Adaptive Offloading for Infotainment Systems

Luis Lino Ferreira, Luis Miguel Pinho, Michele Albano, César Teixeira
CISTER Research Centre / INESC TEC
ISEP, Polytechnic Institute of Porto
R. Dr. António Bernardino de Almeida, 431
4249 – 015 Porto, Portugal
{llf, lmp, mialb, crdst}@isep.ipp.pt

Abstract

Infotainment applications in vehicles are currently supported both by the in-vehicle platform, as well as by user's smart devices, such as smartphones and tablets. More and more the user expects that there is a continuous service of applications inside or outside of the vehicle, provided in any of these devices (a simple but common example is hands-free mobile phone calls provided by the vehicle platform). With the increasing complexity of 'apps', it is necessary to support increasing levels of Quality of Service (QoS), with varying resource requirements. Users may want to start listening to music in the smartphone, or video in the tablet, being this application transparently 'moved' into the vehicle when it is started. This paper presents an adaptable offloading mechanism, following a service-oriented architecture pattern, which takes into account the QoS requirements of the applications being executed when making decisions.

1. Introduction

Infotainment "apps" are getting common in modern vehicles. More and more vehicles are equipped with advanced computational platforms that are capable of executing a multitude of applications, ranging from video reproduction to games, from satellite navigation to social networking. Although many applications can only be executed on the specialized hardware of the vehicle, it is nowadays common for the vehicle to include general purpose platforms, equivalent to the mobile devices carried by the passenger – mobile phones, tablets, wearable devices, etc.

It is increasingly common for the passengers to expect a transparent equivalence of apps, which start to be executed in one device, and then transparently "move" into the other. This is already the case for hands-free mobile communication, but will extend to all other domains: the driver will start planning a trip in a maps app in the smartphone, expecting that when starting the car, the vehicle's infotainment system continues the interaction; kids will expect that the movie which was being displayed in the table tablet is also screened in the vehicle's media system.

At the same time, although the performance and resources of mobile platforms is increasing in an unprecedented way, they are not at the level of common desktops or laptops. The usage of their resources (of which energy is always an issue) needs to be carefully managed and sharing in-vehicle resources is obviously interesting.

These different interleaving requirements hint into an increasingly dynamic environment, where different devices stop being the application providers, and should be seen as simple components partaking into providing an application to the user. The work behind this paper addresses this highly dynamic environment, focusing in a particular functionality to allow mobile devices to dynamically offload some of the applications' services [1 - 7] to the computational platform of the vehicle. In particular, we target applications that present variable Quality-of-Service (QoS) requirements, and that can benefit from the speed-up and increased quality provided by code offloading to computing node(s) with more resources (e.g. multimedia streaming and gaming).

Offloading techniques have advantages over traditional distributed computing: i) the code to execute is available in the client application; ii) the nodes to which computations are offloaded are nearer, consequently communications usually have less delays and better QoS; and iii) the changes required on the original code are usually less significant, and can be implemented as application layer solutions.

To support offloading several frameworks have been proposed, of which some examples for smartphones are Cuckoo [3], CloneCloud [4] and MAUI [5]. However, most

The paper was presented at APRES 2015.

of the existent frameworks do not support QoS requirements. Exceptions are for the case of offloading computation from mobile robots [1], or a generic offloading computation method for server scheduled real-time systems [10] and finally a generic framework to offload QoS-aware services in an Android environment [8,12].

The solution proposed in this paper is an evolution of the works proposed in [8, 12], addressing how offloading can be instantiated within a service-oriented architecture and detailing the interactions between components, such as the one proposed in the Arrowhead framework [12]. The next section provides an overview of the instantiated architecture of the framework, and then section 3 presents the interaction patterns between the system nodes to support the dynamic offload behavior.

2. System Architecture

To provide a solution which is possible in an ecosystem that includes multiple different types of devices, it is required to use a generic, interoperable, framework that allows the establishment of connection between services, looking up for service locations and, at the same time, granting the required levels of QoS, security and confidentiality. Although this paper does not deal with security and privacy issues, it is clear that any framework to be used in this environment requires these properties.

A possible solution is to base the system architecture on the Arrowhead framework [12], which has been proposed to support interconnected, cooperative systems. It is based on a SOA philosophy, interconnecting systems (in this case devices) that provide and consume services, and cooperate (systems of systems).

The framework also includes principles on how to design SOA-based systems, guidelines for its documentation and a software framework capable of supporting its implementations. The software framework includes a set of Core System elements which are capable of supporting the interaction between Application Services. The most important elements are the *Service Registry*, *Authorization* and *Orchestration*.

The *Service Registry* system is used to keep track of all producing services within the system. The *Authorization* system allows access to a list of access rules to other services. The *Orchestration* system contains rules which allow controlling how systems are deployed and how to interconnect them.

In this paper we propose to extend the *Orchestration* system in order to keep track on the utilization status of possible offloading nodes. In this way, it can decide to which nodes to offload the services, using as a simple rule the present and past utilization level of that node. More advanced rules can also be applied that take into account the

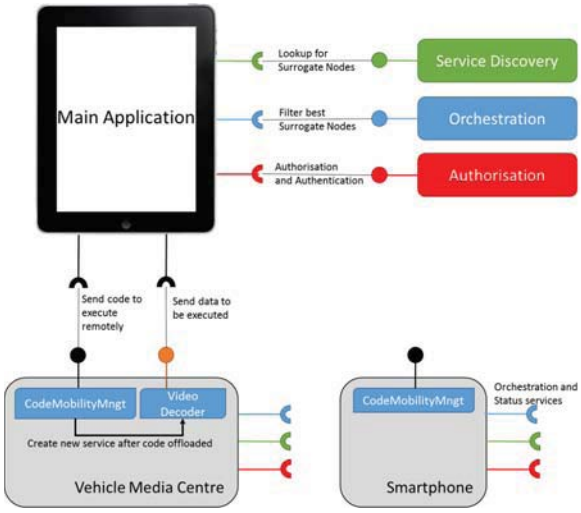

Figure 1 System Architecture

status of all nodes and use an optimization algorithm find an optimal (or close to the optimal) solution for the distribution of services among the system nodes [13].

Each node must also make available a service to determine its utilization status – the *Status* service in Figure 1. It will also need another service to support the mobility of code – the *CodeMobilityMngt* service. Figure 1 shows the main components of the architecture being proposed, where, for convenience only the most relevant connections are being shown. The figure shows a scenario where the streaming of a movie which is being played in the tablet is offloaded to the infotainment system, named as Vehicle Media Centre.

In this scenario all services register themselves with the *Service Registry* system and request the necessary authorizations to connect with the *Authorisation Service*. The *Orchestration* service is queried by the Tablet node whenever it predicts the need to offload some of its computations, as services. Since the *Orchestration* service knows the status of each node in the system it is able to choose the most underutilized node(s) to offload computations. Other kind of criteria can also be used, it simply depends on the parameters of the request made to the *Orchestration* service.

The *CodeMobilityMngt* service handles the transfer of code, state, of installing the mobile code on the destination node and executing the transferred code. After, it runs the offloaded code, which interfaces with the other components through the *VideoDecoder* service to which the main node must connect to. The information regarding this new end point is also transferred through a specific interface of the *CodeMobilityMngt* service.

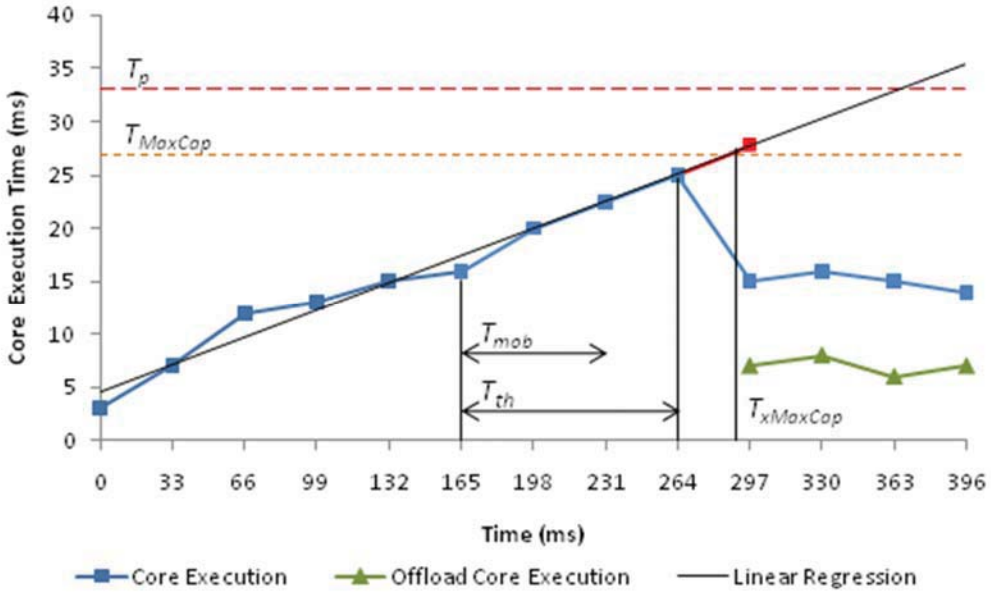

Figure 2 Offloading timeline [11]

3. Offloading of Mobile Services

The offloading mechanisms underlying this proposal are in part based on the QoS-aware code mobility framework proposed in [8], the architecture proposed in Section 2 and on the timing analysis proposed in [9].

The main objective of the mechanisms is to dynamically adapt to the varying execution times by offloading computation to surrogate nodes in a way that the user does not notice any disruption on the application behavior. To that purpose, the algorithm needs to predict the forthcoming core execution times, based on past execution times.

Figure 2 illustrates the operation in the time domain. The core services' execution time on the main and offloading device are represented by square and triangle marks, respectively. The continuous line, without marks, shows the linear regression that best approaches the evolution of the execution time on the original device. At any time it is possible to estimate the instant at which the core execution time will exceed the maximum capacity of the original device – $t_{xMaxCap}$. Obviously, to fulfill the operating objective of the application, no QoS disruption should occur, consequently the code mobility operation, which precludes the offloading procedure, should be completed prior to this time.

It is important to note that we are using a linear regression, since it is much simpler and light to make the necessary calculations, but any other kind of regression might be used

3.1. Services interaction

Figure 3 provides the sequence diagram of the service interaction, for the scenario in Figure 1, where a user with a Tablet, is executing a (resource hungry) application which is registered into the car infotainment systems, running the Arrowhead framework. The Tablet will require the execution of some of the services on one of the available nodes, chosen by the *Orchestration* system, based on status information related to each node.

The boot-up starts by all systems registering their services on the *Service Discovery* system. In such systems only registered services can establish connections.

After some execution time, the Tablet application makes an offloading decision (based on the principles described in Figure 2). It first starts by obtaining the address of the systems which offer the *CodeMobilityMngt* service. The Tablet then sends the set of addresses to the *Orchestration* service using the *filterServices(S, QoS)* service, specifying the set of nodes that can cooperate and the criteria to be used, i.e. the required Quality of Service.

The *Orchestration* system, which has been periodically collecting the status information regarding each node, is aware of the resource availability, therefore it is able to decide on a target node for the offloading application. The algorithm for this decision may be implementation

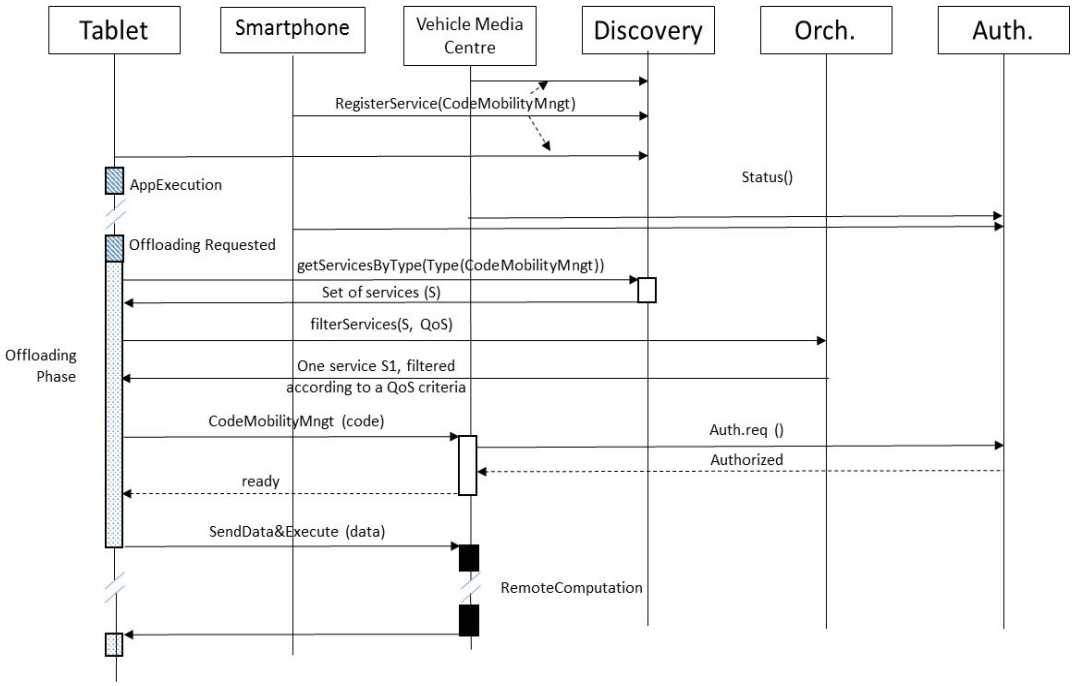

Figure 3 Services interaction

dependent and potentially user configurable. Priority may be given differently by users to multiple QoS dimensions, taking into consideration that supporting one new application may imply reducing the quality of other applications being executed [13].

The Tablet can now connect to the *CodeMobilityMngt* service of the Vehicle Media Centre, sending the code to execute (assuming the same OS is serving both nodes[2]). The Media Centre installs the code and executes the service, replying with an address to be used, specifically in this case. Note the authorization procedure which is implemented by the device to guarantee that the requesting node is authorized to do so.

The Tablet, now sends the data to be used on the remote computation, the Media Centre starts to execute the Video Decoder service (it could eventually execute only part of the application and return results which would be consumed by the requesting device). It is important to note that the offloading phase depicted in Figure 3 is artificially enlarged in the time domain, compared to the actual computation, for the system to be efficient.

4. Conclusions

This paper presents an architecture for dynamic offloading of application services in a vehicle environment. The goal is to support the transparent execution of application components from the users' mobile devices to the vehicle's infotainment platform. The architecture of the system is based on a service-oriented framework, intended for interoperability. This framework provides discovery, orchestration and authentication services, to which offloading services are added. These offloading services take into account the quality of service requirements of the applications, as well as the load in the system, in order to guarantee a continuous experience to users.

Acknowledgements

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project Ref. FCOMP-01-0124-FEDER-022701 (CISTER), by The Portuguese Agency for Innovation (ADI) under the ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project CarCoDe (ITEA2 Nr. 11037, QREN - SI I&DT Nr. 30345) and by FCT and the EU ARTEMIS JU funding, within project ref. ARTEMIS/0001/2012, JU grant nr. 332987 (ARROWHEAD).

[2] The same 'app' code can eventually already exist in the offloading node.

5. References

[1] Nimmagadda, Y., Kumar, K., Lu, Y., Lee, C.S.G., “Real-time moving object recognition and tracking using computation offloading”, IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), Oct. 2010, pp. 2449 – 2455.

[2] Sommer, S., Schola, A., Gapanova, I. , Knoll, A., Kemper, A., Buckl, C., Kainz, G, Heuer, J., Schmitt, A., “Service Migration Scenarios for Embedded Networks”, 2010 IEEE 24th Intl. Conf. on Advanced Information Networking and Applications Workshops, 2010, pp. 502 – 507.

[3] Kemp, R., Palmer, N., Kielmann, T., Bal, H., “Cuckoo: a Computation Offloading Framework for Smartphones”, Proc. of the 2nd Intl. Conf. on Mobile Computing, Applications, and Services, (MobiCASE '10), 2010.

[4] Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., Patti, A., “Clonecloud: Elastic execution between mobile device and cloud”, In EuroSys 2011, April 2011.

[5] Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., Bahl, P., “MAUI: making smartphones last longer with code offload”, Proc. of the 8th Intl. Conf. on Mobile systems, applications, and services (MobiSys '10). ACM, Jun. 2010, pp. 49 – 62.

[6] Gu, X., Nahrstedt, K., Messer, A., Greenberg, I., Milojicic, D., "Adaptive Offloading Inference for Delivering Applications in Pervasive Computing Environments", First IEEE International Conference on Pervasive Computing and Communications (PerCom'03), 2003, pp. 107 – 114.

[7] Xian, C., Lu, Y., Li, Z., "Adaptive computation offloading for energy conservation on battery-powered systems", International Conference on Parallel and Distributed Systems, 2007, vol. 2, Dec., 2007, pp. 1 – 8.

[8] Gonçalves, J. Ferreira, L. Pinho, N., Silva, G., “Handling Mobility on a QoS-Aware Service-based Framework for Mobile Systems”, Intl. Conf. on Embedded and Ubiquitous Computing (EUC 2010), Dec. 2010, pp.97 – 104.

[9] Ferreira, L., Nogueira, L., “On the Use of Code Mobility Mechanisms in Real-time Systems”, accepted for publication at the 10th Intl. Workshop on Real-Time Networks, Jul., 2011.

[10] Toma, A., Chen, J., "Computation offloading for frame-based real-time tasks with resource reservation servers", 25th IEEE Euromicro Conference on Real-Time Systems (ECRTS), Jul 2013, pp. 103 – 112.

[11] Ferreira, L., Silva, G., Pinho, L., “Service offloading in Adaptive Real-Time Systems”, 6th IEEE International Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE2011), Sep, 2011, pp 1-6. Toulouse, France.

[12] Blomstedt, F., et al, “The Arrowhead Approach for SOA Application Development and Documentation”, 40th Annual Conference of the IEEE Industrial Electronics Society (IECON 2014), Nov, 2014, pp 2631-2637, Dallas, U.S.A.

[13] Nogueira, L. and Pinho, L., “Time-bounded Distributed QoS-Aware Service Configuration in Heterogeneous Cooperative Environments”, Journal of Parallel and Distributed Computing, Elsevier. Jun 2009, Volume 69, Issue 6, pp 491-507.