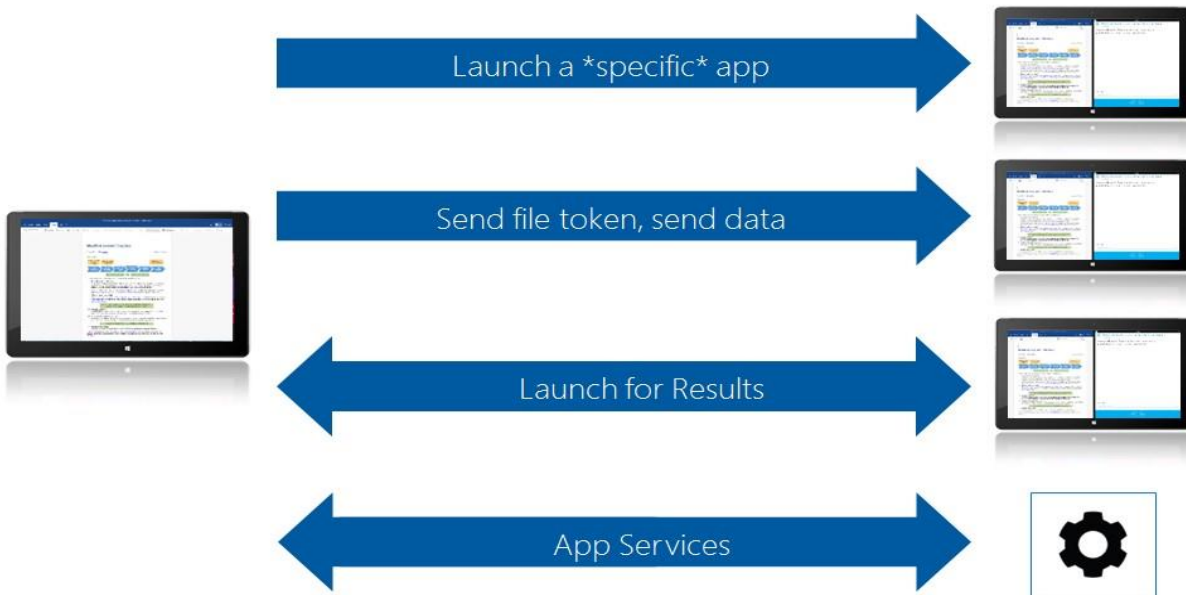


Figure 5 Windows 10 upgrades: launching specific, for results and app service on the target application

The improvements in the IaC have been impressive between the last versions of Windows, which will be discussed throughout this chapter. The new app-to-app APIs come as a solution to the direct communication without the involvement of a third-party application when two applications communicate on the Windows platform.

Although these enhancements have a big effect on the limitations imposed by the platforms using Custom Url Schemas for the IaC problem, it still doesn't solve the biggest difference between the two mechanisms. Which is that the android package name is optional, while the Custom Url Schema always needs to be known before accessing target application.

In the case of the specific application method call. The user doesn't choose the application, which represents a strongly coupled way of launching target application. Mechanism for transferring control one app to another previously commercially unknown.

Sending a file or an access token. Represents a way of accessing another application, receiving data result previously sent through a file token. This is done through the use of dictionary that allows developers to pass a key (instead of string) that represents access to a file.

Launching a target application for results can only be achieved by using direct application communication. Having third-party screens in the navigation can lead to mistakes in the navigation (and more importantly the respective data). That is why this is two way direct communication, where the target application processes the received data and returns result.

Background tasks that can be invoked by anyone, anywhere (like Web Services but on Device). They represent instant on call way of sharing data between applications without bringing up their UI.

## Uri activation to a *specific* App

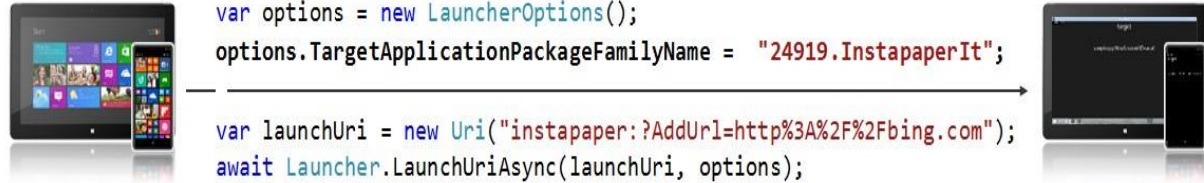


Figure 6 Launching a specific target application

Added to the Launcher Options object with property “Target Application Package Name”. This clearly defines the package and custom URL address. That is how it is possible to access the user interface directly of another target application. In figure 6 we see the options object as a parameter for the API. Where the package family name is a unique name by Windows store. Through this name the OS can identify the unique Target Application, which eliminates the need for the user to choose it.

## Send files – Sending file tokens, Sending Data

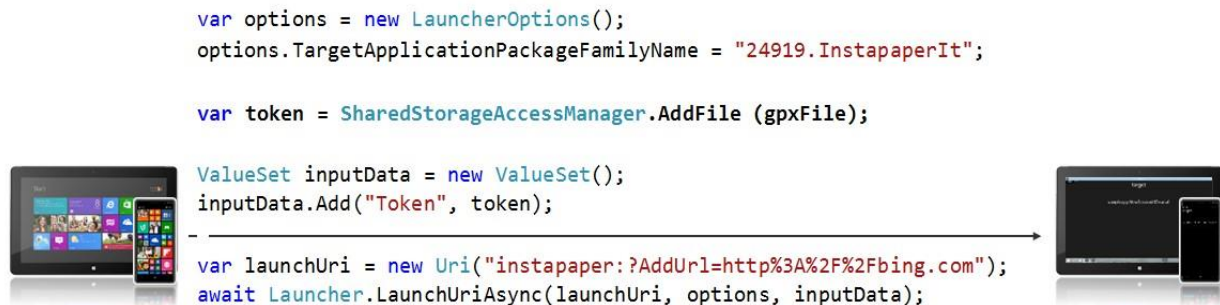


Figure 7 Launching a specific target application with a file token to be delivered

Sending a File, figure 7 – for the Windows platform is basically the same as launching another Application, so the options parameter is set to the specific Target Application, and another parameter “inputData” which is a Value Set (dictionary which you can send across to the Target Application). The value Set is a fully serializable iDictionary (primitive dictionary – easy to pass around). One of the things you could put in a Value Set is a Shared Storage Access Manager “File Token”.

This gives developers the possibility to launch another application with a file added as a token. The file can have information relevant to the user in the point of time when such user experience is needed.

## URI Activation for Device Settings

Another benefit of direct inter app communication is the possibility to offer direct navigation to specific settings pages, relevant to the launching applications needs. That way the user can easily access certain native application’s configuration and use it in an appropriate way.

List of URI available for Settings Pages (Applications) of System, Device, Network & Wifi can be implemented, to ease the user navigation. Useful to take the user from your Application – to – Settings Application of “something”, gives the user confidence that everything is accessible as needed.

## Launch for Results

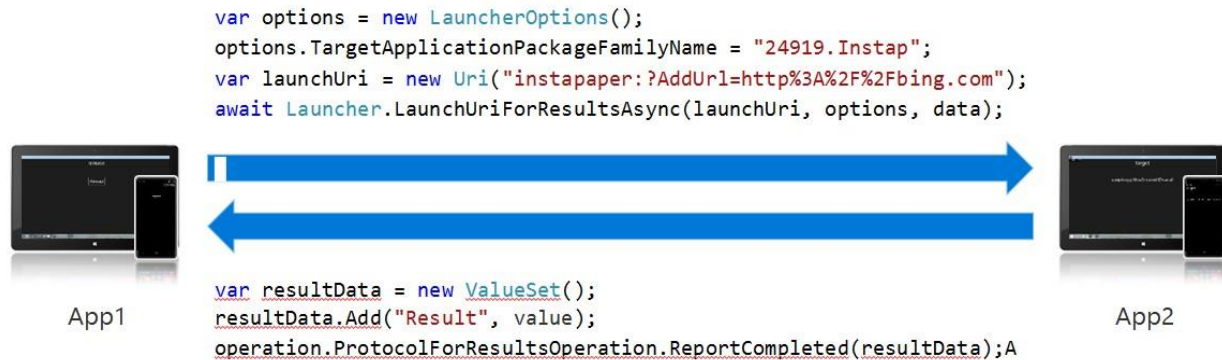


Figure 8 Launching a specific target application for results

Launching an Application and waiting for it to return Results and User Control. This scenario, presented in figure 8, is most common when we are considering the usability to the developers. Use case scenarios can include any form of data. Examples including key information, access tokens for files, images as well as any string can be sent back and forth the applications communicating. The target application based on the input presents appropriate response screens and handles the call. Results are returned to the launching application together with user interaction. This happens after the user interacts with the target app.

For mobile it suspends (freezes) the first Application, until the second Application finishes with User Interaction to return the results. In windows desktop you can interact with both of the Applications (sometimes it suspends the sending Application but this is optional).

**Usual Scenario** that involves launching for results:

Launching the Uri of another Application, and waiting for response from the target application.

It is optional to send data (Value Set) to be processed in the other Application.

The other Application processes the Data, uses its User Interface to communicate with the user and when ready to return Control to the caller Application, calls "ReportCompleted(resultData)" method.

You can serialize the Value set before the call with Json.

## CHAPTER 3.4 WEB OF APPS

The Web is Vast. Vast in space and numbers, and is composed of big and accessible set of applications. The Web is Inter – connected. The accessibility allows easy and reliable connection between the apps. The Web is easy to navigate.

Web of Apps is the byline of this thesis, which comes from the fact that mobile applications are meant to be simple and accessible. On desktop machines the screen is big so it is realistic to use the services of one program for a longer period. The mobile phones on the other hand are more interactive and require dynamical navigation and collaboration between related applications. In other words mobile devices should behave more like web accessible devices and less like desktop devices, and for this it is essential to have good Inter app Communication protocols to support the web of applications.

Inspired by the Web, which is the biggest platform known to Men. Billion plus Applications (Websites) developed for this platform. They work together great. That works for Applications too, Applications on (almost) any Device can reach out to a website: - get Some Data, display Data, make Transactions Happen, send user to Website ...

But when an Application tries to talk to another Application On the same Device? Very rarely, and all of the examples come from applications that come with the android platform. With the latest APIs and future releases this will evolve and become even more important in the developer's world.

Goal of this idea can effectively help developer build a Web of Apps, on user device, so applications talk to each other as often as they talk to Websites. But also still be able to talk to Websites.

This idea consists of two parts that are equally important. Deep linking of applications and Application services. In this paragraphs we will understand how to use the latest APIs that are available to the Custom URL Schema mechanism.

The first thing you want to do, when you are building this Web of Apps, is you want to be able to link to other Applications On the same Device.

LaunchUri / LaunchUriAsync – Can Launch Web Uri (http), but also Launch Uri based on Custom Scheme.

### **Why Are URL Schemes Interesting?** (in the context of Web of Apps)

Have been around since '96, introduced by Internet Explorer, and are used to connect the biggest platform known to men.

Custom Schemes work on All Platforms (Spotify://, launches Spotify Application anywhere)

This is becoming a popular topic as we can see the emerging industry standards:

Applinks (Facebook initiative) is an open cross platform solution for deep linking to content in your mobile phone. Applink (Bing) also connects mobile applications based on custom URL schemas.

Firebase App indexing (Google), like Bing, gets your app into Google search.

So Custom Schemas can Link to any platform (Android, Unix, iOS, ...). Big Tech companies have been interested in the idea of Web of Apps. These initiatives' basic idea is to take a \*http\* Uri and transform it to a Custom Scheme that somebody can use to Launch on the Device.

## Deep Linking Improvements

Windows platform adapts to building Web of Apps. First by the enhancements in the launching methods introduced in the updates of Windows 10, but also by providing useful APIs able to solve the most common problems when implementing the IaC protocol. These APIs and improvements will be discussed in this chapter, while the introduction of app services will be part of the future improvements chapter.

### *Solves the “Evil twin problem”*

There is a big potential for user to end up in the wrong place during navigation between launching and target application. But having a specific Target application to launch and no opportunity for the user to interact with a chooser screen solves this problem. That way on the callback that returns user control, developers use the specific address of the launching application.

Solved by Launcher Operations variable, and setting Package Family Name of the *\*target\** Application. Launching a *\*specific\** target Application without Picker.

Package Family Name – is a unique identifier for your Applications from the Windows Store (not necessarily). (for launching *\*specific\** Application)

### *Sending a File (Through token, and no more direct string)*

Sometimes you want to Launch an Application and send it a file. Because you don't want to take a big file and serialize it into a string.

Shared Storage Access Manager method add file gives you a token, the token is a guide, the guide is a string that you can send to another Application.

The other Application can take that string, put it in a redeem token for file, and get a Storage File back.

## Query URI Support

New API in this kind of set proposed by Windows are very useful. Using them application developers can get information from the operating system in a new way that lets them have a lot more control over the app – to – app communication.

Discovering if the target application has already been installed. Or the enumerated applications able to handle an application call for a specific custom schema. But first lets take a look at the usual user interaction dynamics.

What if the Target Application is not installed on the device? Shell pop up window asking to launch the store application with the appropriate screen with the target app download link.

But on the other hand, now with the Query Uri Support APIs allows developer to handle these interactions in their own launching applications. (now its possible to Query and ask if the target Application has been installed, previously not possible). For this there are two example cases, the first if there is any application to handle the custom Uri and the second is a query for a specific target application.

## Query for Support

The first Launcher method in figure 9 represents a query for a Custom Scheme (myapp). This results can be used to build the application screen in a way that lets the user know about the possibilities to interact with one of those applications.

The second Launcher method queries for a Custom Scheme (myapp) supported by a *specific* app. This API, seen in figure 9, can improve on the use cases the first API is limited on and allow more control over the connection between the applications.

```
var queryUri = new Uri("instapaper:");
await Launcher.QueryUriSupportAsync(queryUri, LaunchUriType.LaunchUri);

var queryUri = new Uri("instapaper:");
string packageFamilyName = "24919.InstapaperIt";
await Launcher.QueryUriSupportAsync(queryUri, LaunchUriType.LaunchUriForResults, packageFamilyName);
```



Figure 9 Query the OS for laC support

## Query to Enumerate Apps

Having an API that returns applications supported by a specific Custom URL schema is an API used to create the third-party screens used as chooser dialogs for application navigation. Now they can be used for directly changing the user interface and informing the user of the possibilities to navigate to one of these applications. In the following pictures we can see a developer made chooser dialog for inter app communication as the ones offered by the platform APIs, by activating an app service in the background.

Find all Applications that support Custom Scheme (myapp) method:

```
var customSchemeProviders = await Launcher.FindUriSchemeHandlersAsync("myapp");
```

Find all Applications that can Launch a File of type .foo method:

```
var fileTypeHandlers = await Launcher.FindFileTypeHandlersAsync(".foo");
```

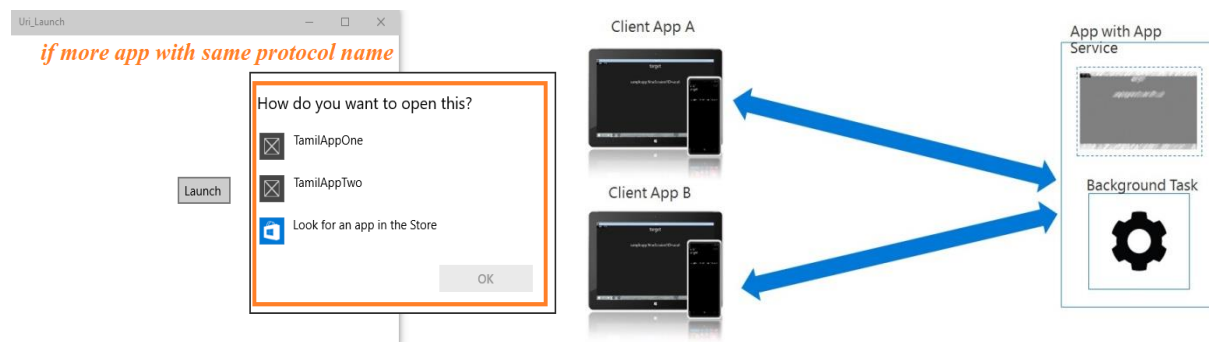


Figure 10a Query the OS to enumerate target apps

Figure 10b Launching an app service

## CHAPTER 3.5

## RELATED WORK

In this chapter we take a look at the way that most of the platforms used to allow applications to communicate in the past. This represents inter app communication, where apps communicate only indirectly with other apps on device. The more restrictive approach we will look into is still used for the IaC in the iOS platform [17] and allows AirDrop to share files and data with other applications. As well as the possibility to define a custom URL schema so that other applications can send information to the application you are developing, and vice versa, using URLs.

A URL Schema allows this communication with the help of IaC protocol that we define later in this work, but in the future will allow extensions that other developers define. To communicate with an application that implements a schema of that protocol, first an appropriate URL must be created and to ask the system to open it. The handling process is provided by a chooser dialog. To implement support for a custom schema, the schema must be declared in the manifest and handle incoming URLs that use the schema.

While Android message passing system promotes the creation of rich, collaborative applications [18]. That goes against the philosophy of application security in iOS, because to allow it also introduces the potential for attack if developers do not take precautions. Outgoing communication can put the application at risk of broadcast theft, data theft, result modifications, etc.

Apart from these technical problems, it is important to provide a balanced trade-off between visibility and usability of the App- to – App communication functionality. This is discussed in detail in [19] and we can see different communication patterns in mobile applications. The analysis takes into account popular applications that provide communication scenarios with or without effect on the user observable application functionality.



## CHAPTER 3.6

## SUMMARY

With this research we analyzed the complex dynamics of the IaC problem. In this chapter we will try to infer the Inter application Communication dynamics on Platform Independent level. In the following chapter we will provide patterns underlying the implications of IaC on User Interaction. If a lot of applications rely on one Knowledge Base (KB), then having a standardized way in which these applications can communicate with each other (IaC) becomes an important concern. Developing services (web service like), supporting IaC type of interactions in a mobile context is equal to defining a set of messages and responses that can be sent back and forth between applications. This set of messages is called Inter app Communication protocol. A table that refers to this message set was introduced in the beginning of this research chapter.

The conclusions from the top-down analysis of the state of the art are simple and realistic. Although most platforms offer different and uneven support for IaC, adaptations and future releases will bridge the gap for platform independent solution.

Android has robust intent system that allows for any kind of app to app communication. iOS uses restrictive Custom Url mechanism, while Windows provides extensive use of the Custom Url mechanism. Driven by these differences, the implementation of IaC Protocol may vary depending on the platform. In Android, apps declare ability to respond to General Intents, this allows two applications to interact with one another => collaboration.

In iOS, the protocol is implemented only by the Gate – keeper application => navigation.

Based on this idea, in the following chapter we try to model high level (reusable) patterns, based on the inferred use cases implied by the dynamics of IaC. With a view on implementation extensibility which will be discussed in chapter 5.

	Android Intent System	URL Custom Schema	Notes
<b>Launch App</b>	Implicit Intent <ul style="list-style-type: none"> <li>Doesn't know anything about resulting context</li> <li>OS enumerates apps based on Intent Resolution</li> </ul>	<ul style="list-style-type: none"> <li>Known Custom Schema</li> <li>OS enumerates apps based on URL custom schema</li> </ul>	Different!
<b>Launch *specific* App</b>	Explicit Intents <ul style="list-style-type: none"> <li>Needs to know the resulting context = Package Family Name</li> <li>(optional) Return User Control with Results</li> </ul>	<ul style="list-style-type: none"> <li>Known Custom Schema</li> <li>(optional) Return User Control with Results</li> </ul>	Similar!
<b>Launch *specific* App Service (future improvements, Data sync. Problem)</b>	Explicit Intents <ul style="list-style-type: none"> <li>Needs to know the resulting context = Package Family Name</li> <li>Return results, show no target UI</li> </ul>	<ul style="list-style-type: none"> <li>Known Custom Schema</li> <li>Return results, show no target UI</li> </ul>	Similar!

Table 3 app - to - app communication use cases

## CHAPTER 4 INTER APPLICATION COMMUNICATION

As a result from the state of the art, we can define a set of use cases as a solution of the model driven approach for app – to – app communication problem. Manipulation of the interaction flow and back-end diagrams represent an IaC use case solution. And at the same time a reusable design pattern for other applications with the same purpose.

### CHAPTER 4.1 INTERACTIONAL PERSPECTIVE

When speaking about mobile applications, user experience – and, more in general, front-end design has a central role. As extensively reported in [9], “A successful user experience is crucial for successful application adoption”. User experience and Inter app Communication mechanisms are closer than we would think in the first place in the mobile environment, since they are hand in glove with performance, which is a major parameter in determining the optimality of a well-thought software. Sometimes the combination of good user interface and satisfactory performance leads to a successful commercial outcome regardless of the originality of the idea inspiring the application; in fact, users tend to prefer applications that look fancy and reactive, even if they are not the best when it comes to fulfill their necessities. To produce such a positive outcome, user-centered design is a sensible option to serve development of mobile applications. As stated in [9], for mobile devices this means users need to feel a continuous thread between the interactors they are maneuvering and the software working. Furthermore, the experience must be delightful, possibly rich and uninterrupted: animated transitions, uncluttered and minimalistic menus and visual elements are frequently cited in design guidelines of the main mobile operating systems. Depending on the functionalities and goals of the application, data-flow events may play a crucial role in terms of interactional limitations: for this reason, correlation between the Inter app Communication logic with the front-end as a fundamental factor may represent a good advice.

In this chapter, we combine the above considerations over interaction with the dynamic data-flow behavior of the IaC problem analyzed in the previous chapter, aiming at the definition of some interactional patterns. Launch App Service (small platform differences) – Data synchronization problem (part of **future improvements**).

Some OS (Windows) need an Application UI = blank, others (Android) implement it with Background Task (to be improved on other Operating Systems).

There is a strong link between the Data synchronization and Inter app Communication problems.

## CHAPTER 4.2

## PATTERNS IDENTIFICATION AND ANALYSIS

**Pattern form.** Patterns are illustrated in compliance with the Coplien Form by James O. Coplien, “one of the founders of the Software Pattern discipline, a pioneer in practical object-oriented design in the early 1990s and is a widely consulted authority, author, and trainer in the areas of software design and organizational improvements” [10]. We slightly alter the pattern form by adding a visual explanation expressed in IFML and UML, so the resulting structure is arranged as follows:

### **Problem**

Statement of the problem(s) that the pattern works to solve.

### **Context**

Definition and constraints of the specific context of applicability of the pattern.

### **Forces**

Scenarios and motivations supporting the pattern logic or application.

### **Visual Explanation**

A diagram or graphical representation explaining the pattern and a description of the interaction between the different elements of the pattern.

### **Solution**

Explanation of how the pattern solves the problem considering the constraints of the context.

### **Resulting Context**

Definition and constraints of the context resulting from the application of the pattern.

### **Rationale**

Philosophy inspiring the pattern

## CHAPTER 4.3

### CHAPTER 4.3.1

## MODEL DRIVEN DESIGN

### IaC Use Cases

There are two ways to start an Application. Starting another application by opening a file is more common for desktop applications where the OS checks extension and provides applications that can handle it. After the user has selected an application from the chooser, the target application builds its user interface based on the contents of the selected file.

More common on the mobile platforms is to start another application by clicking a link where the OS checks the App Manifests and provides applications that can handle it. For this type of communication we send the data through parameters.

There are three global Use Cases inferred from our research (that differ little based on the OS):

- Launch Application (File or Link): Lets OS take care of interaction/communication as a third-party between the linked applications. Where the user input on a chooser dialog, OS generated and enumerated with the available applications, is decoupling and navigating the interaction.

Android Intents don't need to know anything about the resulting context (App). Android OS enumerates Applications that can handle the Intent based on the Intent **type**.

Custom URL Schema (based OS: iOS, Windows) mechanism needs to know the resulting context's (target app) URL Schema, so OS can enumerate the Apps and handle the communication.

- Launch Specific Application (Link or File + (optional) Token): Similar for all OS, provides direct interaction between applications based on the resulting context of the interaction. This is a coupled way of app to app communication, optionally providing two way communication. This way the developer can clearly decide the interaction flow between his and collaborating target applications.

Needs to know the Resulting App Context (Custom app URL Schema or Package Family Name), which is used to create the direct connection to the target application.

(optional) Return after interaction to Launching App (with Results). This way we can use the services of another collaborating application, whether it is back end data manipulation or user interface actions leading to creation or update, and present them back in the launching application.

- Launch App Service (small platform differences) – Data synchronization problem (part of **future improvements**)

Some OS (Windows) need an Application UI = blank, others (Android) implement it with Background Task (to be improved on other Operating Systems).

Link between Data synchronization and Inter app Communication problems.

### 4.3.2.1 Pattern: Launch Application

#### Problem

Providing flexibility and possibilities for User Control of the Inter app Communication.

#### Context

Any application that wants to provide smart navigation orchestrated by the user.

#### Forces

- More than one application is able to handle the app communication request
- No default handling application specified
- No result in return is needed
- The user feels scared to save default (complicated settings page to restore)
- Every time there is an extra interaction for the user

#### Visual Explanation

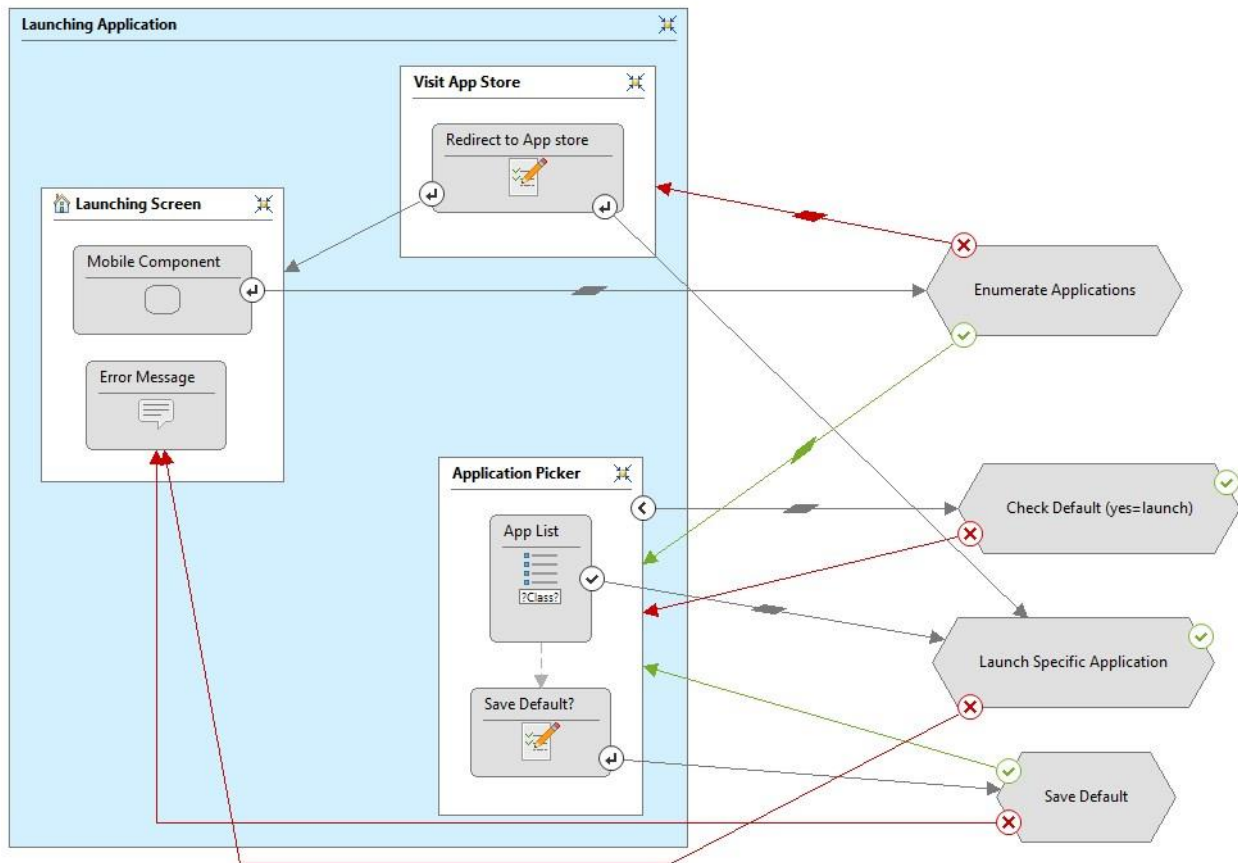


Figure 11 Launching target application using enumerate API

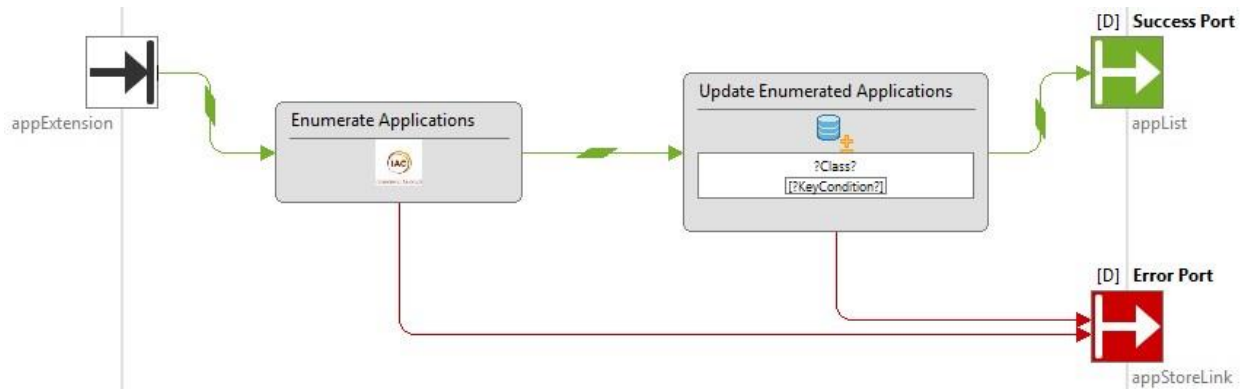


Figure 12 Enumerate applications able to handle the application call

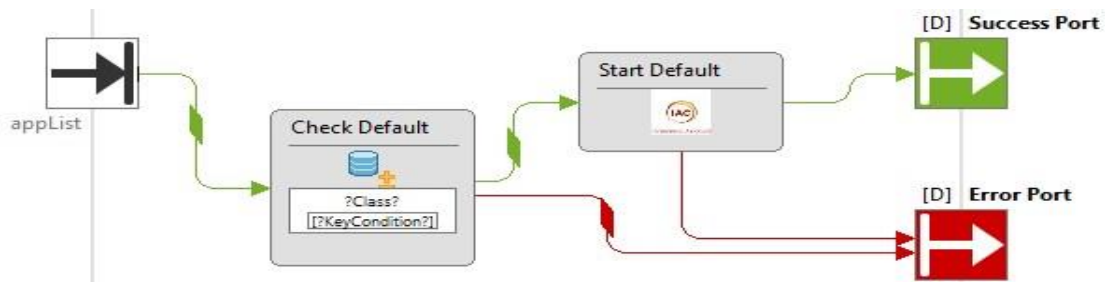


Figure 13 Checking if a default application has been set for the application call

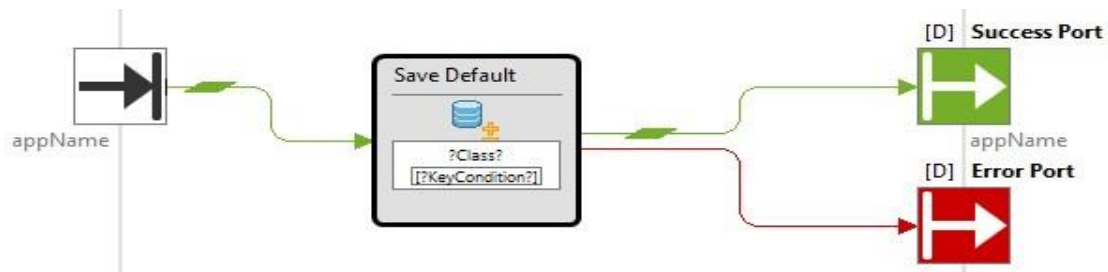


Figure 14 Save selected application as default

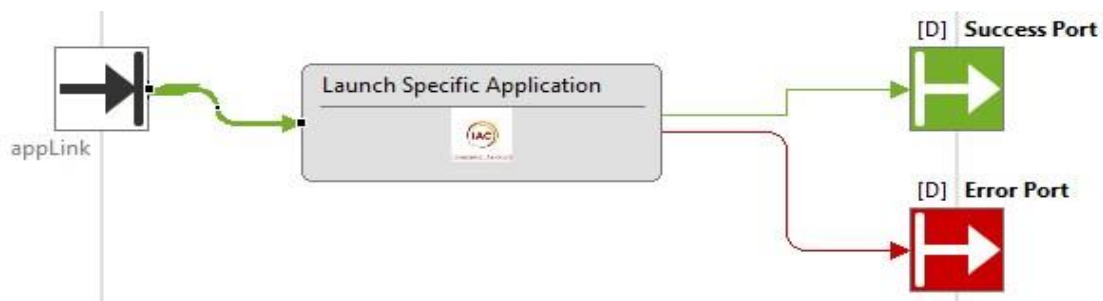


Figure 15 Launch specific application, web ratio custom component

**Solution**

Providing flexibility through User Control of the Inter app Communication mechanism, through user interaction with (App Picker) the Operating System working as a broker.

After user interaction, a background call is activated through Action component to retrieve the Applications able to handle the call with the OS operating as a broker (through App – to – App communication). The results are updated in the database and presented in a list.

Checking if one of the retrieved applications has been marked as default (Background Task). That way before the new chooser is created we can access the default application, with respect to previous user interaction saving that application as the default one.

If so, a direct call is made to launch the default application, with the help of native code calling the appropriate APIs provided by the operating system. Otherwise, no default application has been set previously. So we return back user control to enumerate all the retrieved Apps in a new Modal Screen. Launch the Selected Application by the User through OS background process. If the user instructed, save as default.

**Resulting Context**

Application picked by the user from the Application picker provided by the broker (Operating System).

**Rationale**

The oldest way of providing Inter app Communication, still available and useful. New APIs in some Operating Systems now offer the possibilities for developers to build it manually in their applications. Although a simple launch call with native code activating a chooser dialog can also be useful.

### 4.3.2.2 Pattern: Launch Specific Application

#### Problem

Directly navigating to another application without the Operating System orchestrating the app to app communication as a broker.

#### Context

Any application that wants to provide smart navigation orchestrated by the developer.

#### Forces

- One specific application is in use to handle the app communication request
- Predefined by the developer
- No result in return is needed

#### Visual Explanation

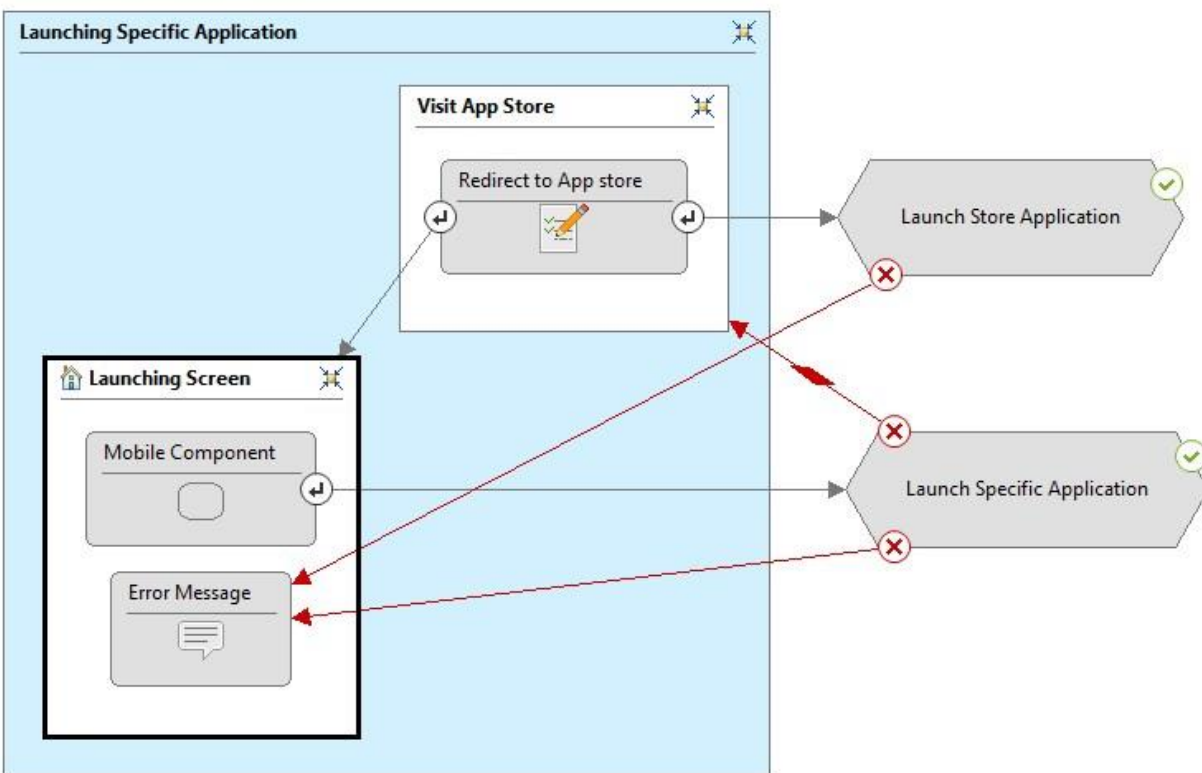


Figure 16 Launching a specific target application without chooser dialog



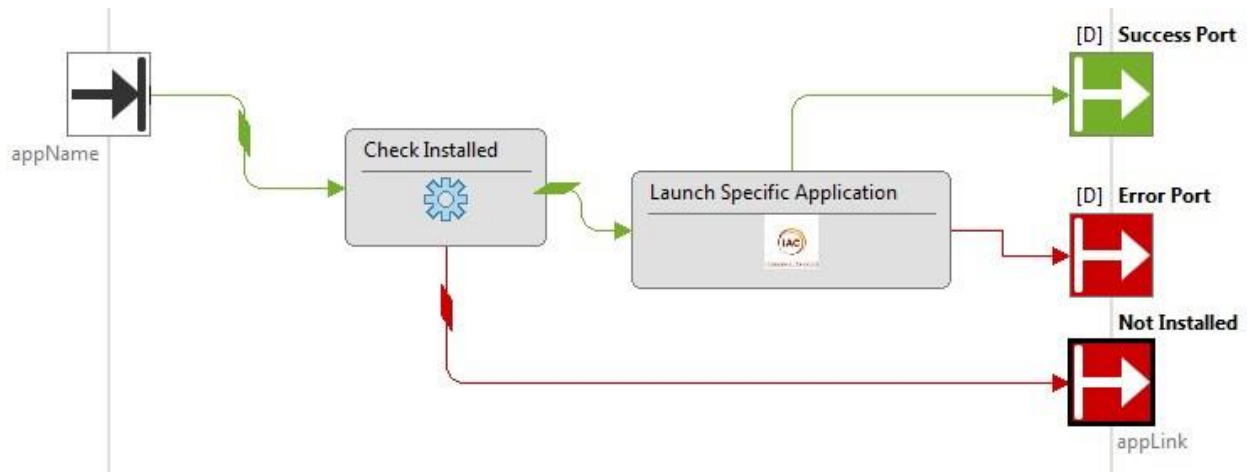


Figure 17 Launching a specific application, if it has been installed

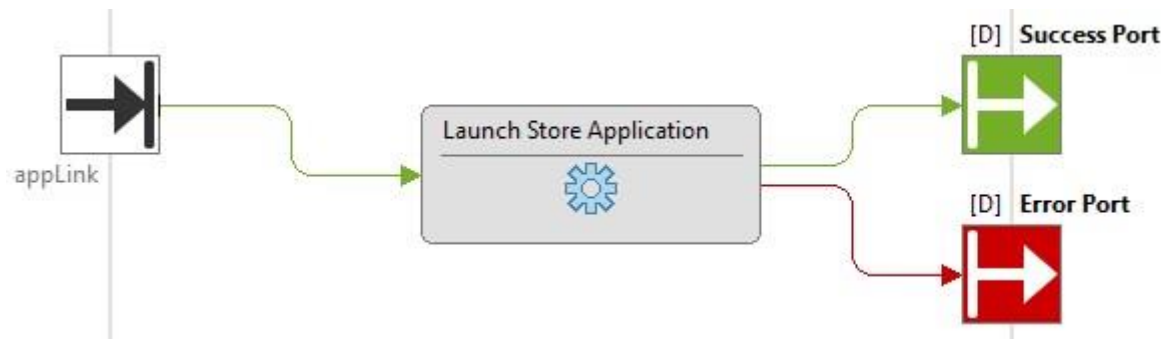


Figure 18 Launching the store application to download the target app

In figure 17 first we try to see if the launching application has been installed and then we start the extended IaC Web Ratio component for launching a specific application using native plugin code.

Launching the Store application is required in the case of error in figure 17 Not Installed, which is reflected in platform specific solution to forwarding the user to the download screen of the target mobile application, represented in figure 18 through optional Web Ratio mobile component. This can be implemented also directly through plugin native code (optionally) in figure 17

**Solution**

Directly navigating to another application predefined by the developer, instead of using the Operating System as a broker.

After User interaction, OS Background call is issued to check if the designated App (by the developer) has been installed on the Device.

If so, launch the specific application through OS background process. The launch direct API is used to provide this functionality by extending the UML modeling language and complementing this extension with Apache Cordova plugin providing the native implementation.

On the other hand, provide direct application store navigation, in a new Modal Screen.

**Resulting Context**

Application predefined by the developer.

**Rationale**

Allowing the developers control over the Inter app Communication, depending on the Operating System's philosophy over application stability – flexibility tradeoff. Having direct way of communication between applications should be used to complement the communication using chooser dialogs, that way the user experience will be enriched by a variety of options to navigate between correlated applications.

### 4.3.2.3 Pattern: Launch Specific Application for Results

#### Problem

Directly navigating to another application for results, without the Operating System orchestrating the app to app communication as a broker.

#### Context

Any application that wants to provide smart navigation for data manipulated results from another application, orchestrated by the developer.

#### Forces

- One specific application is in use to handle the app communication request
- Predefined by the developer
- Data manipulated in the target application and returned as a result with the user control

#### Visual Explanation

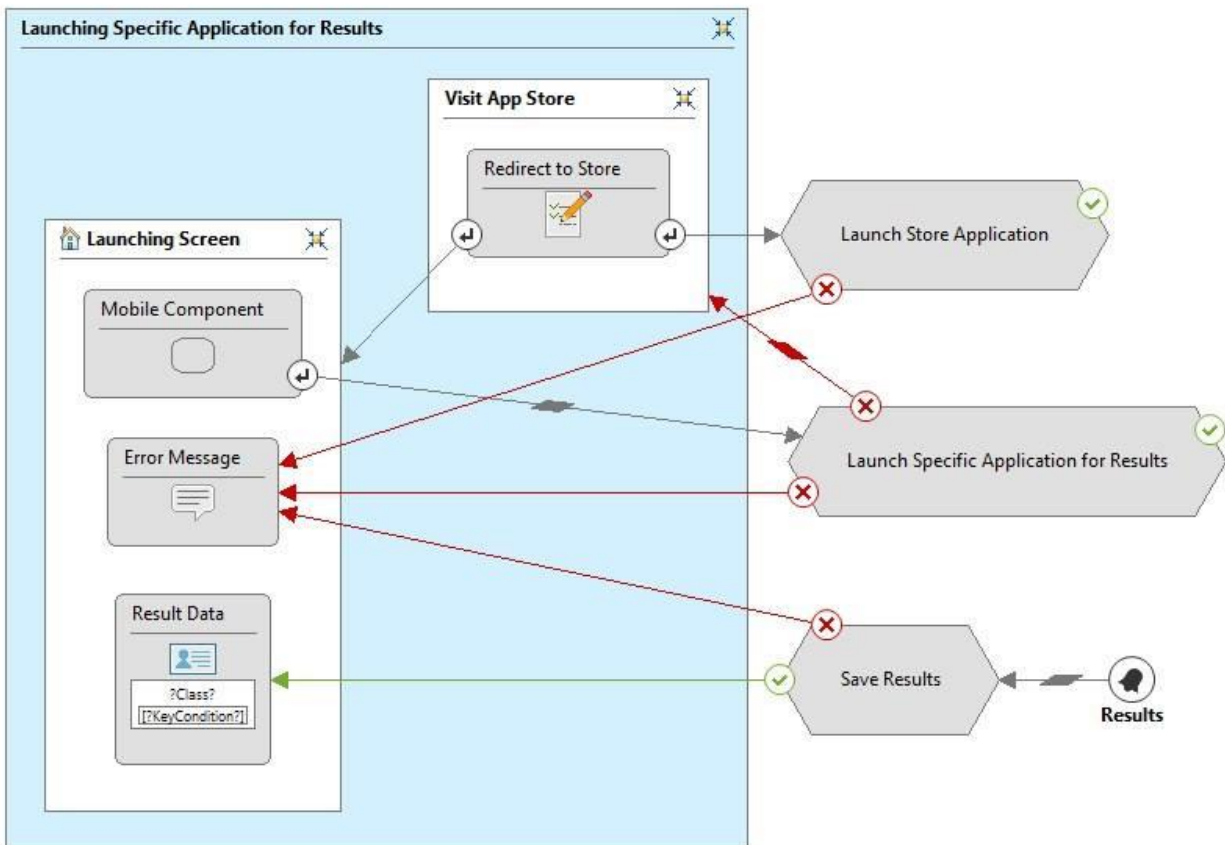


Figure 19 Launching a specific target application for results

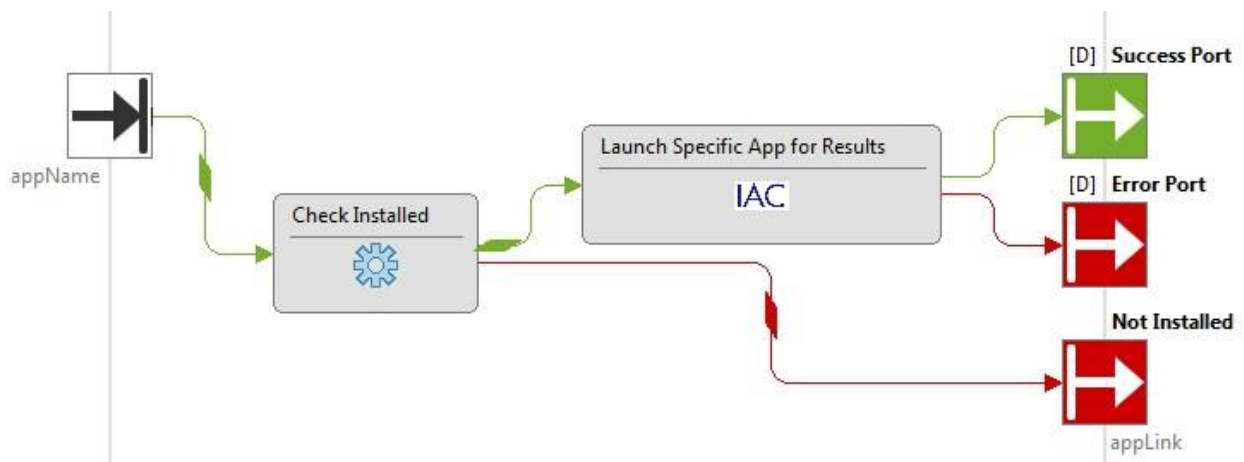


Figure 20 Launching specific application for results if it has been installed

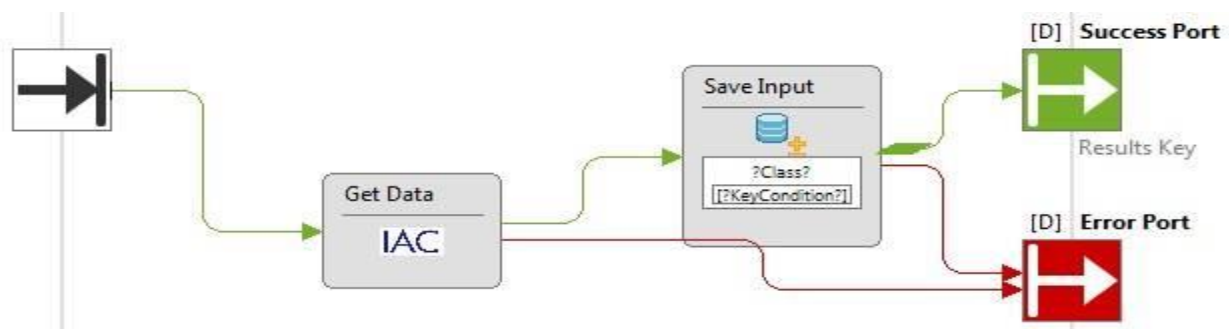


Figure 21 Saving the input data from the target application

In figure 20 first we check if the target application has been installed on the device, if so then we launch a specific method from the native plugin code that starts the specific application. But also prepares with native plugin code and updates on the configuration and application manifest xml files. That way the application is going to be able to handle the returned results, through similar application call back from the target application.

**Solution**

Directly navigating to another application predefined by the developer, instead of using the Operating System as a broker. Data manipulation in the target application and results returned with user control back to the Launching Application.

After User interaction, an operating system background task is started, through the action component, a call to check if the designated application (by the developer) has been installed on the device.

If so, previously designated app by the developer is launch as a specific target application. Through OS asynchronous background task that and wait for system notification to return results and user control back from the target application. Get Data action component is responsible to activate the APIs and provide information for the application to present later in the result data component.

Provide direct app store navigation, in a new Modal Screen, in case the designated application hasn't been installed.

**Resulting Context**

Target application predefined by the developer.

**Rationale**

Allowing the developers control over the Inter app Communication, depending on the Operating System's philosophy over application stability – flexibility tradeoff. Mechanism for the return of user control and results to the launching application. Direct navigation to another application, but expecting a result back and preparing for their display with the return of user control.

### 4.3.2.4 Pattern: Specific Application

#### Problem

Developing application able to handle requests for Inter app Communication, manipulate the data and return results and user control back to the launching application.

#### Context

Any application developed to handle specific Inter app Communication requests.

#### Forces

- Handling requests of Launch Specific Application calls to display smart user navigation
- Handling requests of Launch Specific Application for Results to manipulate the input data and return results and user control back to the launching application

#### Visual Explanation

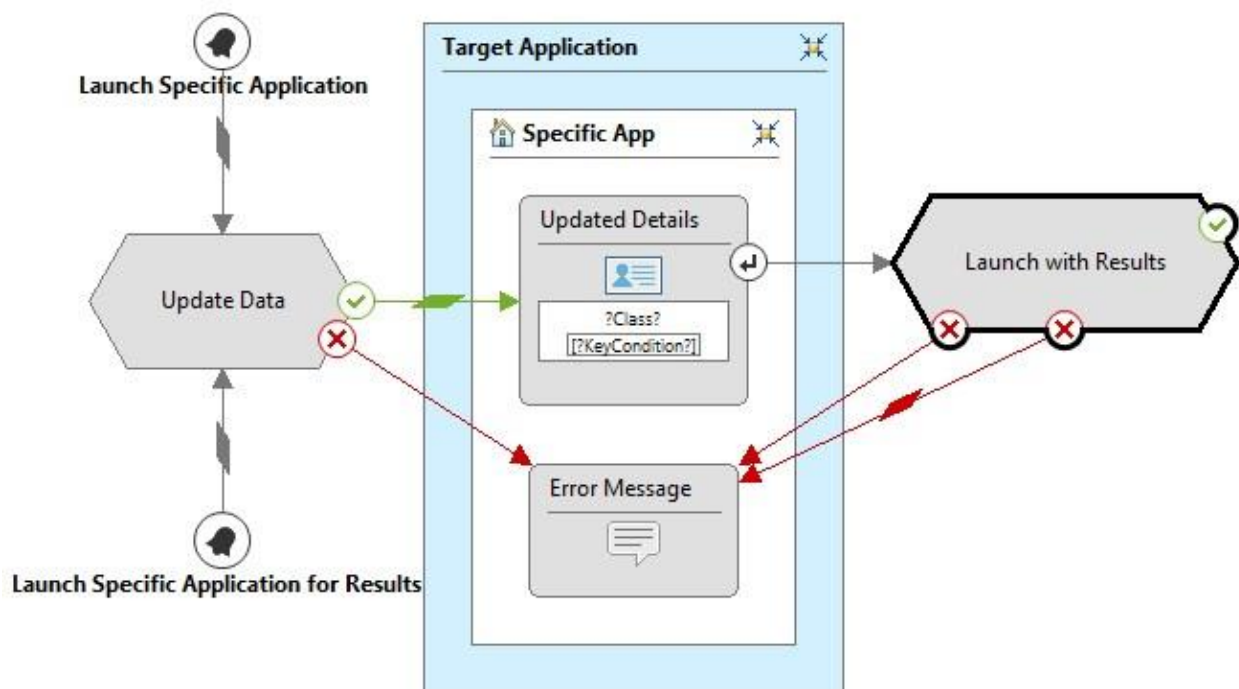


Figure 22 Target application able to handle launching calls from other Web Ratio apps

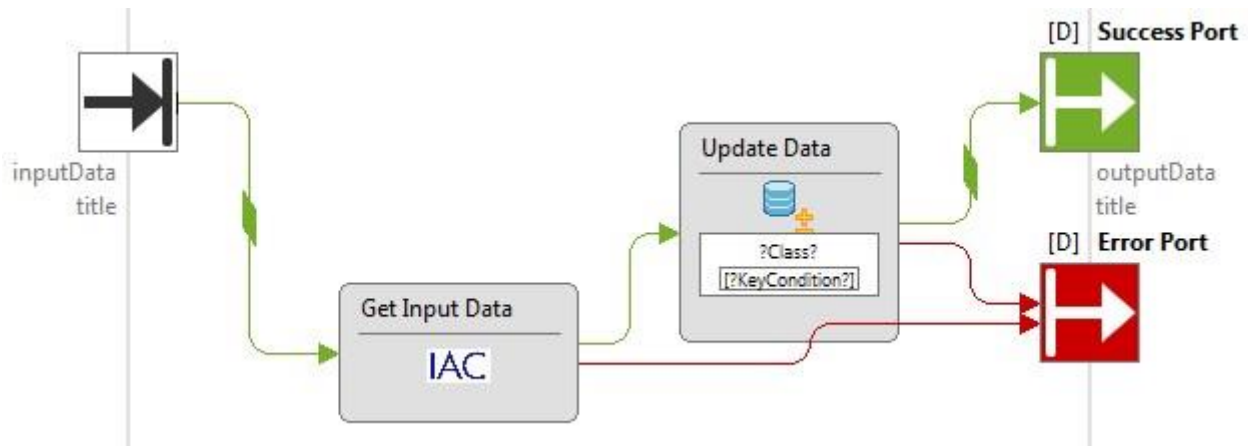


Figure 23 Updating the input data from the launching application

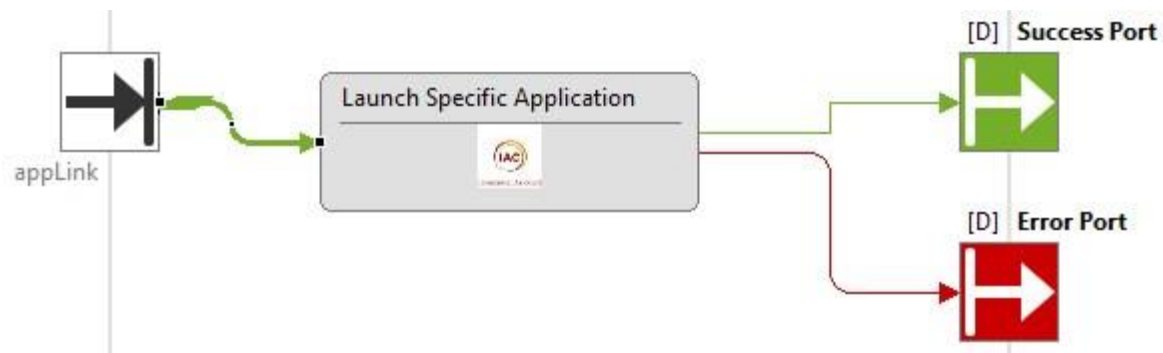


Figure 24 Return results and user control back to launching app, web ratio component

Figure 23, getting the input data from the launching application in MDE can be accessed only through the platform native code implemented. That way the IaC protocol component accesses a plugin in the target application through a Web Ratio mobile component. The results as JSON object are updated in the database to be represented in the user interface.

In figure 24, we see a web ratio mobile component previously used to start a specific application, here the previous plugin method is used to activate the results action and to return user control back to the launching application by starting it.

**Solution**

Developing application able to handle requests for Inter app Communication, manipulate the data and depending on the request (optionally) return results and user control back to the launching application.

Launch Specific Application, represented as system notification, starts the target application able to handle the request. First it activates the appropriate APIs to access the content of the input data. This information is optional, but if received usually is modified and displayed accordingly in the updated details component.

The same is true for the Launch Specific Application for Results system notification, starts the target application able to handle the request, updates the input data and displays it. Except now it allows the possibility to activate the Launch with Results action.

After this user interaction, returns results and user control back to the Launching Application using WCO to provide the extensibility and native code accessing the appropriate APIs.

**Resulting Context**

The target application, if the call was made with the purpose of smart navigation.

Or the launching application, if the call was made with purpose to return back results and user control.

**Rationale**

Application able to handle different request types for Inter app Communication, providing mechanism to handle them appropriately. For direct call or one through a chooser dialog, not expecting results in return, it simply displays the input data from the launching application. The mechanism for application calls expecting results in return, after display and user interaction starts a direct application call back to the starting application.



## CHAPTER 5 IMPLEMENTATION OF APP – TO – APP COMMUNICATION

To realize a platform independent solution for the app – to – app communication problem, one would have to provide a separate platform specific implementation, in order to bridge the gap between their respective constraints. This uniform solution would be able to provide as service the four distinct use cases previously defined for specific platforms in the mobile environment.

This chapter for the implementation of app – to – app communication in the android platform, will be provided as a reference guide for model driven development of a real world application using the Inter application Communication protocol.

Web Ratio Mobile Platform is a model driven software development tool that under the hood relies on Apache Cordova to create platform independent mobile applications. Since there is no component available for the action of IaC, one has to be developed for each platform. App – to – App communication is different for every platform, so this can't be solved by Apache Cordova JavaScript logic, it requires a native solution developed through a Cordova plugin.

Apache Cordova plugin represents a native solution for each mobile platform that bridges some functionality between Web View of Cordova application and the native platform the Cordova application is running on. In this chapter we will present the necessary components leading to the composition of App – to – App communication plugin for the android platform.

The implementation of these plugins in the MDE is accomplished through the development of Web Ratio Custom components, they extend the modeling language to define the new feature. The IaC component is developed for the Android platform, as a Web Ratio Mobile operation component. In this specific tool, the View Component extends the IFML modeling language, while the Operation component extends the UML modeling language. Since the background task of IaC is not provided, we develop an operation component.

The native code bridging the App – to – App communication functionality and its implementation in a modeling application scenario will be presented and analyzed in the subsequent subchapters.

## CHAPTER 5.1

## WEB RATIO MOBILE CUSTOM COMPONENTS

Currently in MDD, there is support for the development of (hybrid) mobile applications using the Web Ratio Mobile Platform, a model-driven development environment that allows for the creation of cross-platform mobile applications. Under the hood, relies on the power offered by apache Cordova to allow developers create mobile applications that can be deployed in the major mobile platforms.

In order to seamlessly integrate with the native mobile functionalities like contacts information, messaging, camera, GPS and others, Web Ratio allows for modeling extensions that can be implemented through specific native plugin code.

Web Ratio Mobile Custom Operations (WCOs) [11] are tailor-made modeling extensions for the Web Ratio Mobile editor (figure laComponents), that can be used to achieve a specific task. In this case the specific task is developing WCOs for App – to – App communication in the Android platform. This component will be able to handle the four IaC use case scenarios.

Figure laComponents

Instructions to create WCOs:



1. Create Mobile Components Project
2. Create Mobile Component (custom) : ViewComponent / Operation / both
3. Set General Properties
4. (optional) Set “native plugin” developed specifically to bridge:  
Apache Cordova => Native platform (for the new component functionality)
5. Specify Component Inputs (IaC input = Uri)
6. Specify Component runtime behavior  
(JavaScript service logic = call plugin exec + success/error flow)
7. Specify Component outputs (none)

The App – to – App Web Ratio Custom Operation will add the appropriate functionality through Apache Cordova plugin to the native code to orchestrate the seamless collaboration between applications in the Android platform. In the following chapter we will take a look at how this plugin is realized. As an input to the WCO component, we get the User’s choice of preferred application to launch. The name of this application is also sent as an input to the Apache Cordova plugin. There is no specified output for the component.

The component’s runtime behavior is specified through JavaScript service, in the execute operation method we make the call to the component’s native plugin method, to access the native bit of functionality (starting a new target application).

## CHAPTER 5.2

## APACHE CORDOVA PLUGIN

Plugins [12] are packages of injected code and are an integral part of the Cordova ecosystem. They provide an interface for Cordova and native components to communicate with each other and bindings to standard device APIs. This enables developers to invoke native code from JavaScript interface.

Apache Cordova project maintains a set of plugins called the Core Plugins. These core plugins are intended to provide to the mobile application access to device capabilities such as battery, camera, contacts, etc.

In addition to the core plugins, there are several third-party plugins which provide additional bindings to features not necessarily available on all platforms. We will develop Inter app Communication Cordova plugin specifically for the Android platform, which later can be extended for other platforms based on the previously defined models for the Custom URL schema IaC mechanism.

IaC plugin allows the Cordova web view within which the application renders to communicate with the native android platform on which it runs. On the web view side it allows the execution after submit, navigate or select event. These events access the methods specified in the IaC Start Activity plugin JavaScript interface, those are the four IaC use cases previously defined. On the other side of the plugin, the native android java implementation, the Cordova exec function has been mapped to the execute method of the Start Activity a Cordova Plugin seen in figure 26.

Through these components and the plugin xml file connecting them and presenting them as accessible Apache Cordova plugin named Start Activity for the android platform. It provides access to device and platform functionality that is ordinarily unavailable to web-based applications. Not only our new IaC implementation, all the main Cordova API features are implemented as plugins and many others are from other developers or simply possible to implement.

First we will take a look at the integral parts of the components which compose the IaC Web Ratio Custom Operation component. This plugin comprises of a single JavaScript interface along with the corresponding native code library for the android platform. In essence this hides the native code implementation behind a common JavaScript interface. All of these essential components are linked in the plugin xml file, part of which can be seen in figure 25. What we see is specifically important for allowing other applications to start the developer's application, needed for the return of results from the target application back. This way applications can collaborate with each other to provide better user experience. For these reasons and the fact that Web Ratio WCOs link the IaC functionality through the plugin xml file it is important to specify these instructions in the file.

```

<config-file target="AndroidManifest.xml" parent="/*">
  <uses-permission android:name="android.permission.INTERNET"/>
  - <activity android:name="iac.polimi.ob1" android:label="ob1 Application">
    - <intent-filter>
      <action android:name="android.intent.action.VIEW"/>
      <category android:name="android.intent.category.DEFAULT"/>
      <!-- <data android:schema="iac.polimi.ob1" /> -->
    </intent-filter>
  </activity>
</config-file>

```

Figure 25 IaC using plugin xml file to update the application's configuration files

The JavaScript interface of Cordova plugin (figure 26) is used in Web Ratio WCO service JavaScript by launching the exported methods in the plugin interface. These methods not only are the link between Web Ratio and the plugin, but also using the Cordova exec method they are the link between the interface and the execute method of the Start Activity Cordova plugin later seen in figure 27.

```

var exec = require('cordova/exec');
var StartActivity = {};
StartActivity.launchApp = function(appName, success, error) {
  exec(success, error, 'StartActivity', 'launchApp', [appName]);};
StartActivity.launchSpecificApp = function(appName, success, error) {
  exec(success, error, "StartActivity", 'launchSpecificApp', [appName]);};
StartActivity.launchAppForResults : function(appName, success, error) {
  exec(success, error, 'StartActivity', 'launchAppForResults', [appName]);};
StartActivity.recieveResults : function(appName, success, error) {
  exec(success, error, 'StartActivity', 'recieveResults', []);};
module.exports = StartActivity;

```

Figure 26 Javascript interface of IaC Apache Cordova plugin

```

public class StartActivity extends CordovaPlugin {
  @Override
  public boolean execute(String action, JSONArray args,
    CallbackContext callbackContext) throws JSONException {
    String appName = args.getString(0);
    if (action.equals("launchApp")) {
      this.launchApp(appName, callbackContext);
      return true;
    }else if (action.equals("launchSpecificApp")) {
      this.launchSpecificApp(appName, callbackContext);
      return true;
    }else if (action.equals("launchForResults")) {
      this.launchForResults(appName, callbackContext);
      return true;
    }else if (action.equals("recieveResults")) {
      this.recieveResults(appName, callbackContext);
      return true;
    }else return false;
  }
}

```

Figure 27 IaC Cordova Plugin Android implementation execute method

In figure 27 we handle each method call depending on the action picked from the IaC plugin JavaScript interface. After directing the use case to the appropriate method handler, we present in table 4 part of the java code creating the options for the native android method to be launched. The options and methods belong to the appropriate method calls, based on the summary of the state of the art and the analysis of the Android intent system mechanism summarized in chapter 3.2.

Android implementation	Scenario
<pre>sendIntent = new Intent(Intent.ACTION_VIEW); sendIntent.putExtra("sms_body", message); sendIntent.putExtra("address", phoneNumber); sendIntent.setData(Uri.parse("smsto:" + Uri.encode(phoneNumber))); this.cordova.getActivity().startActivity(sendIntent);</pre>	<p><b>Launch through chooser</b></p> <p>Action view, without a package name, starts a chooser dialog of apps.</p>
<pre>String defaultSmsPackageName =     Telephony.Sms.getDefaultSmsPackage(this.cordova.getActivity()); sendIntent = new Intent(Intent.ACTION_SEND); sendIntent.setType("text/plain"); sendIntent.putExtra(Intent.EXTRA_TEXT, message); if (defaultSmsPackageName != null) {     sendIntent.setPackage(defaultSmsPackageName); }this.cordova.getActivity().startActivity(sendIntent);</pre>	<p><b>Launch directly</b></p> <p>Action send, with package name, starts directly the SMS messaging application.</p>
<pre>private void pickContactAsync() {     final CordovaPlugin plugin = (CordovaPlugin) this;     Runnable worker = new Runnable() {         public void run() {             Intent contactPickerIntent = new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);             plugin.cordova.startActivityForResult(plugin, contactPickerIntent, CONTACT_PICKER_RESULT);         }     };     this.cordova.getThreadPool().execute(worker); }</pre>	<p><b>Launch for results</b></p> <p>Action pick, launched for results in a new thread, expecting results in return. The Contacts app is started to provide the user with the possibility to choose the results to be returned.</p>
<pre>public void onActivityResult(int requestCode, int resultCode, final Intent intent) {     if (requestCode == CONTACT_PICKER_RESULT) {         if (resultCode == Activity.RESULT_OK) {             String contactId = intent.getData().getLastPathSegment();             Cursor c = this.cordova.getActivity().getContentResolver().query(RawContacts.CONTENT_URI,                 new String[] {RawContacts._ID}, RawContacts.CONTACT_ID + " = " + contactId, null, null);             String id = c.getString(c.getColumnIndex(RawContacts._ID));             c.close();             JSONObject contact = contactAccessor.getContactById(id);             this.callbackContext.success(contact);             return;}} }</pre>	<p><b>Receive results</b></p> <p>On activity results, the intent's data is retrieved with a cursor, to provide the appropriate JSON format of the results. In this case the selected contacts by the user.</p>

Table 4 Android implementation of the IaC protocol

In table 4 we see the native android implementation of the Inter application Communication protocol. We can conclude that as defined in table 2, this implementation of the Android Intent system IaC mechanism provides appropriate method handlers as Web Ratio Custom Operations for all of the App – to – App communication use cases.

## CHAPTER 5.3 WEB RATIO MOBILE APPLICATION

This app will talk to other apps. It represents a reference guide for model driven development of a real world application using the Inter application Communication protocol. As we saw in the previous chapter in table 4 the Android platform implementation of the IaC protocol is conveyed through Web Ratio Custom Operations (WCOs).

We have seen already how the WCOs are used in practice when we presented the patterns for the IaC Custom URL Schema mechanism. In this sample application we present an Alarm messaging application for the Android platform using the previously defined IaComponents Web Ratio Custom Operations.

Alarm Android application has been implemented using the MDE approach and here we present the appropriate IFML and UML diagrams. It provides the possibility to start an App Service Send SMS, direct navigation to the SMS Messages mobile application and chooser dialog between the available messaging applications (Facebook and Messages). Screenshots figures 32 – 37 are presented of the applications.

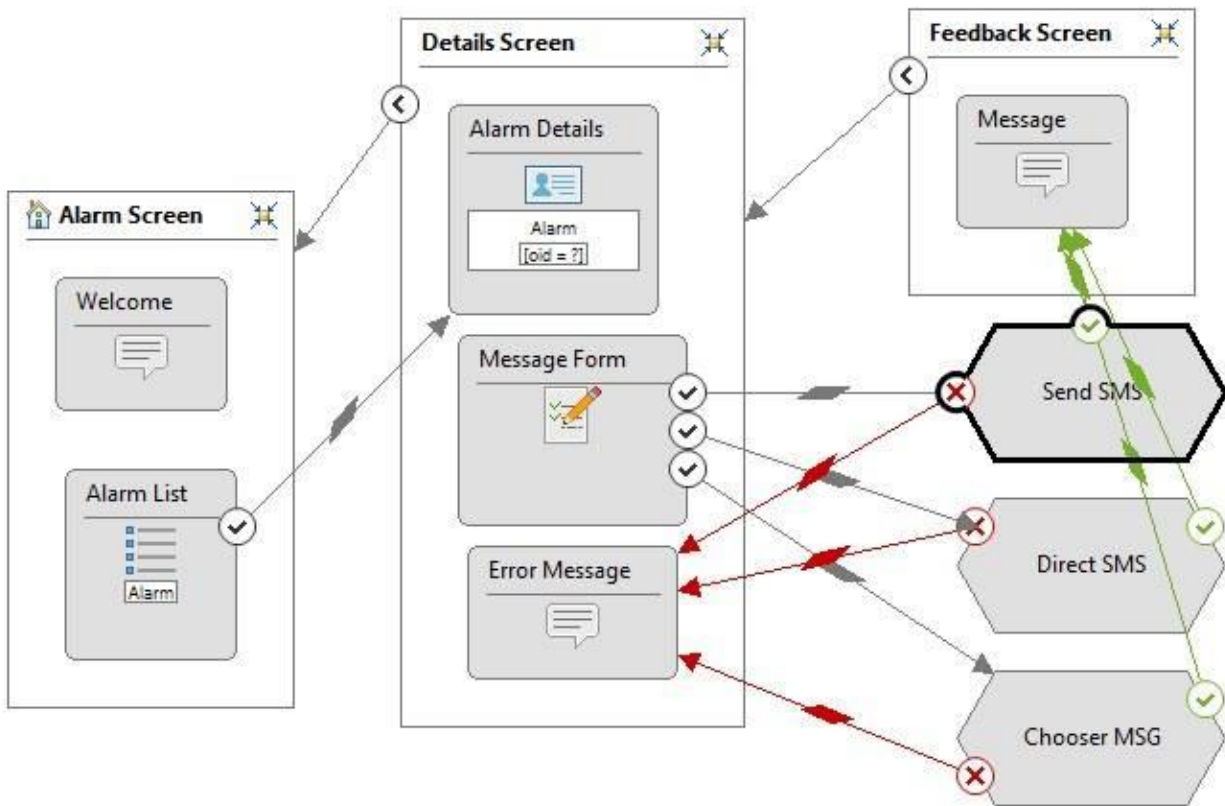


Figure 28 Alarm application IFML diagram

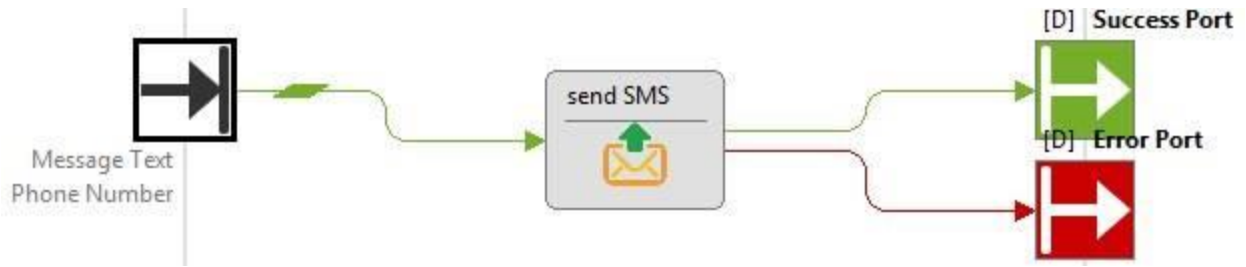


Figure 29 Send SMS App Service Web Ratio Custom Operation component

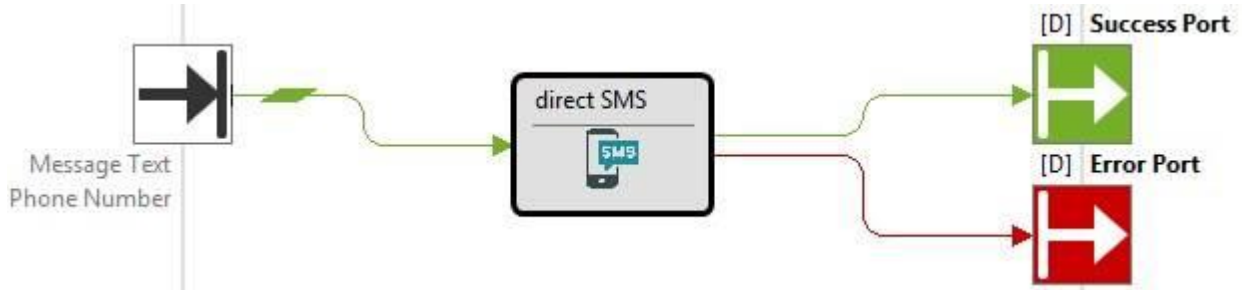


Figure 30 Direct SMS Messages Application navigation, Web Ratio Custom Operation component

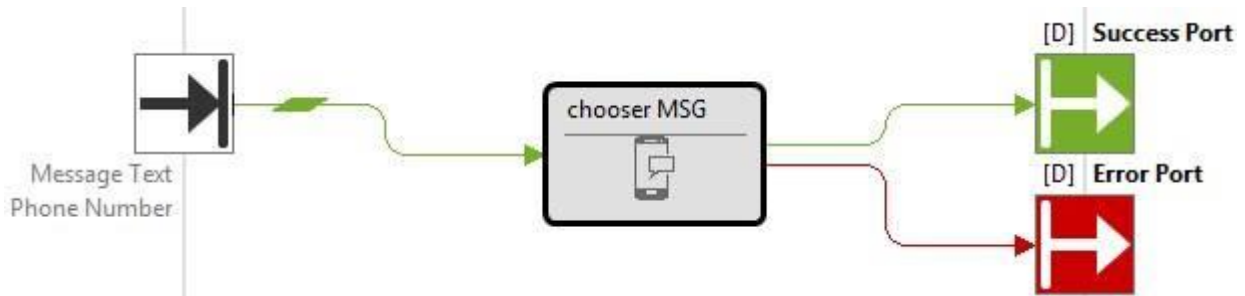


Figure 31 Navigation to chooser dialog presenting all messaging applications Web Ratio Custom Operation component

The implementation of the respective WCOs from the Action UML diagrams have been connected to the Apache Cordova plugin described in the previously. Figure 30 and 31 are direct implementations of the laC protocol, represented by the first two rows of table 4. While Figure 29 is a App Service scenario explained in the following chapter about possible future improvements.

## CHAPTER 6 FUTURE IMPROVEMENTS

### Future Developments

After implementing the proof of concept there are several lines of work that will open, e.g:

- Extending the proof of concept so that it exercises a good part of the messages defined in the Inter App Communication Protocol
- Implementing the communication protocol in the iOS version of the Target app, and reviewing the impact that the **constraints** imposed by the platform may have in the desired use cases.
- Split view or multi window mode [16], displaying more than one application at the same time. Providing real time app – to – app communication between the interacting applications.
- **App Services** [14]

The second big piece of the Web of Apps puzzle, other than linking application, are the App Services. In Android, App Services are represented as Background Services (four distinctive service types). The difference between Azure App Services and Windows and Android App Services is that the azure ones live on the cloud, while the later live on the Device. There are a lot of Web Concepts in App Services, but there is no actual Web Server involved. It is common sense that a Web Server is too big for a mobile device.

#### *The IDEA. What is an App Service?*

With App Services – store Applications can provide Services to other store Applications.

They behave just like Web services, but they run on the mobile device.

Applications provide Services to other Applications (like receive piece of data or send data/file/image).

You can provide Service to other Applications with Launch Uri or Launch Uri For Results APIs as well.

But in the specific case of App Services you want to Communicate with the other Application without bringing up its User Interface.

#### *How does an App Service work?*

Windows platform: execute code without showing User Interface, falls in the general umbrella of Background Tasks. An application with an App Service is a Package with Background Tasks to be executed and return results from target application in the background. The developer gives these Background Tasks names (App Service Name) and you register them in the Application Manifest xml configuration file of the target application that provides the service.



Two way connection is provided for the App Service to allow the ability send and receive Value Sets & Files back and forth between applications. The launching application is the one using the manifest app configuration file to locate the offered App Services and establish a two way connection with them. That way it can access them and provide that bit of functionality in the target application as a service in the launching one. If another application asks for the same App Service (at the same time)? Like the Web, another instance of the same Background Task that serves the other application – only if it is the same App Service at the same time! So if the first application tears up its connection, there are no side effects on the connection between the other app and the new instance App Service.

*App Service Lifetime cycle steps [15]:*

- Service is activated on-demand
- Client may terminate Service (dispose App Service connection)
- If the invoking App is suspended, App Services sponsored by the App will be terminated
- Insufficient resources may cause Launch failure or service termination.

App Services are launched by Client apps. We think of them as something sponsored by Client App. Why we think like that? This is device with battery, and we don't want it to run any faster, so as soon as Client goes away, we terminate App Services we lost. But if the Client is around, the User is using it, your App Services will run. Concurrently using App Services leads to waste of memory! (Many resource, many processors in mobile devices). So all Applications can talk to each other and handle the collaboration in the background, not possible if they were not connected through their respective app services.

For example if an App Service Provider, offers services to App Service Consumer:

App Service connection is opened in a new instance for each Service call. It doesn't close until all messages in the two way connection are sent and received. The background task (while open, invoked by App Client), gets App Service details and connects a method on request receive to handle received messages (asynchronously sent by App Client). After handling the received messages, it sends back a Value Set as a result. The App Service is registered in the manifest file of the hosting application.

## **Background tasks and Inter App Communication in Android**

Service is an Application component that can perform long running operations in the Background and does not provide User Interface. Allow similar possibilities and are the android definition of App Services. There are two ways to use a service in Android. By sending a request to start a service with a particular intent (the location of the target application) in order to execute unattended work in the Background. Or to create a long-standing connection to interact with it.

There are four combinations of Android Service Types:

Local Service, which handle local background tasks in an synchronous way. Intent Local Service handle asynchronously local background tasks.

Remote Service handles remote tasks in a synchronous way, while Intent Remote Service handles the remote tasks in an asynchronous way.

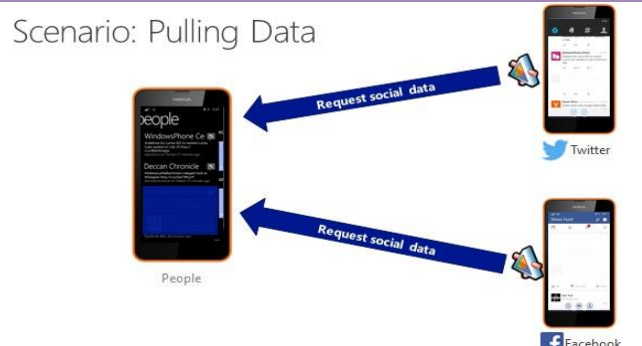
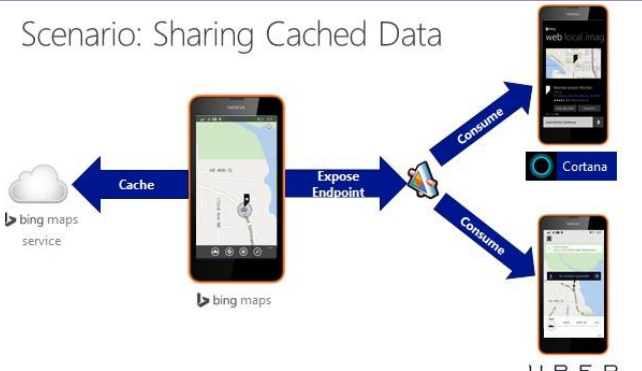

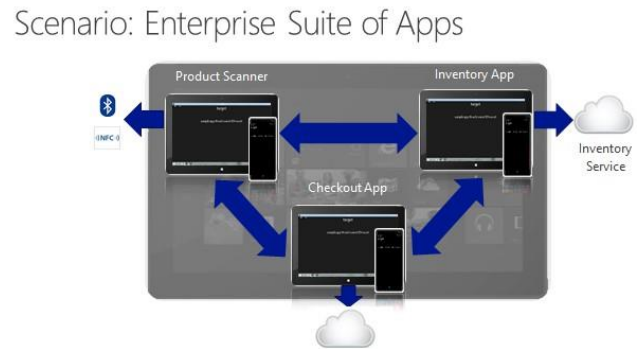
Scenarios	Explanation
<p><b>Pull Data</b></p> <p>Scenario: Pulling Data</p> 	<p>People App is data presenting client application which includes address book data and can also pull news feed for contacts. The client application invokes App Services from Twitter and Facebook mobile applications to pull data to present to the user with the goal to enhance user experience.</p>
<p><b>Sharing Cached Data</b></p> <p>Scenario: Sharing Cached Data</p> 	<p>Client Apps Cortana and Uber both invoke the data hosting application that uses map control. Map Control on hosting app ⇔ Bing maps Web Service, in order to pull data to present. Before it is activated and downloaded, the data from Web Server is exposes as an App Services: Bing maps App ⇔ Map Control to client apps.</p>
<p><b>Hardware Access</b></p> <p>Scenario: Hardware Access</p> 	<p>App talks to a hardware Device: Band App ⇔ Band hardware device. Band App is a data hosting application that provides App Services for historical data + observations on that data. Client Apps instead of talking to the Band (Live Data only), they talk to the App Services the Band App provides. Band App can arbitrate the access to Band device Data. Provide App Services for Historical data and Observation on that data, even if the Band is not connected at that moment (not Live).</p>
<p><b>Enterprise Suit of Apps</b></p> <p>Scenario: Enterprise Suite of Apps</p> 	<p>Enterprise employees don't use only one Application, usually use a suite of Applications. Some custom Applications and other coming from third-parties. Those Applications need to communicate with each other to provide better IT work support and seamless collaboration. (App Services + Deep Linking)</p>

Table 5 Application Services Scenarios

## CHAPTER 7 CONCLUSION

From this thesis and based on the state of the art, we can conclude that App – to – App communication in the mobile environment can be solved in a platform independent way. The MDE approach that we took in the implementation proves this point, and that is just one way to provide this solution adapted to the relevant platforms.

This problem is relevant to the mobile environment, because of the dynamical nature of mobile device usage. That is also why to adapt each solution to all the platforms and to provide this feature is very important for the improvement of the user experience. With future platform updates introducing parallel usage of two applications on a mobile device, the need for adaptation to app services and more platform specific solutions is obvious.

By developing an Android, platform specific, implementation of the IaC protocol we have a solution for one of the IaC mechanisms – Android Intent System. For the Custom URL Schema mechanism we have presented the appropriate Web Ratio Mobile Platform solutions in chapter 4 – IaC Models. The appropriate Web Ratio Custom Operation components and Apache Cordova plugin adaptations need to be developed based on the instructions of their development presented in chapter 5 in their respective subchapters.



Reusable IaC components for the Web Ratio Mobile platform. Adaptable to launching other applications and different platform solutions. They are different Web Ratio Operations that use different execution methods of the same Apache Cordova IaC Plugin. The Plugin currently adapts to the Android implementation of the IaC Protocol, but in chapter 4 we present how it can adapt to the Custom URL Schema mechanism as well.

Adapting to other perspectives of the IaC problem, like App Services and Inter Device application Communication (IDaC) will be a big step towards platform independent solution using Model Driven Engineering approach.

Figure iac Components

App – to – App communication or IaC means how two applications seamlessly collaborate with each other. As to why we focus the mobile environment, it is the nature of usage of mobile devices. Which is dynamic and usually involves navigation between three or four applications in a short space of time.

Platform independent solutions and the future improvements to the IaC problem will lead to better application communication and realization of the Web of Apps idea. Idealistic approach with the goal of adapting mobile applications to communicate with each other, like websites do on the web platform.

# Screenshots of Alarm application

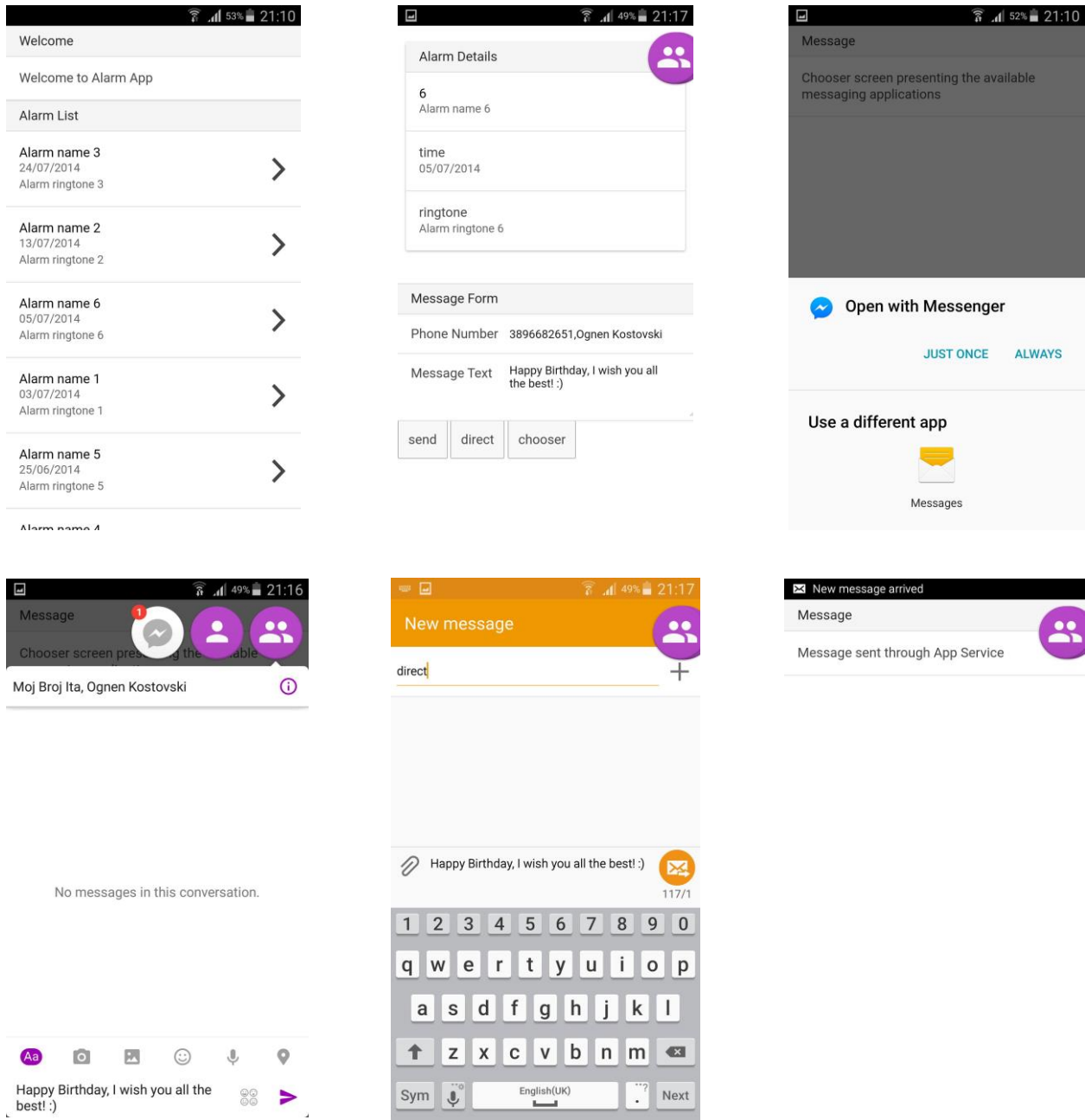


Figure 32 – 37 Alarm application Screenshots

## BIBLIOGRAPHY

- [1] Marco Brambilla, Jordi Cabot and Manuel Wimmer. Model Driven Software Engineering in Practice: Synthesis Lectures on Software Engineering. Morgan Kaufmann, 2013. 1, 53
- [2] Rapid mobile app and web application development platform.  
<https://www.webratio.com/site/content/en/home>. Accessed: 10.06.2016
- [3] Marco Brambilla and Piero Fraternali. *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*. Morgan Kaufmann, 2014. 18, 104
- [4] Web Ratio Mobile Platform.  
<https://www.webratio.com/site/content/en/mobile-app-development>. Accessed: 10.06.2016
- [5] Writing software patterns.  
<http://www.martinfowler.com/articles/writingPatterns.html>. Accessed: 10.06.2016
- [6] Android Intent System.  
<https://developer.android.com/training/basics/intents/index.html>. Accessed: 10.06.2016
- [7] Android Intent Resolution.  
<https://developer.android.com/training/app-links/index.html>. Accessed: 10.06.2016
- [8] Windows URL Schema mechanism.  
<https://blogs.windows.com/buildingapps/2015/09/22/using-cross-app-communication-to-make-apps-work-together-10-by-10/>. Accessed: 10.06.2016
- [9] Eeva Kangas and Timo Kinnunen. *Applying user-centered design to mobile application development*. *Commun. ACM*, 48(7):55-59, July 2005. 17
- [10] jimcoplien – gertrudandcope, Software Patterns.  
<https://sites.google.com/a/gertrudandcope.com/www/jimcoplien>. Accessed: 10.06.2016
- [11] Web Ratio Mobile Custom Operations.  
<https://www.webratio.com/learn/learningobject/mobile-custom-components-v-80?link=oln72ae.redirect&nav=14#!ajax=true>. Accessed: 10.06.2016
- [12] Apache Cordova Plugins.  
<http://cordova.apache.org/docs/en/latest/guide/hybrid/plugins/index.html>. Accessed: 10.06.2016

[13] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994. 101, 102

[14] Background tasks (App Services).

<https://msdn.microsoft.com/en-us/windows/uwp/launch-resume/support-your-app-with-background-tasks>. Accessed: 10.06.2016

[15] App Lifecycle – Keep Apps Alive with Background Tasks and Extended Execution.

<https://msdn.microsoft.com/en-us/magazine/mt590969.aspx>. Accessed: 10.06.2016

[16] Multi – Window support in mobile devices.

<https://developer.android.com/preview/features/multi-window.html>. Accessed: 10.06.2016

[17] iOS Inter app Communication.

<https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Inter-AppCommunication/Inter-AppCommunication.html>. Accessed: 10.06.2016

[18] Analyzing Inter-Application Communication in Android.

<https://people.eecs.berkeley.edu/~daw/papers/intents-mobisys11.pdf>. Accessed: 10.06.2016

[19] Communication in Mobile Applications.

<https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Inter-AppCommunication/Inter-AppCommunication.html>. Accessed: 10.06.2016