The SOCRATES SON-Function Coordination Concept

Tobias Bandh

2011-11-06

Abstract

The SOCRATES FP7 research project provided fundamental research in the area of Self-Organizing Networks (SONs), including a concept on SON-Function coordination. The presented coordination concept contained a detailed description of a coordination architecture, required components and their functionalities. This document combines the information from multiple sources as for example publications and deliverables in order to provide a detailed and holistic overview followed by an analysis of the SOCRATES SON-Function coordination concept.

1 Introduction

SOCRATES (Self-Optimisation and self-ConfiguRATion in wirelEss networkS) [3] was a research projects with multinational partners from industry and academia funded by the European Union under the 7^{th} Framework Program over the course of 3 years (2008-2010). The project aimed to enhance the operation of wireless access networks through the introduction of self-organization methods into network planning, configuration and optimization. The project presented its results in a number of scientific publications and project deliverables which are available from the project website. The project covered a broad range of topics but with a clear focus on the integration of all self-organization methods into a single framework. This holistic view on SON and the is the most important contribution.

Although targeting radio access networks in general the 3GPP LTE radio interface was chosen as a reference radio interface to serve as a basis for all studies that were performed within the project.

For us, the work done within SOCRATES is of special interest as it, for the first time, identified the need for a coordination of SON-Function execution in order to prevent conflicting behavior between SON-Function instances with negative effects on the network operation. Based on this finding the project developed a coordination concept including an architecture consisting of multiple building blocks.

An overview over the SOCRATES coordination concept and the functional entities that are envisaged to provide the functionality for a successful coordination is provided within the next sections.

After a first overview of the coordination concept in Section 2, detailed information about the SON-Functions within SOCRATES in Section 2.1, the usage and definition of the term policy (Section 2.2) and the

key components of the coordination architecture in Section 2.3 is given. Section 2.4 shows the coordination process. A final assessment of the SOCRATES SON-Function coordination concept is given in Section 2.5.

2 SOCRATES Coordination Concept

Ensuring a joint operation of individual SON-Function towards a common goal has been identified as the main challenge for a SON Coordinator Framework. In SOCRATES the common goal which determines the SON-Function operation is specified via the operators' high-level objectives for the network operation.

To achieve the operation of individual SON-Functions towards a common goal, the coordination framework is used to harmonize the operation of the SON-Function instances and detect and avoid undesirable behavior caused by the performed configuration changes.

SOCRATES introduces two types of harmonization that are used to enforce behavior according to high-level objectives.

- **Heading Harmonization:** Is described as a way to **avoid** conflicts. It is comparable to a design-time SON-Function co-design, therefore it is limited in the same way as the co-design as soon as the number of deployed SON-Function increases.
- **Tailing Harmonization:** Is a run-time conflict **resolution** method for all conflicts that could not be prevented through heading harmonization. It evaluates any configuration change request of the SON-Function instances with respect to the operator objectives.

The interesting part of the SOCRATES coordination concept for us, is the tailing harmonization which is performed through a set of functional entities which are described in detail in Section 2.3. The decisions that are taken are influenced through operator guidelines which are represented in so called policies which are introduced an analyzed in Section 2.2.

2.1 SON-Functions in SOCRATES

A small set of examples from the NGMN listing of SON usecases [2] has been selected as a basis for the SON-Functions within SOCRATES. Overall nine use cases have been selected and serve as representative SON-Functions on which the SOCRATES cases studies are based on [4]. In the project deliverables detailed information is given about those particular SON-Functions their implementation and their behavior, but a generic conceptual description of SON-Functions and SON-Function behavior within SOCRATES is missing. Since it is not available there is also no description of a common way of SON-Function operation which limits the functionality of the SON Coordinator to a coordination of the configuration changes requested by the function instances.

2.2 Policies in SOCRATES

Within the SOCRATES project the term policy is used somehow differently to an often used technical understanding of policies, although at a first glance there are many similarities to the policy concept introduced in [5]. Especially the specialization of high level policies into more specific policies has a strong similarity to the definition of the policy continuum

introduced in [5]. A good overview on the SOCRATES understanding of policies is given in [1].

A detailed analysis of the policies reveals the differences between the SOCRATES policies and the common understanding of policies in the area of network management. It especially brings up the question whether either using the term policy does not cover the semantics of the SOCRATES concepts sufficiently or if the refinement of the policies just stopped at a very high level and the lower levels of policy refinement are not treated within SOCRATES.

SOCRATES uses four different types of policies:

- **Operator Policy:** This policy introduced as the most generic policy and refers to high level network performance objectives or operational guideline or requirements. Operators use this type of policy to specify the expected behavior of the network and the SON-Functions. The Operator Policy could also be called a general guideline for the network operation, including high-level specifications of performance goals.
- **Cell specific Policy:** is a specialization of some aspects of the Operator Policy which specifies the expected results for a set of performance metrics of a cell. For example the average cell edge throughput.
- **SON-Function specific Policy:** According to [1] a SON-Function specific Policy is derived from a Cell Specific Policy to govern to decision logic within a SON-Function. Looking at the functionality that should be achieved, the function specific policy can be seen as a configuration or parameterization of the SON-Function algorithm of a specific SON-Function instance. Such a parameterization is done according to current state and the expected performance of the targeted cell.
- **SON-Coordinator specific Policy:** The behavior of the conflict resolution process within the SON-Coordinator is determined by this type of policies. Coordinator specific policies are used to represent the decision logic that should be applied whenever a conflict is treated. In particular it is introduced as a method to enforce operator specific limitations on the configurations performed by the SON-Function instances. In comparison to the other policy types, some of the SON-Coordinator specific Policies are the only policies within SOCRATES that show typical characteristics of high-level policies. Some of the functionality that is subsumed under the term SON-Coordinator specific Policy does not show the characteristic of a policy but resembles more the configurations of policies.

As shown above, SOCRATES uses a very wide definition of a policy, which is mostly not aligned with the common understanding of what a technical policy is. In a very wide context the SOCRATES policies could be seen as some kind of abstract high-level non-technical policies that have to undergo a detailed refinement until they can be applied in an actual system. The analysis of the policies used within the SOCRATES coordination concept showed that for the further analysis and understanding of the coordination concept it is important to take the special semantics of the term policy into account.

2.3 Key Components of the SOCRATES Coordination Architecture

The SOCRATES coordination concept is realized through a set of functional entities, which are called roles within the SOCRATES Coordination Framework. In order to avoid the ambiguity with the term role known from for example the area of policy based network management where a role denotes a way of grouping subjects for policies, the term functional entity will be used instead. The combination of these functional entities is used to contribute to both heading and tailing harmonization. The entities and their functionality are introduced in detail within the following sections.

2.3.1 Policy Function

The Policy function is used to realize the SOCRATES specific refinement of the generic Operator into the SON-Function, Cell and SON Coordinator specific policies.

It takes the high level operator policies and transforms them into a configuration for the SON-Functions, cell specific versions of the performance targets, and configurations respectively coordination logic to guide the coordination decisions for the change requests of the SON-Function instances.

With the refinement of the Operator policies into SON-Function specific policies the configuration of the SON-Functions is adapted in a way that their behavior is aligned towards the common goal. Operator policies limited for example parameter ranges or maximum step sizes that are allowed for configuration requests.

In addition to the SON-Functions, the Policy function configures also all other functional entities that contribute to the tailing harmonization, by providing Cell and SON Coordinator specific policies. By supplying all functional entities with specific input the Policy function contributes to heading and tailing harmonization.

Due to the overall conceptual character of the SOCRATES SON Coordination framework, no detailed information on the realization of the Policy Function is given. [1] refers only to the required detailed understanding of the network and operational experience and an initially high degree of manual intervention.

The somehow diffuse or unusual usage of the term policy together with a very abstract specification of the Policy Function for the policy refinement leaves a lot of open questions and thus doubts about the applicability within a highly automated SON environment. For a further assessment of the practicability of such a functional entity a proof of concept implementation would be required which would bring a clear definition of the policies.

2.3.2 Alignment Function

For the coordination of SON-Functions the versatile Alignment Function plays a central role. The tasks performed by the Alignment Function are distributed between its two sub-functions called activation and arbitration.

Arbitration Conflict resolution an detection is performed by arbitration. It decides whether configuration change requests of SON-Functions

contribute to the same goal or are contradicting. This result evaluation is either done with predictions provided by the SON-Functions or performed by arbitration itself. The results of the predictions are then compared to the Cell-level policies. In addition to reject or acknowledge configuration changes arbitration has the possibility to adapt change requests to resolve the conflicts.

Arbitration can also be used for conflict prevention. Firstly, it implements a locking mechanism, that can be used by SON-Functions to lock parameters or performance measurements so that they are not changed or affected by other SON-Function instances. If measurement values are locked, arbitration function needs to know which measurements are affected by which configuration parameters to enforce the requested locks. If the SON-Functions locks configuration parameters to be able to monitor the effects of performed changes the SON-Function that requests the locks needs to know which other parameters can have effects on the monitored performance measurement.

The second way to prevent conflicts like oscillating reconfiguration is provided through a subscription mechanism. SON-Functions can subscribe to to-be-enforced configuration changes. When a configuration parameter change request for a subscribed parameter is received the subscriber is informed about the pending change. In case any SON-Function considers this change to be conflicting it will notify the arbitration function about the conflict, which will then resolve the issue.

Activation Activation, the second module, also has two responsibilities. On the one hand the activation of required SON-Functions. For example configuration and optimization functions whenever they are required. On the other hand it is responsible to react to undesirable behavior that has been detected by the guard function. Activation uses the knowledge about previously acknowledged configuration changes to track down the root cause for the detected undesired behavior. This is done either by triggering respective SON-Functions or by influencing the behavior of the system. It has the ability to either influence the changes that are performed by reconfiguring the the behavior of arbitration or directly the SON-Functions.

Assessment of the Alignment Function When looking at the functionality provided by the Alignment Function it becomes obvious that it performs most of the tasks required for a successful coordination. But it becomes also obvious that the tasks that are performed are very complex and require a lot of knowledge and understanding of the network. Especially the estimation of the expected effects of a configuration change is very complex. This is usually done within the SON-Functions in order to determine appropriate changes but in this setup this functionality has also to be present within the Alignment-function. Root cause analysis and prevention of similar problems in the future itself is a very complex task, which is added as part of the Alignment function. To be able to perform this step a very good knowledge about the explicit and implicit interactions between function instances is required. If this information is available already at design-time it would be smarter to use it for co-design and designing the decision logic. Being able to detect root causes without this knowledge is almost impossible. There are more parts where functionality is potentially replicated for example in the locking mechanism, for some functions the arbitration function needs to know the targeted

performance measurements. The other possibility is to include knowledge about all configuration parameters that could affect the targeted performance measurements into a SON-Function even if it does not operate on them.

This replication of knowledge and functionality strongly increases the complexity with the introduction of any new SON-Function. It is hard to asses if the alignment-function will still be maintainable and scalable in the presence of potentially thousands of SON-Functions deployed in the network.

2.3.3 Guard Function

Based on the assumption that with a large number of deployed SON-Functions undesired behavior cannot be avoided the guard function is used to detect such behavior. It operates independently from other parts of the coordination framework and performs the detection based on input from the operator policies. It bases its analyses purely on performance measurements and has no further information about running SON-Functions or performed changes. If undesired network behavior is detected the guard function will notify the alignment function, providing information that simplifies the root cause analysis, as for example affected metrics and the characteristic of the detected behavior.

To be able to perform its tasks, the guard function needs access to a large amounts of performance measurement data which needs to be mapped to the operator policies in general respectively the cell specific policies in particular. Within the published documents of the SOCRATES project there is no information on the mode of operation of the guard function, whether there are particular monitoring areas or if constantly performance data from the complete network is analyzed. In case of a constant complete monitoring of the available performance data it might be hard to provide a scalable approach with a very fast detection of undesired behavior due to the vast amount of data that needs to be analyzed and put into context.

2.3.4 Autognostic Function

The data required by all functional entities of the coordination framework and the SON-Functions is provided through the Autognostic function. Functions can request data, which is then collected, preprocessed and provided in the requested format. The Autognostic function accesses different data sources and can perform complex aggregation and processing steps.

Serving all data requests from a single source is seen as a way to ensure data consistency and a common view on the network by all active entities. The operational overhead of simultaneously providing data to a potentially very large number of entities however is not discussed within the available documentation. Therefore no statement on the practicability and scalability of this approach can be given, but with a large number of deployed SON-Functions with specific information requirements a very high number of individual data requests that need to be served must be expected.

2.3.5 Assessment Architecture of the SOCRATES Coordination Framework

The SOCRATES Coordination Framework consists of several building blocks which jointly contribute to coordinate the execution of SON-Functions and to prevent and resolve conflicting behavior. The tasks that are performed by the building blocks and also the SON-Functions are not always clearly separated. This results in an unnecessary replication of functionality which increases the complexity and requires an alignment of the functionality of multiple functional entities, as shown for the Alignment Function.

The alignment should be partly reached through a common configuration via the so called policies but partly also has to be done separately for each functional entity. Especially the result estimation has to be implemented in both the SON-Functions and the Alignment function separately. With each SON-Function that is newly introduced or changed at run-time due to newly detected dependencies, the Alignment function has to be adapted which seems hardly practicable.

Although the Autognostic function makes sense from a conceptual point of view, it is unproven that it is scalable with potentially thousands of SON-Functions requesting an even larger number of data. SOCRATES neither provided an implementation of the Autognostic function nor did a simulation on the load caused when serving many requests.

2.4 Coordination Process

From the published documents it is very hard to derive an actual coordination process which shows which tasks are performed in which functional entity of the SON Coordination framework when a SON-Function requests a configuration change. The case studies which are for example shown in [1] give an example of the coordination of two conflicting functions. But due to simplicity reasons the Guard function is not implemented at all and the functionality of the Autognostic function is integrated directly into the SON-Functions. The same applies to the required policies for the Alignment function. Therefore the policy function is also not needed and not provided. It seems as if the only part of the coordination framework that has been implemented is a small fraction of the functionality of the Alignment function, which evaluates the historical load data of a cell before allowing configuration changes.

From the description of the Coordination framework and its components the following can be derived:

- A SON-Function is executed and requests a configuration parameter change
- The alignment function evaluates the request towards potential conflicts with operator settings like step sizes.
- The alignment function evaluates the request with respect to the anticipated effects, based on the cell specific policies. If the estimated effects do not contribute the operator defined goals the request is rejected.
- The last step to detect potential conflicts is to inform all SON-Functions that are subscribed to the targeted configuration parameters. If any of the functions objects the intended configuration change a notification is sent to the arbitration function, which can

also contain a new proposal for a configuration change. All configuration change request are then aligned and harmonized by the arbitration function.

- Feedback about the coordination decision is sent to the SON-Functions.
- Undesired behavior due to erroneous effect assessment is detected by the guard function.
- The activation part of the alignment function is used to track down the root cause of the observed behavior and trigger countermeasures. Either directly via arbitration or through additionally activated SON-Functions.
- Supporting SON-Functions as for example configuration and optimization functions can be triggered whenever required by the activation part of the alignment function.

Since the coordination process above is not described explicitly in any of the SOCRATES deliverables or publications but derived from the available information there are still some open questions.

In the same way as functionality an information is replicated among the functional building blocks of the Coordination Framework the same tasks are performed multiple times within the coordination process. For example the assessment of effects of the requested changes. A SON-Function instance needs to perform such an assessment for the computation of the parameter change request. Which is subsequently done a second time within the Alignment function. Only for the detection of potential conflicts with the cell specific policies. Conflicts with other SON-Functions are treated via the subscriptions of other SON-Functions to changes of the respective control parameters. In case of conflicts with the requirements of other SON-Functions it is done a third time in order to harmonize the change requests of all SON-Functions.

The evaluation of the coordination process raised the question how SON-Function instances are actually triggered within SOCRATES. Publications and deliverables do not touch this topic directly, but the combination of Guard and Alignment function might be responsible for triggering SON-Function execution. Although the guard function is only introduced as the functional entity that detects undesired behavior caused by executed SON-Functions, seen in the context of the coordination framework it provides a larger functionality. As described in Section 2.3.3 it compares performance data to cell specific policies. Those cell specific policies contain target values for performance measurements for specific cells. The guard function will detect deviations from these values independently from the root cause. These deviations are then provided to the alignment function in order to detect their root cause. Activation (cf. Section 2.3.2) will respond to these notifications by triggering the appropriate healing or configuration SON-Functions. Apart from this mechanism no additional method of triggering SON-Function execution seems to be available.

2.5 Assessment of the SOCRATES Coordination Concept

The SOCRATES Project identified the potential dangers of conflicting behavior of independent individually operating SON-Functions within a network. As a consequence a conceptual coordination framework was

provided, which aims to provide all functionality required to prevent and resolve conflicting behavior.

Even though the positive effects of function coordination are evaluated and presented [1] most of the described functionality stays at a very abstract conceptual level and neither an implementation nor simulations have been performed. The available experimental results are reached through the implementation of a single specific coordination algorithm for only two SON-Functions. Which shows good results for this special scenario but can not be generalized. Especially as the function instances intereact directly and no coordination framework is used.

At least a prototypical implementation would be required to strongly support the introduced SOCRATES coordination approach. Otherwise it is unclear if it is a feasible and scalable approach that can handle a high number coordination of configuration requests.

The named redundancies and replications of information an functionality potentially hide a large complexity which can have major negative effects on the scalability and the usability of the approach. Especially if, each time a new SON-Function is introduced multiple other SON-Functions have to be changed in addition to the required updates of the alignment function. Updates of the alignment function are required in order to support the prediction of the effects of the configuration request requested by instances of the newly introduced SON-Function.

SOCRATES uses a history of performed changes for the root cause analysis of detected undesired behavior. The Alignment function undoes previous changes to resolve the undesired behavior. The project documentation provides no detailed information about how errors that have complex implicit relations as a root cause can be solved by using this approach. For example, if within a close vicinity, both a Handover Optimization and a reconfiguration of Physical Cell IDs (PCIs) has been performed, the handover failure rate could increase strongly even if the Handover Optimization actually optimized the network but the root cause is an accidentally introduced PCI confusion. For a successful resolution of undesired behavior a lot of operational knowledge and a possibility to correlate and assess the effects of different configuration changes that have been performed are required.

The SOCRATES coordination framework seems to have no mechanism to prevent SON-Function execution that is based on erroneous input values. Erroneous input values are typically performance measurements that do not or not yet fully reflect previous configuration changes. The severity of this issue depends on the way the Autognostic function can access and process performance measurements. The alignment function therefore has to have the ability to assess the effects for either any SON-Function in the network or for each thinkable combination of configuration parameters, which strongly increases the computational complexity of the tasks performed by the Alignment function.

Overall there are many open questions and thus doubts that an implementation of the SOCRATES Coordination Framework concept would be able to perform the anticipated conflict prevention and resolution in a network with a large number of SON-Functions. A feasibility study for some of the central parts of the concept would have been required, as for example for the Policy Function as already stated in Section 2.2 since it is the most central part of the concept. Also the functionality provided by the Alignment function should be re-evaluated and, if necessary, reorganized or put into other modules.

References

[1] Mehdi Amirijoo, Andreas Eisenblaetter, Litjensm Remco, Michaela Neuland, Lars-Christoph Schmelz, and John Turk. A Coordination Framework for Self-Organisation in LTE Networks. In *IM 2011*, 2011. submitted for publication.

[2] Frank Lehser. Next Generation Mobile Networks - Informative List of SON Use Cases. Technical report, NGMN Alliance, 2007.

[3] Christoph Schmelz. Socrates - FP7 - http://www.fp7-socrates.org. Website, 12 2008. 02.12.2008.

[4] Neil Scully, Kristina Zetterberg, Szymon Stefanski, and Lars Christoph Schmelz. Socrates deliverable 5.10 measurements, architecture and interfaces for self-organising networks. Technical report, Socrates FP7 Project, 2010.

[5] John C. Strassner. *Policy-Based Network Management: Solutions for the Next Generation*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2003.