

eSERL: Feature Interaction Management in Parlay/OSA Using Composition Constraints and Configuration Rules

Alessandro DE MARCO, Ferhat KHENDEK
*Department of Electrical and Computer Engineering
Concordia University
1455 de Maisonneuve Blvd. W., Montreal, QC, Canada
{a_demarc, khendek}@ece.concordia.ca*

Abstract. SERL is a language and framework for managing the triggering and execution of services in a single-user, single-network-component (SUSC) environment. We propose enhancements to SERL, dubbed eSERL, to allow for personalized customization of services by end-users who do not have expert knowledge of the services and of the environment, while guaranteeing, to a certain degree, that unwanted feature interactions will be avoided. SERL allows for such customization of service usage, but it does not consider the issue of providing the guarantee. Our approach involves validation of user-defined service configurations, specified in the form of rules, against constraints for service composition and interworking imposed by experts.

1. Introduction

Next-generation telecommunication networks will provide new and enhanced capabilities and enabling technologies for application-layer service development. This emerging infrastructure is often referred to as a *multimedia service network*, or simply, a *service network*. It combines the advantages of the IP and cellular phone technologies in a converged framework. In addition, it offers access to network capabilities and services through Open Service Architecture (OSA) gateways. 3GPP [1] OSA is based on the Parlay suite of high-level and standardized APIs [2]. Parlay/OSA enables the provisioning of services independently of the underlying network technologies, and facilitates more rapid service development [3, 4]. The “open” paradigm allows for new players in the service network architecture, new streams of revenue for network operators, and new business opportunities for 3rd party service providers or developers with innovative services to offer. Alas, it also presents a number of technological challenges that must be dealt with before such a model can be realized, not least of which is the Feature Interaction Problem.

The Feature Interaction Problem [5] exists in current telephony systems, but it is expected that the problem will be severely aggravated in next-generation systems due to the openness and distributed nature of the architectures, and the new types of services that will be developed [6]. Another factor to consider in that context is that non-expert users will be provided with means to customize service behaviour to an unprecedented degree, or even develop and deploy their own services. Consequently, powerful mediators or control mechanisms must be implemented, possibly at more than one point in the architecture, in order to avoid, or detect and resolve unwanted, erroneous, or malicious service network usage.

In this paper, we describe a mediation system in a Parlay/OSA context, which provides customizable, yet controlled end-user access to service network capabilities and usage of high-level services without imposing unwarranted restrictions that would obviate the benefits of the functionality offered. The system is based on the Service Execution Rule Language and Framework (SERL) [7, 8, 9]. We propose and describe extensions to the original framework. Our contributions reported in this paper involve language extensions to support the definition of service composition constraints by experts, and a validation scheme to ensure that user-defined service configurations do not violate these constraints.

This paper is organized as follows. In Section 2, we briefly introduce SERL. In Section 3, we describe our approach. We then explain our validation scheme in Section 4. Finally, in Section 5 we conclude by hinting at potential applications and our future work. Notice that we often use the terms *feature* and *service* interchangeably. Similarly, we often use the term *call* to refer to the concept of a *session*, which is more general than a typical telephone call today.

2. SERL: Service Execution Rule Language

SERL [7, 8, 9] is a language and a framework for managing the triggering and execution of services. It is based on *condition-action* rules, and a processing model involving interception of events, matching of event conditions to rule triggering properties, and then application of matched rules. SERL was originally developed for SIP, where deployment of rule processing engines implementing the model is more amenable to Proxy Servers, but not unimaginable in User Agents. The XML-based language is flexible enough to support other technologies, such as Parlay/OSA, and even certain heterogeneous environments.

SERL rules are grouped into Rule Modules, each with one owner. The owner of a Rule Module is typically a subscriber to services affected by the rules in the module. Services are classified into groups according to their behavior, and each group is assigned a Processing-Point identifier. Processing Points refer to the points in the call-signaling timeline where services within a certain group may be invoked. Services are triggered in response to events flowing downstream (i.e. requests) or upstream (i.e. responses) through a node. Events usually originate from the network, or from services. When triggering rules are encoded, they take on the same Processing-Point identifiers as the services they relate to. A rule-search algorithm, set to run upon the occurrence of events, takes into consideration Processing-Point identifiers, priority of rules, as well as the relevant event information (a.k.a. event context) when searching for matching rule conditions. Rule actions may involve delaying, overriding, canceling or generating events.

In addition to managing the execution and triggering of services, SERL may access system capabilities like a database, a Presence server, a Location server, Web services, etc. Examples demonstrating such functionality are not provided in the Internet-Drafts, however such functionality is implied. It is the responsibility of the developer of the SERL engine to build-in adaptors for communication with such network capabilities or distributed services.

SERL is not a language to describe the behavior of services, nor is it intended to be used to detect feature interactions. Rather, it is a mechanism to allow for the application of feature interaction resolution policies in a SUSC [5] context. It is assumed that potential interactions are known *a priori*, and knowledge about how to resolve interactions is encoded in rules. Ordering through Processing-Points and priorities are the only means available to enforce

resolutions. For instance, when several rules are matched, SERL does not define a resolution policy other than a simple ordering scheme based on priorities defined *a priori*.

3. Managing FI in a Parlay/OSA context with eSERL

In this section, we discuss our architecture in a Parlay/OSA context, and our enhancements to SERL to allow for personalized customization of services by end-users who do not have expert knowledge of the services and the environment, while guaranteeing, to a certain degree, the absence of unwanted feature interactions. We assume that any user may attempt to configure any service they subscribe to, or compose and inter-work several of them for added-value. By validating user-defined configurations against constraints imposed by experts, we are able to provide a guarantee that user requirements will be met.

3.1. Rationale and Related Work

Our goal is to develop a generic framework for service composition in which we allow for personalization, yet avoid unwanted feature interactions. We wish to impose no specific application-domain, even though we have focused on a Parlay/OSA context initially. We also aim to provide for a clear relationship between user-defined requirements and services or capabilities available in the network. We believe that by achieving the latter goal, we will facilitate the development of feasible billing and pricing strategies. Our vision is that the next-generation “killer app” is probably not going to be a single service, but the capability for an individual user to compose and personalize a multitude of services according to his own requirements.

With regard to related work, we are aware of two initiatives, namely CPL [10] and ACCENT/PDL [11]. CPL enables end-user service creation and guarantees the absence of feature interactions due to the nature of the language. On the other hand, it is not flexible enough to support a behavior that is not within the realm of Call Processing. ACCENT/PDL is a newer initiative, which is more flexible. End-users specify policies for service behaviors in a constrained natural language format. The goal is to allow comparison of policies online and/or offline in attempts to detect policy conflicts. However, ACCENT/PDL seems to lack a clear mapping between policies and features or network capabilities.

3.2. Overall Architecture

Our approach requires an architecture employing one or more Feature Interaction Managers (FIM), which act as mediators controlling the triggering and execution of features. The behavior of a FIM in our context differs from the commonly understood behavior(s) documented in [12]. Each of our FIMs “virtually” composes services according to user-defined requirements (i.e. rules). In other words, each FIM manages the events that affect the behavior of any number of deployed services, and this may lead to the overall appearance of composed service behavior from the user point of view. SERL is a technology that essentially implements a SUSC subset of the generalized model, and therefore we have selected it as the starting point for our project. Moreover, since Parlay/OSA defines an architecture with clearly specified points of control (i.e. application servers, service capability servers), we intuitively position the FIM(s) in a Parlay/OSA framework as shown in Figure 1.

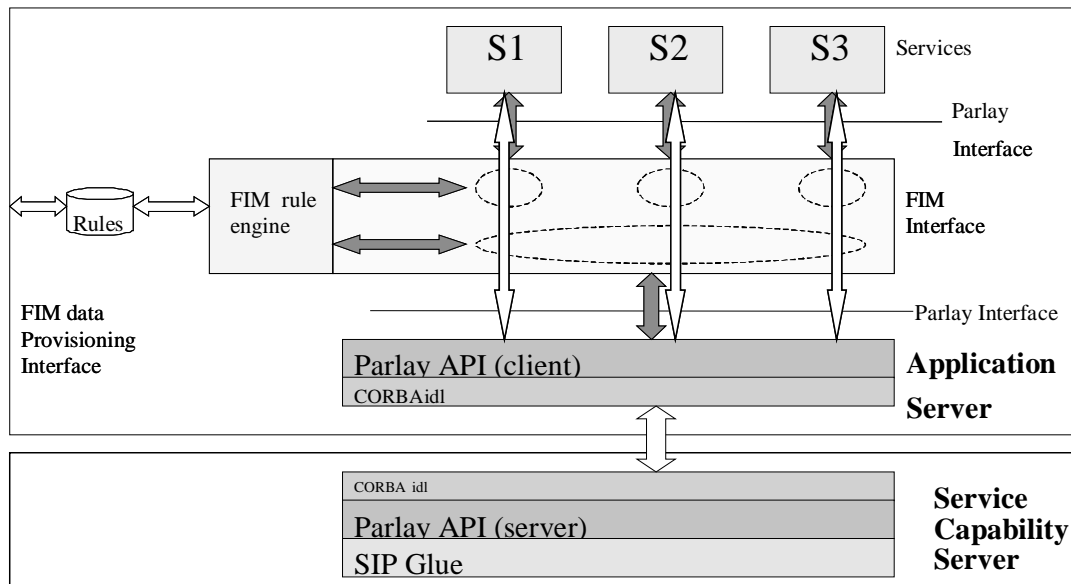


Figure 1. FIM for one Application Server

We take advantage of the standardized Parlay/OSA APIs, and the Half-Object Plus Protocol design pattern [13]. The FIM is inserted between services and the actual Parlay/OSA client side interface implementation. The FIM offers a virtual implementation of the Parlay/OSA APIs to services, and it uses the actual APIs provided by the Application Server and the Service Capability Server to implement the interface offered to services. The idea is to enable transparency of FIM behavior from the service point-of-view. A service developer should be able to develop a Parlay/OSA service and run it on an Application Server, whether a FIM is present or not. With the FIM positioned as such, it is able to intercept all events to or from services that are deployed on the Application Server.

3.3. eSERL: Enhanced SERL

We require each FIM in the architecture to implement our enhanced version of SERL (eSERL). As a primary enhancement, we define two types of SERL Rule Modules: Composition Constraint Rule Modules, and Configuration Rule Modules. Composition Constraints represent expert knowledge about how services may inter-work or be composed. Configuration Rules are user-defined and relate to instances of *acceptable compositions* of services (i.e. not in violation of constraints) along with personalized service data.

3.4. Composition Constraints

Since we are considering a SUSC context, it is reasonable to suggest that all possible compositions of services in the system are known *a priori*, which allows offline analysis of services to detect potential interactions. Moreover, an expert may use his or her knowledge, based on experience, to facilitate detection. The analysis procedure used by experts to detect interactions is out of the scope of this paper. Following analysis, the expert will have the required information to define constraints.

We highlight the fact that the set of possible compositions of services in the system may be reduced simply because we are dealing with SERL, involving service invocation according to

Processing Points and priorities. Intuitively, this may eliminate some potentially disruptive feature interactions. We define such constraints imposed by SERL as *implicit constraints* because they are intrinsic to the framework.

Composition Constraint Rule Modules express *explicit constraints*. To write these rules, an expert must consider the behavior of the services, their input/output data, and implicit constraints of SERL. Explicit constraints extend the service inter-working and composition protocol defined by SERL on a per system basis (i.e. single network component). We require that when services are deployed in the system, service providers will include *deployment descriptors* for each service. Contained within a deployment descriptor are items of information about the service that will be needed for feature interaction detection. If source code for services being deployed is available, then deployment descriptors may not be necessary.

Explicit constraints are classified into three types: *order-preserving*, *data selection*, and *mutual exclusion*. In the absence of a constraint, services may be invoked in parallel. We have extended the SERL language to be able to express these types of constraints and to be able to describe the service objects that will be invoked.

Composition Constraints usually express constraints for service composition and inter-working pair-wise, but triples, quads, etc., are allowed. This does not imply that services (assumed to be atomic) cannot be interleaved, so long as all constraints are satisfied for the duration of an event-flow leading to service invocations at the node, referred to a Cascaded-Chain in [7].

3.5. Configuration Rules

Configuration Rules specify conditions and actions to carry out when conditions are satisfied. They extend the run-time behavior of the call processing system. As such, a Configuration Rule Module may be seen as a meta-service or more abstract service layer. Existing language constructs allow end-users to write Configuration Rules, and as such, these rule sets would be compatible with SERL processing nodes not implementing the enhancements that we propose. On the other hand, we must constrain the SERL language in order to allow for validation. Hence, generic SERL Rules Modules would not be compatible with a SERL processing node expecting more specialized Configuration Rule Modules.

A user may have more than one Configuration Rule Module defined; however for simplicity, we allow for only one *activated* module at any given time. Also, we assume that the services considered in a module are actually in service while the module is active. Otherwise, it would not be eligible for activation.

3.6. Modified Feature Grouping Criteria

The final enhancement to SERL that we have devised is a modification to the criteria for determining the service groups and Processing Point identifiers. As defined, SERL provisions for this type of enhancement. Our modifications are summarized as follows:

- For services that may add, delete, or modify any part of the event context (e.g. SIP message, Parlay/OSA event), and are mostly related to routing, we assign Processing Point 1/-1. Such services may modify source, destination, or intermediary nodes identified in the event context, for instance.

- For services that may add, delete, or modify any part of the event context except routing information, and are mostly related to screening, we assign Processing Point 2/-2. Such services require read-only access to routing information, and are only allowed to block a call based on certain screening criteria. They may not re-route calls.
- For services that may add, delete, or modify any part of the event context except routing information, and are mostly related to the event context payload, we assign Processing Point 3/-3. Such services require read-only access to routing information, and may only modify event context payload (e.g. SIP message body).
- For services that may not add, delete, or modify any part of the event context, we assign Processing Point 4/-4. Such services require read-only access to event context.

4. Validation of Configuration Rules

Our approach for Feature Interaction Management hinges on the concept of validating user-defined Configuration Rules. The purpose of validation is to guarantee, to a certain degree, that user requirements (expressed as rules) can and will be satisfied.

4.1. Determining Acceptable Compositions

The set of acceptable compositions of services deployed in a system is generated automatically from Composition Constraints. Once acceptable compositions are calculated, they are stored in the system for future reference. A basic algorithm to achieve this involves enumeration and elimination of possibilities.

We assume that the set of Composition Constraints is complete. However, in theory, this assumption cannot be guaranteed. We rely on experts to know about possible problematic interactions between services and consider them when defining constraints. As more problematic interactions are discovered through service usage over an extended period of time, Composition Constraints will be updated, and acceptable compositions recalculated. However, this process of updating constraints may invalidate previously valid configurations. Solutions to this type of non-monotonic extension of the system will be explored in future work.

Inconsistency of Composition Constraints, an important issue, may, in the worst case, lead to a set of acceptable compositions where each composition is a service on its own. In such a scenario, an expert would intuitively try to detect inconsistencies and relax constraints when possible.

4.2. Validation of Configuration Rules

The algorithm to validate a user's Configuration Rule Module essentially tries to determine all potential service compositions from the user's set of rules, and then makes sure that each composition is acceptable. The issue requires us to examine the possibility of having separate rules with triggering conditional expressions satisfied simultaneously, thus causing their actions to be composed. In other words, if it is possible for a single event occurrence to satisfy conditions of two or more rules, we say that the rules *overlap* and assume that the services affected by the actions constitute a composition.

This problem has been studied in a different context in [14] where actions have well-known semantics (e.g. discard or forward packets). In our case the actions are unknown, but constrained by the Composition Constraints. If a set of actions is not forbidden by the Composition Constraints, we say that these actions will lead to a non-conflicting service composition.

Determining whether conditional expressions overlap has a solution in polynomial time as long as the set of possible values for variables is discrete, finite, and ordered as shown in [14]. Due to the usage of SERL language constructs in a Parlay/OSA context, this holds, but we also define a general principle.

General Principle: Two rules are said to be overlapping, unless

- (a) conditional expressions have at least one common dimension, AND
- (b) at least one common variable in the common dimensions, AND
- (c) non-overlapping values for the common variables.

Notice that dimensions can be seen as Parlay/OSA APIs, or other discrete, finite, and ordered quantities.

5. Conclusion

Our approach for FI management when composing and personalizing services in a SUSC context is quite general. We expect to be able to apply it in any domain where the set of possible values for event variables (in one or more dimensions) is discrete, finite, and ordered. We have a clear relationship between user-defined rules and services or network capabilities. From a commercial point of view, a user may, for instance, subscribe to three traditional services and a Service Composition Manager (SCM) based on our approach. The user would then define rules to manage the three services as a composed set rather than individually, and deploy them into the SCM. This will lead to overall service behaviour tailored to user requirements, thus creating value that would potentially be much greater than the sum of the values created by the individual, independently configured services. Guaranteeing that a user's requirement can and will be met is of utmost importance in order to ensure customer satisfaction and justify the SCM subscription cost.

We view our contribution as a foundation for our future work towards a solution in a multi-user, distributed environment. This involves new architectures, enhanced algorithms, and accompanying tools. Moreover, we will explore Activation Rules for activating particular Configuration Rule Modules based on user-defined requirements. We are also planning to develop a framework to enable 3rd party packaging of services and theme-based rule writing wizards. Users would purchase or subscribe to such packages, for instance a "Family Package", "Driver's Package", or "World Traveller's Package". We believe that by having 3rd parties with intermediate-level expertise write theme-based wizards, users will be able to specify requirements in a controlled manner, and greatly improve the rate of "first-try" Configuration Rule Module acceptance.

Acknowledgements

We would like to acknowledge the participation and contributions of Roch Glitho, André Poulin, and Kindy Sylla from Ericsson Research Canada, during the initial phase of this project. We also acknowledge funding from les Fonds NATEQ (Québec) and Concordia University.

References

- [1] Third Generation Partnership Project at <http://www.3GPP.org>
- [2] Parlay API Specifications at <http://www.parlay.org>
- [3] Ard-Jan Moerdijk and Lucas Klostermann, "Opening The Networks With Parlay / OSA APIs: Standards and Aspects Behind The APIs", at <http://www.parlay.org/specs/library>.
- [4] Roch H. Glitho and Kindy Sylla, "Developing Portable Applications for Internet Telephony: An Overview of Parlay and a Case Study on its Use in SIP Networks", submitted to IEEE Network Magazine, 2002.
- [5] E. J. Cameron, N. D. Griffeth, Y.-J. Lin, M. E. Nilson, W. K. Schnure and H. Velthuijsen, "A Feature Interaction Benchmark for IN and Beyond", in *Feature Interactions in Telecommunications Systems*, IOS Press, Amsterdam, pp. 1-23, 1994.
- [6] J. Lennox and H. Schulzrinne, "Feature Interaction in Internet Telephony", 6th Workshop on Feature Interactions in Telecom and Software Systems, Scotland, June 2000.
- [7] R.W. Steinfeldt and H. Smith, "SIP Service Execution Rule Language: Framework and Requirements", (work in progress), IETF Internet-Draft: draft-steenfeldt-sip-serl-fwr-00.txt, May 7, 2001.
- [8] H. Smith and R.W. Steinfeldt, "SERL Examples with SIP and SDP", (work in progress), IETF Internet-Draft: draft-smith-serl-ex-00.txt, May 7, 2001.
- [9] R.W. Steinfeldt and H. Smith, "Service Execution Rule Language (SERL 1.0) for SIP", (work in progress), IETF Internet-Draft: draft-steenfeldt-sip-serl-00.txt, May 21, 2001.
- [10] J. Lennox, and H. Schulzrinne, "Call Processing Language Framework and Requirements", IETF RFC, <http://www.ietf.org/rfc/rfc2824.txt>.
- [11] ACCENT Project at <http://www.cs.stir.ac.uk/~kjt/research/accent.html>
- [12] M. Calder, E. Magill, M. Kolberg, and S. Reiff-Marganiec. "Feature Interaction: A Critical Review and Considered Forecast." *Computer Networks*, Volume 41/1, pp. 115-141, North-Holland. January 2003.
- [13] Gerard Meszaros, "Half Object Plus Protocol", in *Pattern Languages of Program Design*, Vol. 1, James O. Coplien, Douglas C. Schmidt, eds. Addison-Wesley, 1995.
- [14] Dong Wang, Ruibing Hao, David Lee. "Fault Detection in Rule-Based Software Systems", Concordia Prestigious Workshop on Communication Software Engineering, Montréal, Canada, Sept. 2001. Extended version to appear in the International Journal of Information and Software Technology, Elsevier, 2003.