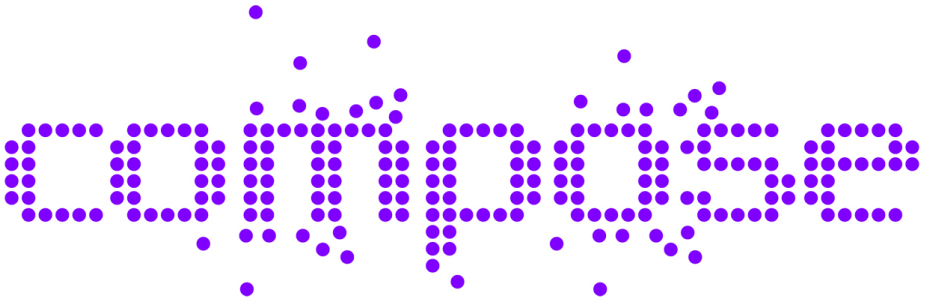

Collaborative Open Market to Place Objects at your Service

D1.1.0

Progress beyond state-of-the-art

Project Acronym	COMPOSE	
Project Title	Collaborative Open Market to Place Objects at your Service	
Project Number	317862	
Work Package	WP1 COMPOSE architecture design and specification	
Lead Beneficiary	OU	
Editor	Jacek Kopecky	OU
Reviewer	Carlos Pedrinaci	OU / WPL
Reviewer	Joachim Posegga	PASSAU
Reviewer	Benjamin Mandler	IBM / Coordinator
Dissemination Level	PU	
Contractual Delivery Date	2013/01/31	
Actual Delivery Date	2013/01/30	
Version	V1.0	

Abstract

This deliverable details the state of the art relevant to the COMPOSE project and describes the envisioned progress beyond the state of the art. It also points out the technological baseline for the different aspects of the project, namely technologies that will be brought to the table by partners at the beginning of the project development cycle. The sections of this document cover various aspects that follow naturally from the aims of the project: These are Objects as a service (WP2), Service-oriented software engineering (WPs 3.1, 3.2), Scalable communication infrastructure (WP4), Security (WP5), as well as Marketplaces and app stores (WP6). Finally, the deliverable also discusses selected finished and ongoing related research projects, showing how COMPOSE builds on their work and advances the state of the art. In effect, the deliverable lays out the plans of the COMPOSE project, as seen at the project's very beginning.

Document History

Version	Date	Comments
V0.1	2012/12/05	Initial version with first contributions
V0.2	2012/12/22	First almost complete version
V0.3	2013/01/15	Section 2 updated by David
V0.4	2013/01/16	Consolidated edits and comments from reviewers
V0.5	2013/01/23	All sections updated by authors
V0.6	2013/01/28	First final version
V1.0	2013/01/30	Submitted final version

Table of Contents

Abstract 2
Document History 3
Acronyms 6
1 Introduction 7
2 Objects as a Service 7
2.1 State of the Art 7
2.2 Progress Beyond the State of the Art 11
2.3 Baseline Technologies 12
3 Service-oriented Software Engineering 13
3.1 State of the Art 14
3.1.1 Service Description Technologies 14
3.1.2 Service Discovery 14
3.1.3 Service Composition 15
3.1.4 Service Orchestration 15
3.2 Progress Beyond the State of the Art 16
3.2.1 Service Description Technologies 16
3.2.2 Service Discovery 16
3.2.3 Service Composition 17
3.2.4 Service Orchestration 18
3.3 Baseline Technologies 18
4 Scalable Communication Infrastructure 18
4.1 State of the Art 18
4.2 Progress Beyond the State of the Art 21
4.3 Baseline Technologies 23
5 Security 23
5.1 State of the Art 24
5.2 Progress Beyond the State of the Art 26
5.3 Baseline of Research 27
6 Marketplaces and App Stores 28
6.1 State of the Art 28

6.1.1 Marketplace and App Store Ecosystem 28
6.1.2 Marketplace Solutions and Features 29
6.1.3 Web Market and Apps 29
6.2 Progress Beyond the State of the Art 30
6.2.1 Concept 30
6.2.2 App Usage and Distribution Analytics 30
6.2.3 Cross-device Apps and Cross-platform GUI 30
6.2.4 App Discovery 31
6.2.5 App Distribution 31
6.2.6 Developers SDK and IDE 31
6.3 Baseline Technologies 31
7 Comparison and Progress with Respect to Other European Projects 32
7.1 SOA4All 32
7.1.1 Progress Beyond SOA4All 33
7.2 iCore 34
7.2.1 Progress Beyond iCore 35
7.3 webinos 36
7.3.1 Progress beyond webinos 37
8 Conclusion 38
9 References 38

Acronyms

Acronym	Meaning
COMPOSE	Collaborative Open Market to Place Objects at your Service
NED	Network Embedded Devices
WSN	Wireless Sensor Networks
DSL	Domain Specific Language
IoT	Internet of Things
IoS	Internet of Services
IoC	Internet of Contents
DHT	Distributed Hash Tables
BYOD	Bring Your Own Device
CSP	Communication Service Providers

1 Introduction

In this deliverable, we detail the state of the art relevant to the COMPOSE project, and we describe the envisioned progress beyond the state of the art. We also point out the technological baseline for the different aspects of the project, namely which technologies will be readily available at the beginning of the project vs. technologies that will be developed throughout the project's lifetime. The sections of this document cover the various aspects that follow naturally from the work structure of the project. They are:

- Objects as a service (WP2)
- Service-oriented software engineering (WPs 3.1, 3.2)
- Scalable communication infrastructure (WP4)
- Security (WP5)
- Marketplaces and app stores (WP6)

In every section, we not only discuss the relevant state of the art, but also clearly identify the baseline technologies that we intend to build upon, and how far we plan to go in COMPOSE.

Finally, we discuss selected finished and ongoing related research projects (SOA4All, and iCore), showing how COMPOSE intends to build on their work and advance the state of the art.

2 Objects as a Service

2.1 State of the Art

Networked embedded devices (NEDs) include any electronic device equipped with a wireless or wired communication interface, and various sensors or actuators. According to this definition, various devices such as digital photo frames, mobile phones, remote controls, or networked media players are all NEDs.

A NED application (also called pervasive application) refers then to any software application that uses and combines in some way services offered by NEDs, for example real-time and historic data from one (or many) energy meter(s) in a house displayed on a digital photo frame. Even though the gap between the physical and virtual world is constantly shrinking [86], interoperability between NEDs from different manufacturers remains limited as no unique standard for connecting physical devices and applications prevails. A wide variety of hardware platforms with diverse capabilities and functionalities have been developed, and often because of commercial reasons most of these devices use incompatible communication protocols. As a consequence, applications that integrate data and services from different NEDs usually present numerous challenges [162]. In particular, most services and tools used both in the back-end and front-end of such composite applications (for example: event detection systems, data processing engines, databases, or Web interfaces) have to be customized for the devices at hand for each project. This results in a waste of resources that could be used to design the application itself, rather than re-implementing components that already exist and could simply be reused. As a result of recent initiatives such as the Make magazine [87], hackerspaces [88], or FabLabs [89], hobbyists and amateurs have been increasingly empowered to build physical computing applications. Tools such as Processing [90] and Arduino [91] have lowered the barrier to use

and program electronic devices, which enabled a large community of designers and developers to create interactive applications with little technical knowledge. However, large-scale distributed sensing applications will require reconsidering how one designs, builds, and deploys applications that take advantage of many networked resources. Easy to use and versatile standards for interacting with embedded devices will be essential to achieve a uniform, scalable, and robust common ground where diverse devices and services can exchange data efficiently.

A particular class of NEDs called Wireless Sensor Networks (WSNs) consist of autonomous sensors (also called nodes) that monitor certain physical conditions (such as temperature, vibration, pressure or humidity). These sensors are geographically distributed and communicate via a wireless (ad-hoc) network where data is forwarded (often via multiple hops) to a base-station. WSNs were initially used almost exclusively for scientific or research purposes (see [92, 93, 94] for general surveys about WSNs), but over the last decade they have gradually become invaluable tools in many disciplines from structural health monitoring, to building automation, to industrial automation, or even the smart electricity grid. As this field further develops, there will be a need for simple and standardized automated data acquisition and control mechanisms, but also ad-hoc manual or mobile interaction mechanisms. For most of these applications, interoperability with existing network infrastructures will be vital.

Typical WSN applications were initially based on a sense, store, and analyse pattern. Sensor nodes transmit the data they gather to a base-station (also called sink) where the data is stored and subsequently analysed by domain experts. A typical example of this pattern is the “Macroscope in the Redwoods” project [95] where sensor nodes were deployed on several redwood trees in California and monitored various environmental factors such as temperature, humidity, and solar radiation over long periods of time.

With the growing usage of WSNs in different areas (e.g., military, environmental, health and home applications), continuous monitoring and immediate event detection became an important requirement as well [92]. As embedded systems keep improving, more processing power and smaller energy use will enable monitoring environments with higher sampling rates. As a consequence, it is very likely that sensor networks will soon become key instruments to observe large-scale phenomena with high temporal and spatial resolutions.

Much of the existing research in the WSN domain addressed specific problems in small-scale deployments such as minimizing energy consumption [96, 97], routing algorithms and MAC protocols [98], or programming paradigms [99, 100, 101]. Because little attention has been paid so far to higher level issues such as interoperability and simpler programming models, most projects still operate in isolation and are incompatible with each other. However, as concrete use cases for these technologies appear and become financially viable, new research objectives for the WSN community have appeared, for example support tools for developing, deploying, and debugging large-scale applications. Besides, the need for sharing device data and services with more actors and users also increases; therefore scalability, robustness, and openness are becoming critical requirements [102].

According to the IP for Smart Objects Alliance (IPSO) [103], an increasing number of embedded devices will be supporting the IP protocol. Many physical objects will therefore be able to connect to the Internet, especially as IPv6 and its embedded variant 6LowPAN reach mainstream acceptance. The convergence of NEDs and the Internet provides new design opportunities, as digital communication networks will soon not only contain virtual data (images, text, etc.), but also data and services from physical objects. IP-based sensor networks have become widely adopted as a way to provide interoperability both with existing networking infrastructure and with existing equipment [104, 105, 106, 107, 108]. Unfortunately, even though IP provides a common networking layer for increased interoperability and evolution of

the system, application-layer interoperability is by no means guaranteed although it is the crucial element for building scalable distributed applications on top of heterogeneous devices.

Because the number of embedded devices with built-in Internet connectivity is growing exponentially, it is very likely that in the near future there will be more NEDs on the Web than humans. The combination of increased processing power on NEDs and the flourishing of robust and easy to use frameworks for developing Web applications will make it easy to fast prototype Web applications that spread from browsers and servers into the real world. However, to realize the vision of a global network of NEDs, devices will need to be fully integrated with the existing Web infrastructure. This means NEDs shall support the same protocols and behave just like any other Web resources. Using Web standards to interact with NEDs will make it much easier to merge the real world with existing Web content and physical things could then be bookmarked, browsed, queried, or shared just like any content on the Web.

At the core of the World Wide Web (WWW) lies the HTTP protocol, which was designed as a substrate for building a distributed hypermedia system for linked multimedia documents [109]. As demonstrated by the tremendous success of the Web, loosely coupled approaches offer the greatest scalability, robustness, and evolvability. In recent Web 2.0 applications [110], the focus has been on the user and on user-generated content, and many applications (and especially the data and services within them) were made accessible via open Web APIs. This has lowered the barrier for programmers to build Web mashups, which are hybrid applications that combine content from various sources, as for example the Chicago crime maps [161].

The *Internet of Things* [164] (IoT) provides a vision in which the Internet embraces everyday objects in the physical world. Items are no longer disconnected from the virtual world, being controlled remotely and acting as physical access points to Internet services. Beyond the Internet of Things where no unique application-layer protocol has gained widespread adoption (i.e. has become a standard), the Web of Things[1] vision proposes to leverage the widely used Web standards as the common application protocol for improve reliability. This bridges the fields of Web technologies and embedded sensing into a unified vision where the Web's well-known standards and tools are leveraged to seamlessly blend NEDs with the existing Web infrastructure. The Web of Things is a viable solution to build more scalable, open, and flexible NED applications for various reasons:

- First, full integration of devices (as opposed to only integrating their data or using a Web page to control them) allows treating devices and their services just like any other Web resource. This makes it easy to tap directly the expertise accumulated over the last two decades in building massively scalable Web sites, but also benefit from other tools and techniques widely used on the Internet such as caching, linking, search, authentication, or scripting among others. In other words, native Web integration allows devices and NED applications to directly leverage the Web with minimal effort.
- Second, native integration diminishes the costs to network heterogeneous devices as the Web infrastructure is already in place: TCP/IP is omnipresent, Wi-Fi routers are ubiquitous in occidental households, and Web standards are efficient, well known, and used by millions of developers and billions of users. HTTP is a highly versatile and omnipresent protocol thanks to its simplicity, powerful and scalable Web servers are freely available as open-source projects, HTTP clients and libraries exist for virtually any programming language and platform. Furthermore, using Web standards to interact with devices makes it possible for the applications built upon them to benefit from the system properties introduced by the modern Web architecture such as scalability, evolvability or simplicity.

[1] See: http://en.wikipedia.org/wiki/Web_of_Things

- Third, Web applications are often simpler and faster to develop than classic desktop software applications. Current software for real-world integration and business applications are tailored for specific use cases, thus are often too rigid and closed to be customized by end-users easily. In a large ecosystem of networked devices, one could use a higher-level, declarative approach to rewire and recombine data, in particular use tools commonly used to develop Web mashups.

Tightly coupled back-end systems using proprietary or optimized protocols [163] will remain the most desirable choice for high-performance systems with specific requirements (as for example in the industrial automation or banking domain). Nothing prevents the functionality of these systems to be exposed on the Web using a high-level API to hide the complexity and underpinnings of the closed back-end. However, much simpler loosely coupled approaches are to be preferred for tasks where latency and throughput are less of a concern (which represent most use cases in building automation or environmental monitoring), because of their inherent flexibility and intuitive use.

Allowing people to easily reuse and combine data sensed by different devices anywhere around the globe will lower the entry barrier for developing monitoring applications. The loss in raw performance induced by choosing HTTP over optimized protocols is largely compensated by the seamless Web integration, a simpler programming model, and by the availability of tools, techniques, and expertise in building robust, secure, and massively scalable Web applications.

With more than 50 billion connected devices predicted to be online by the end of the decade [162], object management is going to be a big challenge. Objects will produce data that needs to be combined and analysed alongside structured data, application logs, customer info, and social media streams. Exposing physical objects as service objects in the form of high-level services requires that the back-end that manages their data, representation, and interfaces can scale out according to the needs of the object population. The integration of Internet of Things (IoT), Internet of Services (IoS), and Internet of Content (IoC) will only increase the size of big data problems. Data management and structuring for cloud-scale infrastructures is dominated by two different industrial solutions: BigTable [111] from Google, and Dynamo [112] from Amazon. Both of them are key-value structured storage systems and provide good scalability properties, and beyond their creators, other cloud-services companies have adopted them, such as Facebook and Yahoo!, to manage their large datasets. Their alternative open-source solutions are Cassandra [113] (derived from Dynamo) and HBase [114] (derived from BigTable), both from Apache. Other popular alternatives in the field of the so-called NoSQL databases are CouchDB [115] and MongoDB [116] which are both document based. In the case of CouchDB documents associated to keys are stored as JSON records, while in MongoDB documents are stored as BSON data. Finally, CouchBase Server [117] groups into one single project the NoSQL principles of CouchDB with the in-memory key/value caching capabilities found in Memcached [130].

As for large-scale data processing, the existing solutions can be divided based on the approach taken: real-time (streaming) or batch processing.

In the field of batch data processing, Microsoft proposes Azure [118] as an application platform for the cloud, based on the Windows Azure operating system, offering basic services, like Azure SQL, .NET services, and means to link to Windows Live. The Manjrasoft Aneka [119] platform is oriented to .NET and offers a cloud/grid platform for authentication/authorisation, dynamic resource allocation, and accounting. Some runtimes implement support for several programming models that aim to facilitate not only the management but also the development of massively parallel applications. The MapReduce programming model, introduced by Google in 2005 [120] aims to support distributed computations for huge datasets on large clusters. Some runtimes that support the execution of MapReduce applications are the Google MapReduce runtime [120],

Hadoop [121], provided by Yahoo! and backed by many companies, such as IBM, Amazon, Facebook, etc. Dryad [122] was a research project and the response of Microsoft to Google's MapReduce, aiming to subsume other programming models such as relational algebra. Some works already introduced the notion of QoS management for data analytics workloads, such as in [123, 124, and 125], extending similar concepts presented in [126] and [127].

Regarding real-time stream processing, two popular alternatives exist. One of them is the Storm Project [128], a distributed, reliable, and fault-tolerant stream processing system, which was open sourced by Twitter after acquiring BackType. Distributed by the Apache Software Foundation, Apache S4 [129] is the data streaming alternative, supported mainly by Yahoo!.

In recent years, in-memory key/value stores have gained momentum. It started with the extensive use of Memcached [130] for session caching in web applications. Other commercial solutions came to compete with Memcached after that, with some of them delivering some impressive performance properties. The Memcached protocol has prevailed in many situations as the way to follow for performing CRUD operations on key/value stores. Some solutions provide specific architectures for addressing the problem of storing large amounts of data in such a way that it can be quickly processed, as addressed in [131]. Finally, in the last years, some solutions based on the integration of in-memory key/value stores and NoSQL back-ends have appeared. The most representative case is CouchBase Server [117].

2.2 Progress Beyond the State of the Art

In COMPOSE, the objects will be exposed as high level services. For such purpose, each object will be represented in the two repositories planned for COMPOSE: the object registry and the object data store. The former will store the object representation, basic information, and semantic data. The latter will be used to store any object-generated data. Both data stores will have to offer flexibility to deliver linear performance scalability with the number of objects as needed to provide constant response time. The data stored in both data stores will be retrieved through data digestion layers, that is, high-level services that will get access to data management primitives and will provide customizable views of data. For that purpose, basic Domain Specific Languages (DSLs) will be leveraged to express, with minimal complexity for the service developers, such data processing primitives. The approach taken in COMPOSE will either be extending well-known data processing languages such as Apache Pig [132] or to develop new ones based on DSL development frameworks such as Scala [133] specifically aiming the needs of the use cases. At the same time, any object generating real-time streams of data will be associated with real-time distributed data processing mechanisms that will allow for the storage of digested data only. In other words, the deployed services will be allowed to express data processing primitives that will transform object-generated raw data into the defined data model. For that purpose, the same DSLs mentioned above will be leveraged. Interaction with physical objects will take place though the Web of Things approach: APIs based on web technologies (RESTful when possible) will be leveraged to sense and actuate the objects.

The object store might represent the most demanding component of the COMPOSE architecture, thus there will be a concentrated effort to make it extremely scalable and extensible to meet the demands of a highly dynamic and continuously growing environment such as the Internet of Things [162]. To achieve that goal, both data repositories will be designed as a multi-level key/value storage architecture. In COMPOSE we will define multiple layers of key/value stores according to the characteristics of existing storage devices (i.e. RAM, Flash, and Mechanical disks), allowing for an adaptive performance that scales linearly with the number of connected devices. In-memory key/value stores will be combined with state of the art NoSQL technologies to produce such a novel architecture, not existing in currently existing

technologies. New caching strategies will be explored to operate in fully distributed modes, providing critical reliability and scalability properties needed for the COMPOSE marketplace.

Aggregation, filtering, and joining of data in the object store will be handled with cloud-scale data analytics solutions, such as MapReduce. Data management primitives will be expressed using newly developed Domain Specific Languages (DSLs). The DSLs will be developed and the primitives will be deployed with the services. The COMPOSE framework will transform the primitives deployed with the services into data access strategies on top of the multi-level repositories. New advances will be required to extend such runtimes to support processing large amounts of data while observing QoS requirements imposed by objects and services, especially in the scope of on-the-fly integration and processing of heterogeneous multi-party information to provide near real-time services exploiting data streams coming from sensors. While work on data management and structuring exists, the goal of proposing new mechanisms for Internet-scale data management for a huge network of service objects poses new challenges when a multi-level architecture is considered, such as consistency, replication and performance management.

Services will have the option to be deployed with associated real-time data processing primitives. Such primitives will be expressed using Domain Specific Languages (DSLs) and will be translated into stream processing units that will digest object-generated raw data into data suitable to be stored. Changes in the services, or deployment of services with different needs will have to be handled by the COMPOSE runtime.

The data models selected to store object data will be enhanced to work with semantic data, keeping in mind that semantic composition will have to be supported. As a result, on-the-fly data model changes should be supported also.

2.3 Baseline Technologies

COMPOSE will build on top of existing state-of-the-art technologies. In particular, in the scope of the Objects as a Service work, we will start exploring the following ones:

REST (for the Interfaces)

REST (Representational State Transfer) is the architectural principle that lies at the heart of the Web. Rather than being a technology or standard, REST is an architectural style that attempts to increase interoperability for a looser coupling between the various components of distributed applications. In REST, a client issues a new request when it is ready to make the transition to a new state. Requests and responses are relative to the transfer of representations of resources. A representation of a resource is a document that represents the current or intended state of a resource. REST leverages the underlying principles of HTTP for the interactions between clients and servers, and consists of a set of constraints that defines how distributed applications should be built. In REST, Interactions happen directly with data instead of ready-made services wrapped around the data. This offers more flexibility for developers to access data in a neutral format and with a finer granularity. Interfaces are usually fixed, and each resource supports only a set of fixed operations that rarely change. Therefore no need to regenerate all the clients/stubs after each change, as would be the case with SOA. Thanks to these properties, resource-orientation allows more scalable and robust architectures, and for this reason REST defines the core architecture of the modern Web and emphasizes the use of the Web (and of its central protocol HTTP) as an application protocol, and not only as a transport protocol. Web applications that comply to the REST architectural style are said to be RESTful.

Data Back-End

CouchBase Server integrates an in-memory key/value store (memcached) and a NoSQL back-end (CouchDB) to provide a novel approach to the field of horizontally scalable databases. The communication between Memcached and CouchDB is performed through a custom C library interface to the CouchDB files, instead of the CouchDB original RESTful API. In fact there is not a direct communication between Memcached and CouchDB, but the Couchbase storage engine called ep-engine [95] ('eventually persistent') makes use of the CouchDB library in order to achieve persistence. This library provides functions to do all the operations available in the REST API and those functions deal directly with the database document files instead of using CouchDB as an engine. The Couchbase Server is completely compatible with the memcached protocol. To search and aggregate information stored within CouchDB database, views are used. Views convert the individual documents in the database into a list of information. CouchDB will be extended to work on a multi-layer storage platform (including slow HDDs, SSDs, and RAM), and therefore will require the development of new consistency and performance management strategies and algorithms. Also, it will be adapted to work in the COMPOSE architecture to provide the desired scalability properties required by the project, with mixes of real-time and historic data. Additionally, it will be extended to support the presence of embedded data management primitives within the services deployed.

Streaming

The two dominating distributed stream processing frameworks, Apache S4 and BlackType Storm will be the starting point technologies to be considered. Both offer similar capabilities, but it will have to be studied how they can be leveraged to perform the automatic translation between service-defined data management primitives into stream processing graphs.

DSLs

In general, a Domain Specific Language (DSL) provides a high level language specifically suited for one problem domain, which allows the runtime system or the DSL compiler to take advantage of domain specific optimizations designed to exploit the specific structure and characteristics of the domain's problems. Language virtualization allows implementing a DSL embedded into a host language to leverage as much infrastructure and features as possible from the host language.

In COMPOSE, different options to define DSLs will be explored to find the best trade-off between flexibility and simplicity for the COMPOSE use cases. Different technologies will be evaluated from the point of view of the needs identified in the use cases. Options will go from more static and simple solutions such as Apache Pig [132], to more complex approaches such as the Scala framework [133], which provides language virtualization support and generates Java byte-code so that the resulting program runs in the JVM execution framework.

3 Service-oriented Software Engineering

Service orientation [179] has increasingly been adopted as one of the main approaches for developing complex distributed systems out of reusable components called services. Realizing the potential benefits of this software engineering approach requires semi-automated and automated techniques and tools for searching or locating services, selecting the suitable ones, composing them into complex processes, resolving heterogeneity issues through process and data mediation, and reducing other tedious yet recurrent tasks with minimal manual effort. Research in this area therefore spans a large number of aspects, such as services descriptions and representation, communication protocols, service discovery, service composition, and

service orchestration, to name the main ones. In the remainder of this section, we cover the state of the art in each of these topics and indicate the contributions that will be made by COMPOSE.

3.1 State of the Art

3.1.1 Service Description Technologies

The initial work towards exposing IT services in the Internet was largely dominated by the Web Service Description Language (WSDL, [180]) and SOAP [181]. WSDL was created specifically for this purpose and was subsequently extended with the WS–* standards and composition languages [135] to provide further capabilities (e.g., orchestrations, security, transactions, etc) to service-oriented solutions [134, 136]. Recently, plain and simple Web technologies (HTTP, XML, and JSON), which underlie machine-oriented Web applications and APIs, are increasingly being used to provide access to distributed software seamlessly, constituting simple and "lightweight"[2] service-oriented software, commonly referred to as RESTful services—when they follow REST principles [143]—or Web APIs in general.

Independent from the technologies adopted, the systematic development of service-oriented applications remains less agile than initially anticipated. The fundamental reason for this lies in the need for software developers to devote significant labour to cover the life-cycle of services and applications. Semantic Web services research has tried to alleviate these issues by combining services with semantic technologies that can support a higher level of automation. A number of conceptual models and service annotation mechanisms have been devised to this end, which includes, among others, the Web Service Modelling Ontology (WSMO) [139], OWL-S [140], WSDL-S [141], which was subsequently simplified and standardised through W3C as SAWSDL [166].

The evolution in this area is leaning toward simple semantic models that can support higher levels of automation while retaining and minimizing the overhead related to crafting and processing the semantic descriptions. This focus on simplicity is also evidenced by the increasing adoption of Web APIs for which the community has still not agreed on a standard description technology, despite a number of proposals, e.g., WADL [142], hRESTS [144], and SA-REST [182]. Final trends in this area are the increasing integration of services with linked data [145]—dubbed linked services [147]—and the inclusion of business oriented aspects in service descriptions, e.g., USDL [183, 184].

3.1.2 Service Discovery

Universal, Description, Discovery and Integration (UDDI) [148] is perhaps the best-known effort to support the publication and discovery of WSDL services on the web, although it was discontinued in 2006, mainly due to a lack of support for expressive queries. On the basis of the aforementioned conceptual models, several researchers have worked on enhancing service registries, especially UDDI, using semantic technologies [149]. Additionally, much research has been devoted to improving the performance or expressivity of discovery algorithms and also to exploiting machine learning and information retrieval techniques to deal with less precise descriptions and to bring additional flexibility to the matchmaking process [149, 150, 151].

[2] "Lightweight" is used here to indicate the (perceived) low amount of infrastructure support needed when building RESTful services.

Work on Web APIs discovery is less mature. Perhaps the most popular directory of Web APIs is ProgrammableWeb, which at the time of this writing lists about 8,500 APIs. This directory is based on the manual submission of APIs by users and currently provides simple search mechanisms based on keywords, tags, or a simple prefixed classification, none of which are particularly expressive. APIHut [185] is a platform that increases the accuracy of keyword-based search of APIs compared to ProgrammableWeb or plain Google search, although it does not provide richer discovery mechanisms able to discern between services, operations, inputs and outputs, as necessary for supporting effective discovery.

iServe is a public linked services discovery platform that unifies service publications and discovery on the Web, through the use of lightweight semantics. iServe, previously introduced in [186], builds upon lessons learnt from research and development on the Web and on service discovery algorithms to provide a generic semantic service registry able to support advanced discovery over both Web APIs and WSDL services described using heterogeneous formalisms. iServe is, to the best of our knowledge, the first system to provide advanced discovery over Web APIs comparable to that available for WSDL-based services.

3.1.3 Service Composition

Service composition aims at providing solutions for requests that cannot be answered with existing services by dynamically discovering and composing services. To this end, a set of services have to be located according to their provisioned functionality, capability, context and dependency, in order to be combined to fulfil the request of the user. A survey on some of the main concepts and approaches in this area can be found in [154]. Existing approaches differ considerably on the basis of their capability to carry the composition at runtime (dynamic composition), the use of semantic descriptions to support the composition, the ability to monitor the execution and react or adapt accordingly, e.g., to maintain a certain Quality of Service, or to support transactions.

Among the most advanced approaches to supporting the automation of service composition, one can find those based on process algebras, semantic technologies, and AI planning algorithms. Some of these approaches have been surveyed in [155]. Fujii and Suda [156] also suggest a means for exploiting semantic descriptions to deal with highly dynamic and mobile environments featuring vast device heterogeneity like the IoS.

Work on service composition is tightly bound to composition languages, that is, languages that allow defining complex workflows out of existing services. An extensive survey of some of the main languages can be found in [157]. Among the main proposals, Business Process Execution Language for Web Services (WS-BPEL, [67]) is worth mentioning and has become, to a certain extent, the de facto standard within industrial settings. More recently, researchers have tried to accommodate Web APIs and provide new means by which non-experts could define simple workflows, also called mashups. In this last generation of approaches, we can cite, for instance, Bite and Yahoo! Pipes among the most relevant [158].

3.1.4 Service Orchestration

Once services are composed into workflows, the resulting processes needs to be executed. Most work in this area has focused on the development of workflow engines that can interpret the process descriptions and carry out their enactment in a centralized engine.

Chafle et al. [135] discuss run-time issues related to decentralized orchestration of composite Web services. The main issues highlighted concern the correct and efficient distribution of a workflow specification and the corresponding error handling. The authors' proposition focuses

on an efficient protocol for engine-to-engine communication, designed to avoid potential deadlock and error handling, as well as the corresponding recovery infrastructure. Nanda et al. introduce a code-partitioning algorithm, which can be deployed to distribute a Web service composition [159]. In this work, the main objective of its algorithm is to maximize the throughput of multiple concurrent instances and to minimize the communication costs.

Another relevant work was introduced by Liu et al. who provide an integrated framework able to deal with the lack of a general method to realise cross-platform distributed applications [137]. Finally, Barret and Pahl describe distribution patterns, i.e., reusable composition patterns that express how a composed system is to be assembled and subsequently deployed [160]. They allow modelling the distribution of a system and *ipso facto* facilitate meeting of varying non-functional requirements.

3.2 Progress Beyond the State of the Art

3.2.1 Service Description Technologies

Given the latest trends both surrounding services deployment on the Web as well as the related research efforts, COMPOSE will build upon and enrich the work done on linked services mostly initiated within the SOA4All project (see Section 7). COMPOSE will thus support both WSDL and Web APIs and will integrate these with linked data technologies, enabling services to process, produce, and be described as linked data. Particular emphasis will be put on Web APIs for their increasing popularity for sensor data and Web content publication. Linked services descriptions will be aligned and extended with sensor-specific description approaches (esp. [187]) to properly support the integration and management of objects. Finally, we shall envisage the inclusion of rules, e.g., expressed using RIF [188] to provide further capabilities such as services reactivity and the means to provide services' business logic in a declarative and decomposable manner.

The infrastructure will be complemented with novel solutions for the automated identification of services and a best-effort approach to the automated generation of annotations and additional metadata characterizing the services at hand in order to provide semantic descriptions of the services found in terms of the kind of service, the data provided, and some general assessments about their reliability, the quality of their data, etc. To this end we shall devise novel information extraction and semantic annotation algorithms able to process, wherever possible, the existing descriptions of services including for instance their HTML documentation, as well as machine learning algorithms for the automated characterization of services based on the data provided and previously encountered examples.

3.2.2 Service Discovery

Irrespective of the approach, service discovery has essentially been based on centralised repositories. That is, they assume all the descriptions are locally available. In contexts like the IoT, in which a large number of service objects and services will be available in a largely distributed manner, assuming a central registry for all the service description is not realistic. Additionally, most discovery engines are solely able to discover WSDL services that are not prevalent on the Web. In COMPOSE, we will build upon iServe, which provides state-of-the-art (semantic) discovery algorithms, to develop a distributed discovery engine able to provide advanced discovery support over Web APIs and WSDL services in highly distributed settings.

Semantic matchmaking algorithms will be extended and improved to provide efficient and scalable distributed discovery. At the lowest infrastructure level, the discovery engine will exploit the notion of a Distributed Hash table (DHT) in order to provide a distributed and highly efficient look up system supporting continual node arrivals and departures as well as failures.

The discovery of services by a user or an application has traditionally been an activity essentially triggered by the consumer, may this be at design time by a developer, or at runtime by an application using what is often referred to as late binding capabilities [147]. While this functionality indeed covers the most typical usage scenario, i.e., one whereby the developer or application knows exactly what type of service is required, it does not enable other highly valuable complementary usage scenarios. For instance, it does not promote or contribute in any way to application innovation since users are only driven by their initially anticipated objectives and therefore hardly explore the use of other services that could potentially be valuable for the objective sought [189]. Similarly, traditional discovery techniques do not directly cater for the continuous improvement of applications that could be achieved by continuously and dynamically changing the service(s) used on the basis of their currently exhibited behaviour or simply the availability of more suitable services at a given moment.

In the scenarios contemplated in COMPOSE, situations where a service disappears at runtime, its quality degrades, or a new more adequate service is made available, will be very frequent and must therefore be adequately accommodated. We shall devise a novel service recommendation engine that will provide continuous proactive advertising of services to users and applications based on the knowledge about i) the actual application being executed or developed, ii) the services used in the past in similar applications and others available in general, iii) previous actions of users.

At the infrastructure level, the service recommender will rely on the highly efficient, distributed and scalable infrastructure with notification support in order to quickly determine situations where potential improvements may be obtained by replacing services, and it will subsequently trigger the appropriate notification to the application layer. At the component level the recommendation engine will combine content-based and collaborative filtering techniques over services, applications, and users in order to determine potentially interesting services both at design time and at runtime [154]. Notably, we shall leverage the information automatically derived about services while discovering them as well as previous usage of services in other applications in order to better characterize the services as well as the users and their preferences.

3.2.3 Service Composition

Within COMPOSE, we will work on facilitating service composition by end users. We shall therefore develop (semi)automatic algorithms for service composition, based on lightweight semantic descriptions for service objects and services. We shall build upon state-of-the-art solutions for service composition based on semantic technologies but will retain a level of simplicity on services descriptions that can lead to good results with reasonable computational requirements for scalability reasons. The solutions we will devise will target highly heterogeneous settings (e.g., WSDL, Web APIs, heterogeneous sensors) and we will focus on supporting the deployment of the resulting workflows in a distributed fashion to maximize the scalability and reactivity of systems as necessary for dealing with real-time data coming out of sensors.

In order to cater for the dynamicity, scale and heterogeneity faced in the project we shall develop an assisted service composition engine able to provide efficiently semantically compatible service compositions of highly heterogeneous services. To this end, we shall i) leverage the use of Linked Data for capturing service descriptions and sensor networks,

ii) devise a novel approach to service composition based on the opportunistic interleaving of traditional semantic composition algorithms with our efficient dynamic semantic service discovery, iii) enhance the composition algorithm with advanced caching and dependency indexing mechanism to ensure an efficient execution, iv) combine the system with our advanced security analysis infrastructure in order to automatically provide an assessment of the level of security of the composition.

3.2.4 Service Orchestration

The ambitious goals of COMPOSE require sophisticated techniques for the orchestration of workflows such that i) services and service objects can easily be combined within processes, ii) services and objects can be dynamically replaced at runtime to cope with variable availability, requirements, and context, and iii) support for maximizing the scalability and performance of the execution even in scenarios involving a large number of devices that may be disconnected for periods of time.

To achieve these goals, we shall build upon [134] and [136] and shall leverage the advanced discovery algorithms that will be devised in the project to support the replacement of services at runtime, based on changing conditions. We will devise a novel distribution algorithm able to decompose the execution of workflow and efficiently deploy it over numerous objects with variable capabilities, in a manner that would allow an efficient distributed execution of workflows. The solutions devised will incorporate security mechanisms to ensure the validity of the executed process and the privacy of the data exchanged.

3.3 Baseline Technologies

The basis for **service descriptions** in COMPOSE will be the SOA4All technologies WSMO-Lite which builds on the standard SAWSDL, and hRESTS/MicroWSMO for describing RESTful APIs. Within the semantic annotations in these descriptions, we will use RDFS and OWL ontologies, with the possibility of expressing rules with RIF. Example ontologies already available for semantic annotations of services are the vocabularies for Semantic Sensor Networks and for Provenance (PROV-O).

For **service discovery**, the baseline is the discovery functionality in iServe, which includes RDFS and SKOS functionality classification matching, input-output matching, and text-based similarity search.

For **service composition** and **orchestration,** the main baseline technologies will be the work of Pfeffer et al. [134, 136].

4 Scalable Communication Infrastructure

4.1 State of the Art

Commercial systems, including large scale cloud infrastructure and IoT serving systems, are continuously growing in scale, heterogeneity, and complexity. This direction poses challenges for the back-bone clustering infrastructure used for providing continuous availability and internal communication within such systems. At the very centre of such systems lies a scalable communication technology which provides services to maintain information related to group

membership as well as scalable group communication mechanisms of varying patterns among participating nodes. In environments such as anticipated by COMPOSE, with ample communication requirements and complex services and applications composed from more basic services, we have to overcome scalability challenges intertwined with fast detection and notification of failures. We intend to build and improve on the wealth of prior art that exists in this area.

Scalable communication infrastructures in general and peer-to-peer systems in particular, are the subjects of active research. A good summary with a large number of references can be found in [7]. The tendency towards a peer-to-peer architecture grew hand in hand with the scale and distribution of envisioned systems and applications, partly due to its cooperative resource sharing nature. Some of the main benefits of such systems as described by Rodrigues et al. [1] lie with their independence from specific infrastructure and the lack of a centralized control point. We plan to make use of a peer-to-peer architecture which is up to the requirements imposed by the platform and services using it. It shall be a loosely coupled system that self organizes and self-adjusts according to run-time events and characteristics such that no central authority is needed to manage and control proper system behaviour.

In recent years, as the size of commercial systems and applications continued to grow, the need for scalable and thus self-managing communication infrastructure has intensified. In particular, peer-to-peer systems have become popular and have been widely used in main stream domains such as cloud computing components rather than only in the file sharing niche they captured in the past. A summary of the latest trends in the cloud computing area can be found in [2]. One of the finding in that study points to peer-to-peer technologies as one of the prominent means for handling the demands of growing systems scale. Overlay networks represent one of the available and promising peer-to-peer technologies. Numerous overlay network examples can be found in [3], [4], [5][6], [13], [14][15]. An overlay network, in our context, is one which maintains full access within a group of processes, from any process to any of its peers, without necessarily having physical connections between all members of the group. Our aim is to be able to efficiently provide all envisioned services while maintaining a small amount of physical links between members of the group. Prominent examples of such systems include Chord [13] which is one of the first published Distributed Hash Tables (DHT), and has set the path for many to follow. In Chord consistent hashing is used to clearly identify an element's location within the ring. Chord maintains a successor and predecessor for each location for correctness, and adds a finger table to enhance lookup performance. The fingers are selected in a deterministic manner thus forming a structured overlay. Chord's flat architecture hinders scalability and it's structured overlay design is targeted towards more stable environments than envisioned by COMPOSE. We intend to borrow some concepts from Chord, and extend as necessary, especially to attain higher scalability. An additional similar effort is Pastry [14] which handles a WAN connected system. Kelips [15] is another proposal for a DHT based on peer-to-peer technologies. Their unique point is to enhance system stability by sacrificing more participants' memory space and communication bandwidth, by constantly replicating indexing information as a background task. The system enables communication of membership updates rather quickly and should tolerate the existence of weaker nodes in the group. Astrolabe [81] is another example of a peer-to-peer system mainly used for management of large clusters and readily provides information regarding distributed resources, by creating a hierarchically distributed database which is kept up to date as the underlying information changes, and may be used for real-time data mining. In addition, Araneola [3] presents a scalable and reliable application level multicast system for highly dynamic wide-area environments. In this work an unstructured overlay topology is built, maintained, and used for transmitting by a gossip mechanism application level multicast messages. We plan to use such concepts and enhance them in a manner suitable for the platform, requirements, and envisioned workload and access

patterns at hand, by employing a hierarchical structure and adequate internal structure and relaxed semantics.

One of the most significant services provided by group communication systems is a membership service. Such a service provides a view of the currently active members of the group. For proper functioning, a membership service must identify failing members, receive and disseminate information concerning nodes leaving or joining the group, as well as discover new nodes that have become operational or connected. Currently existing membership services do not scale to the level envisioned by COMPOSE [18]. One of the manners that are used to reach high scale is to relax system semantics and guarantees. Naturally, this has to be well understood by users of such a service and should satisfy their needs. There are various membership services with varying level of semantics and guarantees, as detailed in [7]. The stronger the semantic guarantees provided, the lower the level of scalability achieved by such systems. Thus, systems providing a consistent membership service even under advanced concepts such as virtual synchrony [8], scale up to a few hundred nodes. Commercial group communication systems have been built on the basis of virtual synchrony [9], and we intend to learn from such experience as well. Virtual synchrony is a consistency guarantee for view-based group communication systems which ensures that all messages have been received by all surviving members of a view before embarking on a new view. Such a consistency guarantee is important for a certain kind of applications, but is a stumbling block for scaling. Thus, we plan to relax the semantic consistency guarantees in order to achieve higher scale. There exist known efforts to relax semantic guarantees to achieve higher scales. For example, [9] provides eventually consistent views, due to its conclusions that devising a distributed failure detector is hard and in reality exhibits problems in a scale larger than a few dozen nodes. A popular example of this direction is the Cassandra distributed storage system [11], created for the management of a large amount of distributed structured data possibly residing within failure prone commodity servers, while providing a highly available service. This service is designed not to have a single point of failure, and scales efficiently to a couple of hundred nodes. Our goal is to reach higher levels of scalability, thus among other mechanisms we intend to employ relaxed consistency guarantees, namely eventual consistency. In further efforts to scale beyond this size, there exist protocols such as SCAMP [12], which is a scalable membership protocol, providing randomized partial views. Such a system may scale up to tens of thousands of nodes, but the utility and robustness of partial views is rather limited, so that only a specific and narrow set of applications can make good use of such guarantees.

Many such systems are built around a “gossip” mode of communication [16], in which members of the group exchange membership and potentially additional information periodically. As explained by Allavena, scalability of such a fault tolerant infrastructure is achieved by relaxing the strong reliability guarantees in exchange for probabilistic guarantees. One of the tasks of a membership service is to detect faulty members. Such systems are extremely robust and easy to implement, but are somewhat slow at detecting and reporting members failures [17], and tend to generate traffic even when no membership changes occur. As explained by Gupta [17] scalable failure detection in a distributed environment is a challenge that needs to be addressed for run-time efficiency. Moreover they claim that traditional heart beating schemes are inherently not scalable. We plan to learn from such lessons and design a scalable distributed failure detector, with adjustable levels of accuracy, which shall satisfy applications needs without overwhelming system resources. We intend to make use of such building blocks while improving and adjusting them to the specific scale and needs of our system.

Traditional membership algorithms are flat and thus inherently raise a scalability issue. In order to enhance scalability a couple of methods called for a hierarchical membership scheme. HiSCAMP [18] offers a hierarchical construct which is dynamic, and self-organizing designed to work well in a rapidly changing environment. Their method takes into consideration the

underlying network topology and delivers partial views both on a local and a global scale. Census [19] describes a platform for building large-scale distributed applications, providing a membership service and a multicast mechanism by describing a hierarchy that takes into consideration the geographic location of the nodes. Their core algorithms are based on achieving consistent views within different nodes within their system. The system was evaluated to scale to a hundred thousand nodes. We advocate a hierarchical scheme that emphasizes fast failure detection and eventual consistency for scalability.

Another family of algorithms are tree-based. One such proposal, by Varma [20] described a flat tree-based algorithm, which scaled up to 1024 nodes. Tree-based algorithms tend not to exhibit good properties when faced with multiple simultaneous failures. Moreover, a failure of the root leads to problematic behaviour patterns until the tree can be re-built. We intend to improve on such schemes in our design by having redundant paths between nodes and thus handle gracefully more than a single failure at a time with no notion of a specifically designated root.

The issue of fast and accurate failure detection has been tackled in [82]. In this work it's claimed that failure detection time is the crucial component of the recovery process, and that accuracy of such detections should be assured, even at the price of killing a functioning process. In order to achieve their goals they call for constructing a layer of spies which guard different levels of the system and having each layer guard its following one, such that in case a system part needs to be killed, the smallest component is killed. We agree that failure detection time is crucial thus we call for fast failure detection schemes to be implemented even in a highly scalable potentially hierarchical peer-to-peer system.

The system we envision will tie closely together the membership services, via its underlying topology, with the group communication services running on top of it. The topology will be constructed in a manner that will enable efficient messages routing for different kinds of communication paradigms, such that different services can be built for the use of other platform components. Such services may include publish / subscribe and DHT. In addition, due to the scale we envision a semi-autonomous system that will need very little human configuration to get started and will be able to continue normal operation under different kinds of failure scenarios without human intervention. Thus, the system will exhibit many self-* characteristics, such as self-organization and management.

4.2 Progress Beyond the State of the Art

One of the major goals of COMPOSE is to design and develop a scalable and fault-tolerant communication network that will provide a basic infrastructure for connecting service objects and services to (i) connect all service objects into the marketplace of services, (ii) support service discovery and lookup, (iii) share information, and (iv) control and manage distributed marketplace services. Thus, we aim to alleviate existing system scalability problems by providing a highly scalable clustering infrastructure, based on peer-to-peer technologies. This system will provide its services both to higher-level applications as well as to the platform runtime. In order to reach the scale anticipated we aim to support eventual consistency guarantees in conjunction with a hierarchical design. This overcomes two of the main obstacles to system scalability, namely the stricter semantic guarantee, which is easy for programmers to handle but hinders performance and scale, and the flat architecture. Moreover, we aim to reduce the overhead incurred by such systems when no changes have taken place. We envision a "grow as you go" design such that the infrastructure will work equally well from small to large scale, and will not assume only a large scale design point. Smaller installation will not have to suffer the consequences of having a robust system at large scale as well. We propose an underlying

layer which supports application fault tolerance in conjunction with a platform promoting high availability schemes including proper placement and load balancing.

We will advance the state of the art in scalable communication infrastructure by designing the basic networking infrastructure of the marketplace in way that it will leverage the storage and computational resources available in participating nodes to execute various operations and services required by the service objects participants. Such an "edge computing" concept will harness the power of participating entities, such as smartphones, in order to distribute computations widely among the participants, and have the processing take place as close as possible to the data location or the end-user location. This will make the infrastructure resilient to faults and attacks, as it will be completely independent of the existence and availability of a centralized entity. Dealing with the heterogeneity of participating objects, namely smart objects, composite services, in addition to the more traditionally handled computing elements such as applications, servers and VMs is another angle in which we plan to advance the state of the art. Moreover, different entities may use different communication mechanisms, which is an additional challenge to handle in a uniform manner. In that respect special attention will be given to smartphones, by putting in place the necessary mechanisms to deploy, manage, and monitor services over smartphones.

We anticipate exposing several services for the benefit of other entities running on the COMPOSE platform, as well as for the platform management itself. These services are designed to provide a great deal of autonomy to the components themselves. Services envisioned include a membership service (including fast and efficient failure detection), and group communication services such as scalable publish / subscribe and DHT. Publish / subscribe enables application level multicast at scale (we opt for the broker-less variation), while a DHT service, enables large scale indexing and search / discovery capabilities, such as needed for search for service objects, services and their associated data. Additional potential services include a convergecast mechanism (a sort of a reverse broadcast), which enables efficient collection of information from participating nodes. Services provided by this layer will be readily available for use by the run-time component, which will allow it to take more informed decisions concerning issues such as components replacement. Thus, all services combined will portray a unified source of control, management, and information distribution. The exact set of services and their priorities will be set by the requirements of other COMPOSE components

Our design calls for increased scalability while not paying a high performance penalty, including fast failure detection and notification, and handling gracefully multiple simultaneous failures.

We envision a hierarchical peer-to-peer design. The building block will consist of a group, which is expected to be somewhat similar to that of Chord or Symphony [83]. Thus, each node is connected to a small amount of its peers, and its successor(s), and exchanges information with them using a gossip style mode of communication. For efficient routing, as needed for group communication services, we envision marinating peer connections, building a topology that will ensure efficient routing (be it deterministic based on node identities or semi-probabilistically). All members of the group will have access to information concerning all other members of the group, but will not necessarily hold all the information kept up to date continuously. Group communication services will operate throughout the entire group, regardless of full or partial information exchange between the communicating nodes.

We will focus on designing a system that is self organizing, thus automatically adapts to change, requires almost no manual operations, and simplifies objects addition/deletion to reduce the management overhead and ensure a sufficient level of scalability. Finally, the COMPOSE infrastructure will also handle heavy-churn scenarios, since service objects may join/leave the system at an unpredictably high rate. We anticipate that new requirements, such as supporting

special discovery capabilities, monitoring, and search, will be placed on this layer by the COMPOSE system.

4.3 Baseline Technologies

We aim to start constructing the COMPOSE scalable communication service from a research asset developed by IBM Research which provides membership and several flavours of group communication services in a modest scale of a few hundred nodes. Having built a couple such systems we have gained vast experience of varying algorithms as well as pros and cons of several design choices along with the adequacy of different approaches to meet different sets of requirements.

As we go through the process of defining requirements for the platform itself and the different components thereof we will extract the set of services that are required from the scalable communication infrastructure. Based on the expected workloads and access patterns we will employ the most adequate approach to meet the needs. For example, within the underlying topology of the peer-to-peer topology we will have to choose between semi-structured or unstructured underlying topologies. Such decisions will stem from the churn level anticipated in the system. There are several options on the table in terms of existing technologies, at different levels of evolvement and maturity, starting from a conceptual and algorithmic level all the way to existing prototype level code.

It is anticipated that we will have to add a hierarchical infrastructure on top of existing technologies. In addition there exist skeletons of group communication services using an underlying membership service, but all these will need to be enhanced and made to work well in a far more scalable environment which includes a hierarchical inter-dependent set of groups.

In addition, current technologies call for embedding a piece of software within each participating node. We will have to investigate and determine whether this is the most adequate way to move forward in our foreseen environment, or rather call for a change in methodology in which a representative daemon runs on each node and have different components, such as services and objects, running on that node register with it. The local daemon in turn will monitor the registered entities and will represent them towards the rest of the group. Such a methodology may be used in conjunction or as alternative implementation to achieve high scalability levels.

5 Security

The main focus of COMPOSE is the Internet of Services and its interaction and coupling with the Internet of Things and Contents. COMPOSE will therefore concentrate on the application layer and on the data that is processed. Further, in order to provide an effective security framework, we must also consider platform and network security. For this purpose, this project is going to discuss the applicability of existing solutions and adapt them if necessary. Additionally, the feasibility of existing mechanisms in the information flow domain will be investigated and the techniques known from this research area will be integrated in COMPOSE.

The next section will outline how the application and data focus in COMPOSE provides a research direction which has mostly been disregarded in previous work. As the body of literature in this field is very big, in particular in the information flow analysis domain, this section will focus on technologies which have been developed in projects or settings which are comparable to COMPOSE and its specific architectural characteristics.

5.1 State of the Art

Security has been discussed extensively in the research domain of the Internet of Things. Many security solutions have been proposed, these mostly concentrate on communication security. The EU project IoT-A [69] is a good example for this observation. As a consequence, the security of the Internet of Services has also mostly been analysed with a strong focus on communication security for Web services. The attacks discussed, analysed, and mitigated in this area mainly focus on message, targeting stateless Web services, XML structures and services processing XML, or service states to circumvent a pre-defined business logic [59]. This fact is also reflected by tools such as TulaFale [35] which is based on ProVerif [36]. They use abstract interpretation of SOAP messages exchanged between Web services to validate secure interaction.

However, as explained above, the focus of COMPOSE is application and software centric. Therefore, this section will mainly review frameworks which highlight secure execution environments and ensure the appropriate processing of security critical data, the protection of platform resources as well as other applications running in the same environment.

A project which is similar to COMPOSE is MobiCloud [53], whose security framework is mainly focused on the network layer and enforces classical security properties for node communication. Techniques such as attribute based encryption, context-aware routing, etc. are features of this security framework.

To protect nodes from malicious service executions and to isolate services from each other, virtualization techniques are used. Comparable to most existing approaches, applications in MobiCloud use different trust layers to obtain different capabilities such as access control. This resource- and node-centric approach is emphasised by security policies which are specified for nodes.

The project iCore ensures security by enforcing classical information security properties through access control mechanisms and cryptographic primitives [31]. It also aims at the resource level to enforce integrity, confidentiality, authenticity and non-repudiation. Thus, iCore also lacks the ability to dynamically enforce information security properties of data which is processed in dynamic service compositions. As soon as data is released at the resource level iCore loses the ability to control data usage.

Such elaborate monitoring capabilities required for usage enforcement techniques are foreseen in the EU project SOA4ALL. However, SOA4ALL only provides a high-level specification of security properties and policies [75]. They are seen as pure non-functional properties. While such a classification is admissible, it does not suffice to specify efficient and effective enforcement mechanisms. Finally, although many security properties can be enforced using runtime monitors they do not provide the ultimate solution. Monitors on their own cannot guarantee the information flow security goals targeted in COMPOSE.

This lack of information has been removed by the OMTP and the BONDI initiative in various specifications [63, 64]. However, also BONDI is similar in the level on which security enforcement is applied. Security requirements and policies, and therefore also their enforcement, are specified at a coarse-grained resource level. This granularity has been shown to be unfeasible for many applications in the Smartphone domain and appears to be problematic for modern application development involving a lot of security problems.

Despite this observation and many attempts to actually enhance or correct systems which implement similar security frameworks, the wholesale application community (WAC) adopted BONDI but did not notably enhance it [78]. It also offers an API-based security model and

isolates device capabilities and access to resources by using a middleware approach. The latter is a runtime environment for so called “widgets”.

The secure WebOS application delivery environment project webinos adopted most of the concepts of the WAC architecture and specification. It aims at a unified user interface on different platforms to encourage users to specify their security needs [49]. Further, webinos claims to provide better solutions to prevent attacks which are almost identical to so-called “confused deputy” attacks [48, 39, 51, 52] known from popular application runtime environments such as Android. The effectiveness of their solution has not been investigated yet. Webinos also focuses on access control as an enforcement mechanism to ensure information security. The access control mechanisms deployed in webinos are slightly more advanced in comparison to those applied in iCore or MobiCloud. The user can specify contexts in which access should be allowed [50]. However, this mechanism still suffers from its coarse granularity at the API level and at the enforcement point. Once access to a resource has been granted, the entire resource is accessible. Additionally, the processing of data is not controlled by webinos. Usage control cannot be enforced with such an architecture.

The FP6 EU FET project BIONETS followed a different approach. We mention this project here because some partners involved in COMPOSE have also been engaged in BIONETS, thus this know-how can easily be carried forward to COMPOSE. BIONETS was facing similar questions as it had to deal with dynamic service compositions which may change dynamically depending on context and performance. However, BIONETS’s approach to compose services with specific security properties was based on analytical methods guided by security requirements for data items. As compositions were based on models, validation techniques using a model-checking approach [34, 58] were used to verify security properties of service compositions. Rewriting methods [38] which are comparable to inlining mechanisms [47, 55, 27, 56] for reference monitors were deployed to eliminate potential, statically detectable security problems. The applicability of this approach to the crowd-sourcing domain, which is very similar to COMPOSE, has been demonstrated as well [72].

AVANTSSAR [26] follows a related paradigm to implement security, which is similar to the techniques envisioned in COMPOSE. Modern validation and verification techniques from the static analysis domain are deployed in AVANTSSAR [37]. They do not only support the automated verification of service compositions but, based on security patterns, they also support the secure development of applications. For this purpose a high-level security specification language called ASLan has been developed. It uses expression from Linear Temporal Logic to describe security properties, and it is claimed to be a super set of WS-BPEL [67].

The tools applied in AVANTSSAR detect different vulnerabilities. SATMC [28, 60] deploys bounded model checking to analyse trace-based security properties on inter service communication. SATMC basically validates specific security properties of communication protocols between application components. It successfully detected a SAML vulnerability which affected popular mobile phone vendors. Another tool used in AVANTSSAR [40] automatically generates mediators to ensure data security properties. Although COMPOSE also strives for the deployment of static analysis techniques on service compositions AVANTSSAR misses the flexibility and granularity required in scenarios like those contemplated in COMPOSE. The automatically validated workflows are fixed. Thus, the composition lacks flexibility and cannot account for a dynamically changing context. Furthermore, fine-granular flow properties as pursued in COMPOSE are out of the scope of AVANTSSAR.

Also the FP7 project Aniketos - Ensuring trustworthiness and security in service compositions - does not increase flexibility [79]. It basically enhances and refines the capabilities of policy specification languages and tools developed in AVANTSSAR. Compositions patterns are used for a secure development process. These can be verified in MATTS, which is a user front-end

for validating security compositions. The composition plan defined by the developer is checked against predefined patterns and suitable feedback is communicated to the developer. In COMPOSE, the use of static security patterns is not feasible to correctly validate the usage of data as the security requirements for components may change depending on the data processed. Consequently, ensuring data security properties in Aniketos is restricted to access control. Thus, although Aniketos also deploys related security by contract mechanisms and monitor inlining techniques [29] as projected in COMPOSE, the techniques appear not to be feasible to ensure security-related flow properties. Instead, COMPOSE is going to use similar mechanisms [38, 72, 73] which offer a higher flexibility through their focus on data items.

Aniketos checks service compositions after they have been generated. The FP7 project ASSERT4SOA has chosen a different approach and defines secure composition patterns [68]. They support the developer in directly discovering insecure compositions. This approach is similar to the mechanisms envisioned in COMPOSE. It also uses a specification language which defines side effects and preconditions of composition modules. The main difference to COMPOSE will be the way such compositions are verified. As COMPOSE operates on the data level, security patterns will not be sufficient.

Aniketos also targets the certification of security properties of software services [71]. It allows for the certification of the behaviour of service components. Using secure composition patterns specific security properties can be assumed also for specific compositions. This characteristic is desirable as it simplifies software verification and reduces the related effort. However, the security properties currently verifiable with such an approach do not expand to security properties of data which have to be maintained in COMPOSE.

5.2 Progress Beyond the State of the Art

Today's perimeter-driven security has obviously reached its limits since state-of-the-art platforms like distributed Web applications or smart-phone platforms simply fail to provide suitable perimeters. Consider a business application running in an Android platform and a BYOD ("bring your own device") scenario: Neither a functional view, nor a communication view, not even a combination of both, provides a useful basis for defining or even verifying security properties in advance, since relevant specifics of the execution platform and the actual interaction of the application in question at run time with other applications simply cannot be determined in advance.

This situation resulted in the re-discovery of information-flow analysis [70] in security: Obviously, a data-centric view and properties thereof are an adequate approach to compensate for the effect of the dynamically "moving" security perimeter. One fundamental problem, however, still remains and gets even more difficult: When verifying security, what are the properties to aim for? Unless we overcome the need for statically defined properties fixed in advance, we will hardly be able to secure such real-life applications in a useful manner, i.e.: without being overly restrictive and therefore prohibiting practical acceptance of such security solutions.

COMPOSE will overcome this by tackling the problem at its source; we automate the process of deriving security-related properties of applications. This automated analysis will be used to investigate the data flow compliance of generated service compositions running in the COMPOSE framework. Additionally, we will also ensure that dynamically injected data whose security requirements change during runtime are processed according to their security requirements. Thus COMPOSE will not only ensure communication and platform security but also guarantee the correct processing of security critical data in the COMPOSE back-end as well as in the customer devices.

In that respect, COMPOSE also introduces a new dimension in the security response: Unattended Adaptive Security. While automatic patching mechanisms and autonomic security parameter tuning are well known in the security community [41, 47, 55, 56, 57], automatic and unattended manipulation of networks and services to guarantee specific security properties move towards a new dimension. As we consider this dimension to be indispensable for COMPOSE, we will aim towards its realisation.

Information flow analysis requires a magnitude of resources. As the proposed methods have to run on a large number of dynamically changing services and potentially on IoT-devices, efficient, on-the-fly mechanisms have to be supported. COMPOSE will use and extend existing and well-known mechanisms, such as incremental static analysis [74, 80], lazy abstraction [32, 33, 34, 43] staged information flow analysis [42], as well as program slicing [77, 76, 61] techniques. Also hybrid systems which combine and extend popular mechanisms of the software analysis domain [30, 46, 45, 65, 66] are foreseen. While there are some examples of these hybrid combinations their extensive use is still missing in particular in systems and marketplaces comparable to COMPOSE.

In addition, COMPOSE will enhance its market components with processes which also allow for the efficient certification and validation of information flow characteristics of applications offered by the market. Here, COMPOSE is going to use inspirations from the proof-carrying code paradigm [62]. It will be adjusted in such a way that flow characteristics are efficiently verifiable. Here, COMPOSE will develop a novel process in which the proof generation is independent from the flow policies specified for the information processed by an application.

As analytical processes in application markets are often not scalable, COMPOSE will investigate methods known from the variability-aware analysis [54] to efficiently verify security properties of data flows in systems which may run arbitrary combinations of applications.

To support the above processes, COMPOSE will also couple the analytical processes with a data provenance framework. This will not only induce and enhance existing policy and API specification languages but also require the definition of new runtime environments. Additionally, new efficient monitoring mechanisms will be required to develop a scalable and distributed architecture able to track data and their appropriate usage.

The further refinement of the security perimeter envisioned by COMPOSE comes at a price. It induces an analytical as well as a runtime overhead which will be investigated in COMPOSE. However, the dynamics, flexibility, and variety of applications enabled by the Internet of Services demand such fine granularity. Otherwise, existing security mechanisms will either be insufficient to protect users and their private data or too restrictive for the development of the future Internet.

5.3 Baseline of Research

The security architecture in COMPOSE will rely on static analysis mechanisms and dynamic enforcement technologies mainly guided by the provenance and usage of data.

To ensure information security properties we will use frameworks which support the concept of sticky usage control policies. An implementation for Android which defines and enforces such policies during initial access time, is already available and can be adjusted accordingly [178]. During the lifecycle of a COMPOSE application, it can be used to provide usage control policies for data in the analysis as well as execution phase of applications.

Furthermore, our techniques to track the security state of information processed by an application can be based on a real-time policy tracking mechanism which has also been

developed for the Smartphone operating system Android [177]. The COMPOSE core, will enhance these tracking mechanisms by mechanisms which also accumulate additional information such as origin and use of data. As our real-time tracking mechanism is a central lightweight monitor, we can also deploy it on "things" which contribute or process data in COMPOSE.

Based on the user-specified security requirements for data and based on the provenance of data, COMPOSE will deploy different types of analyses. To analyse applications running on COMPOSE "things" we will build on mechanisms we have developed for Android applications. These static analysis methods are able to analyse bytecode and detect control or information flows which leak private information to remote locations, which alter data in unauthorized ways, or which perform unauthorized system actions. The nature of this analysis is lightweight and can also be integrated in devices with less powerful computational resources. Further it is independent of the source code of an application. This additionally increases the flexibility of our security framework. Thus the adaptation to the runtime environments and application framework used in COMPOSE will be focus of our work. The actual analysis framework for Android is currently under development but is about to be published.

To increase precision and flexibility of our analysis, we also extended the configurable program analysis (CPAchecker) framework [174] to Android bytecode. We will use it to provide static attestations for applications which can be verified on even resource constraint devices contributing to COMPOSE. Of course, the choice of static analysis frameworks deployed in this project will also depend on the technologies used to implement the COMPOSE framework. Here, our expertise includes SOOT [172], WALA [175], Jif [176], and LLVM [173]. This puts the analytical tasks on a broad basis. As the latter approaches and the CPAchecker can also be applied to the source code of an application we will be in a good position to integrate tools which support the developer with implementing applications with required security properties.

The work on automatic reconfiguration of applications to inline security reference monitors will resort to mechanisms we also have designed for Android applications, to frameworks which have been developed in the EU projects BIONETS and WebSand, and to approaches we have developed using the aspect oriented programming paradigm.

6 Marketplaces and App Stores

6.1 State of the Art

6.1.1 Marketplace and App Store Ecosystem

Marketplaces and app stores have become the new rage in the evolving world of technology and lifestyle, with the development of leading apps stores and touch screen devices becoming essential tools in everyday life. According to Gartner, in January 2013 Apple's App Store reached more than 40 billion app downloads with nearly 20 billion in 2012 alone. The Apple App Store has over 500 million active accounts and had a record-breaking December with over two billion downloads during the month [170]. This incredible development indicates continued strong demand for mobile app content. They also mentioned that Apple's market share is the largest, considering its App Store accounts for 25 per cent of available apps in all stores [21]. But times have changed: the overall number of apps available is driven by an increasing number of stores in the market today that includes platform owners, device vendors, communication service providers (CSPs) and others who want to offer mobile app services as planned by the

COMPOSE Open Marketplace. These stores will see their combined share of total downloads increase, but demand for apps overall will still be dominated by Apple, Google and Microsoft. Besides this small number of major app stores, Gartner analysts believe that there are also stores from third parties that attract users with their brands or take advantage of the lack of dominant players in some markets. For example in China there is a boom market of independent Android stores, due to the lack of presence of Google Play and 'weak' stores from CSPs. Gartner expects to see more new entrants to the market, aiming to deepen relationships with their customers and/or to capture some of this growth market. Especially in this diverse environment COMPOSE will enhance the current situation with a cross-platform solution which is open for apps, objects and services.

6.1.2 Marketplace Solutions and Features

Most current app stores are selling native applications. The evolution was initiated by the *Apple App Store* (iTunes Store) in 2003, but the massive investments and the sustained efforts by all competitors in this market shows the significance of such provider-driven platforms, which allow for third-party app deployment and sales. Virtually every mobile OS is highlighting its own take on the mobile application storefront. *Marketplace solutions* differ in primary indicators, like the way applications are structured, which search options exist, whether applications can only be bought on the phone or also on other devices (supporting cross-platform app deployment), and whether any recommendation systems and rating functionalities are supported. These store solutions, coming from such competitive companies as Apple, Google, Palm, BlackBerry, Nokia, and Microsoft, are notably similar in appearance and performance. They promote *similar feature sets*, for both users and developers, despite having arisen from fundamentally different models of operation, business (in-app purchasing and advertising), and ecosystems, as well as from development approaches.

6.1.3 Web Market and Apps

App stores have recently entered the *Web market*, with Web applications or widgets, which attempt to combine the best from the Web and mobile in terms of being integrated (providing the convenience and rich interaction of traditional native applications), discoverable, and linkable. Furthermore these applications are built using standard Web technologies, such as HTML, CSS, and JavaScript to achieve platform independence. Therefore Web applications as an alternative to platform-dependent native apps can mitigate developer concerns to some extent [22]. In the past, mobile Web techniques hardly caught on due to usability issues. Supporters of *Web technologies* now claim that with more powerful processors in the smartphones and more capable mobile Web browsers that support some of the newer standards such as HTML5, usability will improve. HTML5 provides capabilities that are very close to native capabilities for supporting audio, video, user interfaces, and offline support for computing using local data storage. The EU funded project webinos (FP7 project no. 257103) addresses this Web approach by delivering an Open Source platform for Web applications across mobile, PC, home media and in-car devices [23]. It cannot be denied that cross-platform technologies are changing the market for manufacturers, CSPs, developers and users. To conclude the COMPOSE Open Marketplace will follow this trend and help the above mentioned stakeholders to develop, deploy and advertise platform-independent apps, objects and services.

6.2 Progress Beyond the State of the Art

6.2.1 Concept

The COMPOSE project will enhance the state of the art in app stores for native applications by providing an environment in which not only applications but also service objects and services can be traded. COMPOSE promotes a *novel concept* of trading services and service objects by sharing their interfaces between 3rd party applications which are able to run in the context of other users who are not the owners of the service object itself. COMPOSE will provide software components for managing the access rights for each smart object connected to the marketplace. On top of that, COMPOSE will give the owner of the object the full control to monitor the history of each application that accesses the service object and to update the associated rights and rules for accessing the service at any time.

Furthermore, the COMPOSE project will enable the information stemming from such objects to be integrated into software components, and smart objects to be dynamically controlled by services actuating on them. The COMPOSE open marketplace is then expected to contribute to the creation of a new generation of app stores that will enable developers to create services interacting with, and acting on, the real word through leveraging services for smart objects.

6.2.2 App Usage and Distribution Analytics

COMPOSE will further integrate automated *application analysis* mechanisms into its market to support the efficient and automated detection of malformed applications. In addition, COMPOSE will also enable developers to design secure applications a priori. As existing analysis frameworks are mostly general and target holistic solutions, COMPOSE will complement such processes with additional information, such as semantic descriptions of APIs or labelled data, and focus on specific data-centric security issues. This will help to advance such analytical processes towards being fully automated. Additionally, the COMPOSE open marketplace will enable mechanisms for tracking and measurement of app usage and distribution. These analytics can be used as feedback for developers but also for app ratings and recommendations.

6.2.3 Cross-device Apps and Cross-platform GUI

The COMPOSE open marketplace will address cross-device apps such as mashups of service objects and services to be used not only on smartphones, but also on desktop computers, set-top-boxes or TV sets. The challenge, which applies equally to users, developers, and CSPs of an app, is to support as many terminal devices as possible. Users would like to access individual services on all of their devices whereas the CSP would like to make the service available with as little operational effort as possible. COMPOSE open marketplace will facilitate the convenient creation, administration, distribution, and updating of apps for a variety of terminal devices. Thus, the COMPOSE open marketplace can be accessed via smartphones (Android, Windows Mobile, iOS), set-top boxes / TV sets (CE-HTML, HTML5) as well as tablets and desktop computers (Web browser, W3C widgets) of different manufacturers in order to use apps, rate and purchase them, as well as to get recommendations.

COMPOSE will provide a cross-platform presentation layer for the open marketplace. The frontend of the open marketplace will provide several graphical user interfaces (GUI) for a variety of terminal devices. Furthermore, the open marketplace GUI will be designed for users without any programming experience and will be implemented using Web technologies

(HTML5, JS, CSS). To access the open marketplace, users don't need to install any software clients; the open marketplace will be accessible from anywhere using a traditional Web browser.

6.2.4 App Discovery

Finding new apps involves time and effort in today's App stores. This issue is of particular interest for developers searching for sensor data, linked data and services in repositories of IoT platforms. COMPOSE open marketplace will ease this by offering smart search for interactive app and service objects, and service discovery. The marketplace will use app ratings, reviews, tags, and usage data to offer more than just a keyword-based search like today's stores. Therefore, COMPOSE will exploit modern recommendation algorithms, social and semantic technology, and interactive data visualization schemes to provide interactive, fun, and innovative ways to discover relevant apps.

6.2.5 App Distribution

As outlined in Section 6.2.3, COMPOSE open marketplace will support a variety of terminals ranging from smartphones and tablet devices to desktop computers and TV sets. This enables users to execute their apps on the specific device they would like to use the app on in a certain situation. The synchronization of app data allows for seamless switching from one device to another. COMPOSE will address future extensions for the open marketplace, for example a single app can also be used across different devices simultaneously in a multi-modal fashion forming a personal mobile cloud (such as playback on TV, input on a smartphone). The webinos project already researched and developed suitable solutions and concepts for such kind of extensions. So COMPOSE will elaborate these research findings regarding future reuse for the open marketplace.

6.2.6 Developers SDK and IDE

The COMPOSE project will support a development kit (SDK) and an integrated development environment (IDE) for developers that contains the necessary application programming interfaces (APIs) to implement services and apps for the open marketplace. Furthermore, COMPOSE provides developers with access to the features of the open marketplace for service and application discovery and deployment as well as analytics for app usage and distribution. The IDE will provide tools that support developers in virtualizing real-world objects such as camera, RFID, parking sensor, traffic sensor, etc. for deployment into the open marketplace. From the business perspective, the COMPOSE open marketplace will also let app developers define flexible feature-based access policies for their apps as a way to monetize their provided service. So developers can integrate flexible licensing with their apps. In addition to established licensing schemes, a CSP can define flexible licensing models for promotional and other purposes.

6.3 Baseline Technologies

The COMPOSE project aims to extend the latest research findings in cross-device and cross-platform app stores of Fraunhofer FOKUS. Fraunhofer developed the Megastore that enables application deployment across devices and platforms. Developers can provide their applications for a broad range of terminals including mobile phones, set-top boxes, TV sets and desktop computers. Beside cross-platform support, Megastore implements a variety of innovative app

deployment features. Furthermore Megastore supports a Graphical User Interface (GUI) for a variety of the before mentioned terminal devices. Megastore applies standard Web technologies such as JavaScript, CSS, HTML5 and CE-HTML for TV sets/set-top boxes. This cross-platform GUI is already deployed as webinos Apps Application Store for the webinos project [190]. The webinos Apps Application Store provides an easy way of publishing web-based and especially webinos applications (W3C Widgets) for mobile phones, desktop computers and set-top boxes. This is the same approach as COMPOSE will implement the open marketplace. Based on this work COMPOSE will develop the open marketplace concept. However under the premise that the open marketplace will be built up on several layers (service objects, composed services and applications) it should therefore be expected that much work remains to be undertaken in the requirements and specification tasks. Aspects for application sharing and deployment will be taken into account from research work that is already done in the domain of future app store models [171].

7 Comparison and Progress with Respect to Other European Projects

In this section, we highlight three European projects, namely SOA4All, iCore and webinos, which are particularly related to COMPOSE, and we describe how the work planned in COMPOSE complements and/or builds on the work of those projects.

7.1 SOA4All

SOA4All (http://soa4all.eu/) was a 3-year Large-Scale Integrating Project funded by the *EU Seventh Framework Programme*, under the *Service and Software Architectures, Infrastructures and Engineering* research area, running between March 2008 and February 2011. The project aimed at realizing a world where billions of parties are exposing and consuming services via advanced Web technology: the main objective of the project was to provide a comprehensive framework that integrates complementary and evolutionary technical advances (i.e., SOA, context management, Web principles, Web 2.0 and semantic technologies) into a coherent and domain-independent service delivery platform.

The SOA4All project developed **Linked Services**, a framework for dealing with services on the Web [165], following the Linked Data principles, which essentially dictate that every piece of data should be given an HTTP URI which, when looked up, should offer useful information using Web standards such as RDF for representing data and SPARQL to support online data queries. Importantly, data should be linked to other relevant resources thereby allowing humans and computers to discover additional information.

Linked Services build upon Linked Data and govern the way data sources and services are described, discovered, invoked, and integrated. In a nutshell, Linked Services are services that can consume and produce Linked Data and whose descriptions (such as their functionality and input/output data types) are also published as Linked Data. Linked Services can easily be integrated with existing Linked Data sources as both data and service interfaces are semantically described according to shared vocabularies. The data from the Web of Data can be directly used to invoke services. Combining Linked Services with Linked Data also enhances service discovery due to the provision of semantic descriptions that include links to/from other datasets and that are exposed using standards for data access and querying (esp. HTTP, RDF and SPARQL). For example, based on the types of data in an application's workspace, it is possible

to exploit the semantic description of services inputs in order to obtain only services that can process the available data.

The Linked Services approach is supported by several tools and the associated Linked Services models, primarily developed in SOA4All:

- **iServe** is an open registry for publishing and discovering Linked Services described using the following models. With Linked Service descriptions, the iServe registry transparently supports the discovery of heterogeneous services, without need to distinguish between WSDL services and Web APIs. To support discovery, iServe provides an extensible set of discovery algorithms that can be combined to create custom-tailored solutions depending on specific needs.
- **Minimal Service Model** (MSM), used in iServe as the core conceptual model, is a simple RDFS integration ontology based on the principle of minimal ontological commitment. Driven by Semantic Web best practices, the MSM builds upon existing vocabularies, most importantly SAWSDL [166]. MSM is able to capture the core semantics of both Web services and Web APIs in a common model, homogeneously supporting publication, discovery and invocation.
- **WSMO-Lite** is a lightweight ontology for describing four types of service semantics (functional, nonfunctional, behavioral and information model). It allows service providers to describe their service offerings so that a client can make an up-front decision on whether and how to consume the service's functionality.
- **hRESTS** is a microformat for marking up Web API documentation in HTML with the MSM structure of a service description, extended with properties specific to Web APIs such as the URIs of the resources and the HTTP methods allowed.
- **MicroWSMO** is a microformat that extends hRESTS with SAWSDL-like semantic annotations. In effect, hRESTS and MicroWSMO are the equivalents of WSDL and SAWSDL for Web APIs.
- **SWEET** [167] is a Web-based application that supports users in annotating API documentation on the Web by using the hRESTS microformat. The tool works in two stages: i) the HTML documentation of a given service is structured by identifying services, operations, inputs and outputs; and ii) the structure is annotated with service semantics in WSMO-Lite.
- **SOWER** [168] is a tool which is similar to SWEET except that it is designed to annotate WSDL files. The annotations are represented as SAWSDL annotations in the underlying WSDL.
- **OmniVoke** [169] is a tool that provides a simple interface which uses as input the URL of a Linked Service and some RDF input data, and returns RDF data as a response. OmniVoke provides: i) a single entry point for invoking any Linked Service; ii) a generic client for invoking third party Web APIs independently of their intricate details; and iii) a means for seamlessly integrating Web APIs and other Web services as Linked Data producers and consumers. A distinctive feature of OmniVoke is its generic support for invoking most types of Web APIs on the Web thanks to the use of semantic annotations captured in iServe.
- **SOA4All Studio** integrates iServe, SWEET, SOWER, and further components (e.g. for service composition) in a unified tool suite.

7.1.1 Progress Beyond SOA4All

COMPOSE will build on the set of tools and languages developed in SOA4All and extend them further in the following directions (summarized from the preceding sections of this deliverable):

- Linked services descriptions will be aligned and extended with sensor-specific description approaches to properly support the integration and management of objects.
- We shall envisage the inclusion of rules, e.g., expressed using RIF, to provide further capabilities such as services reactivity and the means to provide services' business logic in a declarative and decomposable manner.
- We shall devise novel information extraction and semantic annotation algorithms able to process the existing service documentation, as well as machine learning algorithms for the automated characterization of services.
- In contexts like the IoT, in which a large number of service objects and services will be available in a largely distributed manner, assuming a central registry for all the service description is not realistic. In COMPOSE, we will build upon iServe to develop a distributed discovery engine able to provide advanced discovery support over services and resources in highly distributed settings, esp. by employing distributed hash tables.
- We shall devise a service recommendation engine that will provide continuous proactive advertising of services to users and applications based on the knowledge about i) the actual application being executed or developed, ii) the services used in the past in similar applications and others available in general, iii) previous actions of users.
- To facilitate service composition by end users, we shall develop (semi)automatic algorithms for service composition, based on lightweight semantic descriptions for service objects and services. Using SOA4All models, we will retain a level of simplicity on services descriptions that can lead to good results with reasonable computational requirements for scalability reasons. We will also focus on supporting the deployment of the resulting workflows in a distributed fashion to maximize the scalability and reactivity of systems as necessary for dealing with real-time data coming out of sensors.

7.2 iCore

iCore (http://www.iot-icore.eu/) is a 3 years Integrated Project (IP) led by CREATE-NET and funded under call FP7-ICT-2011-7-1.3. The project addresses two main issues related to the design and deployment of services based on Internet of Things (IoT), which are:

1. The complexity arising from the huge number and heterogeneity of Internet-connected Objects that will be part of the Future Internet (FI);
2. The complexity inherent in the provisioning of services based on IoT, taking into account the different views of the different users/stakeholders (owners of objects & communication means) and ensuring the proper application provisioning and business integrity.

In order to tackle such challenges, iCore takes a "cognitive approach", aiming at designing and prototyping a cognitive management framework and associated functionalities for the Internet of Things. Such a framework will abstract the complexity deriving from the technological heterogeneity of the vast amounts of real world and digital objects, and will provide means to easily build applications based on IoT. The cognitive part of the iCore platform will provide self-management/awareness and learning functionalities, and will allow developers to easily share and re-use service components based on IoT [85].

To validate the proposed solutions, iCore addresses a number of application domains covering some of the most promising applications of IoT. In particular, it focuses on six main application areas [84]:

- "Smart Home: Social Sensors": This application scenario is focused on the functionalities and resources in the home and how they can collectively improve the

quality of life of residents.

- "Smart Office: Smart Meeting": Smart Office environments (to be seen as the business counterpart of Smart Home) aim at easing and improving the working activities and productivity of employees.
- "Smart Business: Consumer Moving Patterns": The Consumer moving pattern service can be used for tracking customers within supermarkets and malls.
- "Smart Transportation: Transportation: Supply Chain Management and Logistics": promotes the applicability of the iCore concepts in the Smart Supply Chain application domain.
- Smart Living: Health Care Prison Security: The objective of this application scenario is monitoring the security, controlling and tracking prisoners and their health condition in a prison hospital
- Smart City: IJKDijk: the use of IoT technologies for monitoring of dikes.

7.2.1 Progress Beyond iCore

COMPOSE will extend the work of iCore in the following directions:

- COMPOSE will focus on the provisioning of IoT-based services through an open Marketplace. The delivery of IoT-based services is therefore what characterizes most COMPOSE, and requires to address specific issues such as e.g., scalability, accountability, reliability, security, robustness, etc., which are not the primary focus of iCore.
- Computational and communication scalability: COMPOSE has the ambition to become the reference technological enabling platform for the various stakeholders involved in the development and deployment of IoT-based services. Scalability of both communications and computation will then be one the key aspects addressed in the design of the COMPOSE architecture, which will be based on a fully distributed architecture. Novel No-SQL technologies will be used to ensure critical reliability and scalability properties. This will be combined with a scalable P2P architecture to enable handling the dynamic churn rates of objects.
- Service description and representation: while for iCore the semantic reasoning is a way, among various others, to extract knowledge from Objects, in COMPOSE this will represent one of the main building blocks of the platform and of the standardization activities. COMPOSE will support both WSDL and Web APIs and will integrate these with linked data technologies, enabling services to process, produce, and be described as linked data. This will be complemented with semantic service discovery technologies which will provide advanced discovery support over Web APIs and WSDL services in highly distributed settings.
- Business perspective: the aim of COMPOSE is to provide a technological enabler for the various stakeholders involved in the development and provisioning of IoT-based services, starting from the community of developers. One of the objectives of the project is therefore to create the most adequate software tools and facilities (e.g., SDK) to unleash the potential and creativity of software designers and developers.
- Standardization: COMPOSE will be active in standardisation activities, with special focus on W3C (being a member of the consortium). We intend to build upon and extend existing and upcoming standards, many of which are being developed at W3C. These standards cover application programming interfaces (APIs) and communication protocols.

7.3 webinos

The purpose of the webinos project is to define and deliver an *Open Source platform*, which will enable web applications and services to be used and shared consistently and securely over a broad spectrum of connected devices. In practice, this is more than the APIs on individual devices. So, within a web-based scenario, the service should be able to potentially:

- Run across devices and domains
- Share preferences, status and synchronized information across multiple devices
- This applies to device features as well – especially to use smart phones as input devices
- Allow consistent access to developers
- Manage user authentication, cross device events, metrics and application distribution across devices.

The tagline of the webinos project is "Secure Web Operating System Application Delivery Environment", indicating that security (and also privacy) is a significant part of the project. Some of the functionality already exists in proprietary implementations – for example Sky go – which allows Sky subscribers to watch Sky TV on their designated iPhone and iPad devices. webinos aims to do this and a lot more for the web across platforms by creating truly distributed applications. In practice, this means for web based applications, webinos will allow for the following:

- Applications which make optimal use of the resources on the featured devices of TV, Automotive, Tablet, PC and Mobile
- Applications which interoperate over diverse device types
- Applications which can make use of services on other devices owned by the same person
- Applications which can make use of services on devices owned by other people
- Discovery mechanisms to find services, devices and people, on multiple interconnect technologies – even when they are not connected to the Internet
- Efficient communication mechanisms, that can pass messages over different physical bearers, can navigate firewalls, and make sensible use of scarce network resources
- Promiscuous communication mechanisms, that will find the best physical connection to pass messages over (not just IP)
- Strongly authenticated, communication mechanisms that work bi-directionally
- And finally, implementing distributed, user centric policy:
    - allowing the user to define what applications work on what devices,
    - to define what information is exposed to other services
    - and ensuring these capabilities are interoperable and transferable – ensuring a user stays in control of their devices and their applications.

To implement functionality, webinos architecture introduces three components: *the webinos web runtime*, the *webinos personal zone hub* (PZH) and the *webinos personal zone proxy* (PZP). A *webinos web runtime*, is a special type of browser which is capable rendering the latest JavaScript, HTML4/5 and CSS specifications. It is responsible for rendering the User Interface (UI) elements of the webinos application. A webinos WRT must be able to access the webinos root object from JavaScript. Via this root object the third party developer will be able to access the webinos functionality. A webinos WRT differs from a normal browser or web runtime in

that all extended JavaScript functions as well as some normal browser behaviours (such as XHR) must be mediated by the webinos policy enforcement layer. The webinos web runtime is tightly coupled to the PZP and presents environmental properties and critical events to the PZP.

In webinos, the *Personal Zone* is a conceptual construct, that is implemented on a distributed basis from a single Personal Zone Hub (PZH) and multiple Personal Zone Proxy (PZP)s. The webinos personal zone hub PZH provides a fixed entity to which all requests and messages can be sent to and routed on – a personal postbox as it were. The PZH is also the authoritative master copy of a number of critical data elements that are to synced between Personal Zone Proxy (PZP)s and Personal Zone Hub (PZH) – for example certificates. The PZH enables functionality like the creation of a User authentication service, secure session creation for transport of messages and synchronization between the PZP and PZH. The PZH also stores the policy files. The webinos personal zone proxy PZP acts in place of the Personal Zone hub, when there is no Internet access to the central server. The PZP fulfils most, if not all of the above functions described above, when there is no PZH access. In addition to the PZH proxy function, the PZP is responsible for all discovery using local hardware based bearers (Bluetooth, Zigbee, NFC etc.). Unlike the PZH, the PZH does not issue certificates and identities. For optimization reasons PZPs are capable of talking directly PZP-PZP, without routing messages through the PZH.

Thus, a *webinos application* has the following characteristics:

- A webinos application runs “on device” (where that device could also be internet addressable i.e. a server).
- A webinos application is packaged, as per packaging specifications, and executes within the WRT.
- A webinos application has its access to security sensitive capabilities, mediated by the active policy.
- A webinos application can expose some or all of its capability as a webinos service.
- An application developer is granted access to webinos capabilities via the webinos root JavaScript object.

An application developer programs and packages the application according to the webinos specification. They use the API to gain access to functionality. While much of the distributed capabilities of webinos are transparent to the developer, the developer is able to access functionality like discovery and service binding.

7.3.1 Progress beyond webinos

webinos supports sensor and actuator APIs for M2M applications. So the idea is obvious to enable webinos applications for COMPOSE as well as to integrate webinos-based sensors and actuators in the COMPOSE service objects concept. The webinos sensor and actuator APIs provide the following capabilities:

- The sensor and actuator services can be located in the user’s personal zone or be shared on the current network
- Sensors API: each sensor has a GeoPosition information, sensor’s configuration (timeout, rate, etc.), sensor’s value can be get periodically or when it changes
- Actuators API: support for switches, motors, thermostats, setValue function

Based on these APIs, COMPOSE addresses the integration of web-enabled devices, sensors and actuators in two layers. The first layer addresses service objects such as sensors and actuators

from the webinos M2M implementation. The second layer in COMPOSE reflects applications which should be device and platform independent. Both layers will be targeted by webinos. To conclude webinos applications as well as sensors and actuators will be adapted for the before mentioned two layers of the COMPOSE marketplace.

8 Conclusion

This deliverable expands on the COMPOSE project Description of Work by further specifying the relevant state of the art and our intended advances beyond that. This reflects the plans of the COMPOSE project at its very beginning.

9 References

[1] Rodrigo Rodrigues and Peter Druschel, Peer-to-Peer Systems. In Communications of the ACM Vol. 53 No. 10, Pages 72-82.

[2] Ken Birman, Gregory Chockler, Robbert van Renesse: Toward a cloud computing research agenda. SIGACT News 40(2).

[3] Roie Melamed, Idit Keidar: Araneola: A scalable reliable multicast system for dynamic environments. J. Parallel Distrib. Comput. 68(12).

[4] Gregory Chockler, Roie Melamed, Yoav Tock and Roman Vitenberg, Constructing Scalable overlays for pub-sub with many topics, in PODC 2007.

[5] Gregory Chockler, Roie Melamed, Yoav Tockand Roman Vitenberg, SpiderCaast, a scalable interest aware overlays for topic-based pub/sub communication, in DEBS 2007.

[6] Sarunas Girdzijauskas, Gregory Chockler, Ymir Vigfusson, Yoav Tock and Roie Melamed, Magnet: Practical subscriptions clustering for Internet-scale publish/subscribe. In DEBS 2010.

[7] G. Chockler, I. Keidar, and R. Vitenberg, “Group communication specifications: a comprehensive study,” ACM Computing Surveys, vol. 33, no. 4, pp. 427–469, 2001.

[8] K. Birman and T. Joseph, “Exploiting virtual synchrony in distributed systems,” in SOSP’87, 1987.

[9] Distribution and Consistency Services (DCS), http://www.research.ibm.com/haifa/projects/systems/dcs/index.html

[10] R. van Renesse, Y. Minsky, and M. Hayden, “A gossip-style failure detection service,” in Middleware’98: Proc. of the IFIP Int’l Conf. on Distributed Systems Platforms and Open Distributed Processing, 1998, pp. 55–70.

[11] A. Lakshman and P. Malik, “Cassandra: a decentralized structured storage system,” SIGOPS Oper. Syst. Rev., vol. 44, no. 2, pp. 35–40, Apr. 2010.

[12] A. Ganesh, A. M. Kermarrec, and L. Massoulie, “Peer-to-Peer Membership Management for Gossip-Based Protocols” IEEE Trans. Comput., vol. 52, no. 2, pp. 139–149, 2003.

[13] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, " Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", in ACM SIGCOMM 2001, San Deigo, CA, August 2001.

[14] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems", IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany: 329–350.

[15] Indranil Gupta , Ken Birman , Prakash Linga , Al Demers , Robbert van Renesse, "Kelips: Building an efficient and stable peer-to-peer DHT through increased memory and background overhead", 2nd International Workshop on Peer-to-Peer Systems, 2003.

[16] A. Allavena, A. Demers, and J. E. Hopcroft, "Correctness of a gossip based membership protocol," in PODC '05: Proc. of the annual ACM Symp. on Principles of Distributed Computing, 2005, pp. 292–301

[17] I. Gupta, T. D. Chandra, and G. S. Goldszmidt, "On scalable and efficient distributed failure detectors," in PODC'01: Proc. of the annual ACM Symp. on Principles of Distributed Computing, 2001, pp. 170–179.

[18] A. J. Ganesh, A. M. Kermarrec, and L. Massouli´e, "HiScamp: self-organizing hierarchical membership protocol," in Proceedings of the 10th workshop on ACM SIGOPS European workshop, ser. EW 10, 2002, pp. 133–139.

[19] J. Cowling, D. R. K. Ports, B. Liskov, R. A. Popa, and A. Gaikwad, "Census: Location-Aware Membership Management for Large-Scale Distributed Systems," in USENIX, 2009.

[20] J. Varma, C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Scalable, fault tolerant membership for MPI tasks on HPC systems," in ICS'06: Proc. of the 20th Annual Int'l Conf. on Supercomputing, 2006

[21] Pettey Christy and Rob van der Meulen. Key Trends for the Mobile Industry to Be Examined at Gartner Symposium/ITxpo 2012. In Gartner, Inc. NYSE, IT. September 2012.

[22] Jain Ashish. Apps Marketplaces and the Telecom Value Chain. In IEEE Wireless Communications, Industry Perspectives. August 2011.

[23] webinos project. Funded by the European Commission within the Framework Programme Seven (FP7). Project No. 257103. Online Available: http://www.webinos.org, November 2012.

[24] Nicolas Seriot. iPhone Privacy. In Proceedings of Black Hat DC 2010. Arlington, Virginia, USA. August 2010.

[25] Tyler Shields. Blackberry Mobile Spyware – The Monkey Steals the Berries. February 2010.

[26] AVANTSSAR: Automated Validation of Trust and Security of Service Oriented Architectures, 01.01.200831.12.2010. FP7-ICT-2007-1, Project No. 216471.

[27] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-flow integrity principles, implementations, and applications. ACM Transactions on Information and System Security, 13(1):1–40, Oct. 2009.

[28] A. Armando, R. Carbone, and L. Compagna. LTL Model Checking for Security Protocols. pages 403–429, 2009.

[29] M. Asim (ed.). D2.3 - Models and methodologies for implementing Security by-Contract for services, July 2012.

[30] A. Askarov and A. Sabelfeld. Tight enforcement of information-release policies for dynamic languages. In Proceedings of the 2009 22nd IEEE Computer Security Foundations Symposium, CSF '09, pages 43–59, Washington, DC, USA, 2009. IEEE Computer Society.

[31] G. Baldini. iCore - D2.2 Security requirements for the iCore cognitive management and control framework, May 2012.

[32] D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar. The software model checker Blast. Int. J. Softw. Tools Technol. Transfer, 9(5-6):505-525, 2007.

[33] D. Beyer, T. A. Henzinger, and G. Thóduloz. Configurable software vereification: Concretizing the convergence of model checking and program analysis. In Proc. CAV, LNCS 4590, pages 504–518. Springer, 2007.

[34] D. Beyer, T. A. Henzinger, and G. Thóduloz. Program analysis with dynamic precision adjustment. In Proc. ASE, pages 29–38. IEEE, 2008.

[35] K. Bhargavan, C. Fournet, A. D. Gordon, and R. Pucella. Tulafale: A security tool for web services. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, editors, Formal Methods for Components and Objects, Second International Symposium, FMCO 2003, Leiden, The Netherlands, November 4-7, 2003, Revised Lectures, volume 3188 of Lecture Notes in Computer Science, pages 197–222. Springer, 2003.

[36] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In 14th IEEE Computer Security Foundations Workshop (CSFW14), pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.

[37] R. Carbone, M. Minea, S. A. Mödersheim, S. E. Ponta, M. Turuani, and L. Vigano. Towards Formal Validation of Trust and Security in the Internet of Services. volume 6656 of LNCS, pages 193–207. Springer, Heidelberg, 2011.

[38] I. Carreras, L. Bassbouss, D. Linner, H. Pfeffer, V. Simon, E. Varga, D. Schreckling, J. Huusko, and H. Rivas. Bionets: Self evolving services in opportunistic networking environments. In Proceedings of the 4th International Conference on Bio-Inspired Models of Network, Information and Computing Systems (BIONETICS 2009). ICST, December 2009.

[39] P. P. Chan, L. C. Hui, and S. M. Yiu. Droidchecker: analyzing android applications for capability leak. In Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks, WISEC ’12, pages 125–136, New York, NY, USA, 2012. ACM.

[40] Y. Chevalier, M. Mekki, and M. Rusinowitch. Automatic composition of services with security policies. pages 529–537, Los Alamitos, USA, 2008.

[41] S. Chong, J. Liu, A. C. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng. Secure web applications via automatic partitioning. In Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, SOSP ’07, pages 31–44, New York, NY, USA, 2007. ACM.

[42] R. Chugh, J. a. Meister, R. Jhala, and S. Lerner. Staged information flow for javascript. ACM SIGPLAN Notices, 44(6):50, May 2009.

[43] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexampleguided abstraction refinement for symbolic model checking. J. ACM, 50(5):752–794, 2003.

[44] D. Schreckling. Adaptive Security in BIONETS. BIONETS (IST-20042.3.4 FP6-027748), Deliverable D4.4, Feburary 2009.

[45] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In Proceedings of OSDI 2010, October 2010.

[46] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In CCS ’09: Proceedings of the 16th ACM conference on Computer and communications security, pages 235–245, New York, NY, USA, 2009. ACM.

[47] U. Erlingsson. The inlined reference monitor approach to security policy enforcement. PhD thesis, Cornell University, Ithaca, NY, USA, January 2004. Adviser-Schneider, Fred B.

[48] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin. Permission re-delegation: attacks and defenses. In Proceedings of the 20th USENIX conference on Security, SEC'11, pages 22–22, Berkeley, CA, USA, 2011. USENIX Association.

[49] Fraunhofer FOKUS (ed.). D3.1: webinos phase I architecture and components, June 2011.

[50] Fraunhofer FOKUS (ed.). Webinos D3.5: Webinos Phase 1 Security Framework, June 2011.

[51] M. Grace, Y. Zhou, Z. Wang, and X. Jiang. Systematic Detection of Capability Leaks in Stock Android Smartphones. In Proceedings of the 19th Network and Distributed System Security Symposium (NDSS), Feb. 2012.

[52] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks, WISEC '12, pages 101–112, New York, NY, USA, 2012. ACM.

[53] D. Huang, X. Zhang, M. Kang, and J. Luo. Mobicloud: Building secure cloud framework for mobile computing and communication. In Proceedings of the 2010 Fifth IEEE International Symposium on Service Oriented System Engineering, SOSE '10, pages 27–34, Washington, DC, USA, 2010. IEEE Computer Society.

[54] J. Liebig, A. von Rhein, C. K¨stner, S. Apel, J. D¨rre, and C. Lengauer. Large-Scale Variability-Aware Type Checking and Dataflow Analysis. Technical Report MIP-1212, University of Passau, Nov. 2012.

[55] J. Ligatti, L. Bauer, and D. Walker. Edit automata: Enforcement mechanisms for run-time security policies. International Journal of Information Security, 4:2–16, 2005.

[56] J. Ligatti and S. Reddy. A theory of runtime enforcement, with results. In Proceedings of the European Symposium on Research in Computer Security (ESORICS), Sept. 2010.

[57] J. Magazinius, R. Russo, and A. Sabelfeld. On-the-fly inlining of dynamic security monitors. In In Proc. IFIP International Information Security Conference, 2010.

[58] M. Brunato and P. Dini and L. Czap and L. Dora and J. Golic and G. Horvath and F. Martinelli and M. Petrocchi and A. Faschinbauer and D. Schreckling. Security in BIONETS. BIONETS (IST-2004-2.3.4 FP6-027748), Deliverable D4.6, Feburary 2010.

[59] A. Miede, T. Ackermann, N. Repp, D. F. Abawi, R. Steinmetz, and P. Buxmann. Attacks on the internet of services : The security impact of crossorganizational service-based collaboration. In Tagungsband Multikonferenz Wirtschaftsinformatik (MKWI) 2010, 2010.

[60] S. Mödersheim and L. Vigano. Secure pseudonymous channels. volume 5789 of LNCS, pages 337–354. Springer, Heidelberg, 2009.

[61] B. Monate and J. Signoles. Slicing for security of code. In Proceedings of the 1st international conference on Trusted Computing and Trust in Information Technologies: Trusted Computing - Challenges and Applications, Trust '08, pages 133–142, Berlin, Heidelberg, 2008. Springer-Verlag.

[62] G. C. Necula. Proof-carrying code. In Proc. POPL, pages 106–119. ACM, 1997.

[63] OMTP Limited. BONDI Architecture and Security Requirements, July 2009.

[64] OMTP Limited. BONDI Architecture and Security Requirements (Appendices), Jan. 2010.

[65] M. Ongtang, K. Butler, and P. McDaniel. Porscha: Policy oriented secure content handling in android. In Proceedings of the 26th Annual Computer Security Applications Conference, December 2010.

[66] M. Ongtang, S. E. McLaughlin, W. Enck, and P. D. McDaniel. Semantically Rich Application-Centric Security in Android. In ACSAC, pages 340–349. IEEE Computer Society, 2009.

[67] Organization for the Advancement of Structured Information Standards (OASIS). Web Services Business Process Execution Language (WS-BPEL) Version 2.0, Apr. 2007.

[68] L. Pino and G. Spanoudakis. Finding secure compositions of software services: Towards a pattern based approach. In New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on, pages 1 –5, May 2012.

[69] M. Rossi (ed.). D3.3 Initial IoT Protocol Suite definition, Apr. 2012.

[70] A. Sabelfeld and A. C. Myers. Language-based information-flow security. IEEE Journal on Selected Areas in Communications, 21(1):5–19, 2003.

[71] A. Sabetta, M. Bezzi, and S. P. Kaluvuri. Towards a development environment to orchestrate services with certified security properties. In Proceedings of the 2012 ACM SIGSOFT symposium on Industry Day, Industry Day '12, pages 1–4, New York, NY, USA, 2012. ACM.

[72] D. Schreckling and J. Posegga. Adaptive Security Architectures for Global Sensing Applications. Electronic Communications of the EASST, 37, 2011.

[73] D. Schreckling, J. Posegga, J. Köstler, and M. Schaff. Kynoid: Real-Time Enforcement of Fine-Grained, User-Defined, and Data-Centric Security Policies for Android. In Proceedings of WISTP'12, LNCS. Springer Verlag, June 2012.

[74] V. C. Sreedhar, G. R. Gao, and Y.-F. Lee. A new framework for exhaustive and incremental data flow analysis using dj graphs. SIGPLAN Not., 31(5):278–290, May 1996.

[75] UIBK. SOA4All Reference Architecture Specification, Mar. 2009.

[76] M. Weiser. Program slicing. In Proceedings of the 5th international conference on Software engineering, ICSE '81, pages 439–449, Piscataway, NJ, USA, 1981. IEEE Press.

[77] M. D. Weiser. Program slices: formal, psychological, and practical investigations of an automatic program abstraction method. PhD thesis, Ann Arbor, MI, USA, 1979. AAI8007856.

[78] Wholesale Applications Community. WAC 2.0 Core Specification: Widget Security and Privacy. http://members.wholesaleappcommunity.com/corespec/widget-securityprivacy.html, July 2011.

[79] A. Yautsiukhin (ed.). D3.3 - Run-time Secure Composition and Adaptation Realisation Techniques, Dec. 2012.

[80] J.-S. Yur, B. G. Ryder, and W. A. Landi. An incremental flow- and contextsensitive pointer aliasing analysis. In Proceedings of the 21st international conference on Software engineering, ICSE '99, pages 442–451, New York, NY, USA, 1999. ACM.

[81] Robbert Van Renesse , Kenneth P. Birman , Werner Vogels, "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management and Data Mining" in ACM Transactions on Computer Systems, 2001.

[82] Joshua B. Leners, Hao Wu, Wei-Lun Hung , Marcos K. Aguilera, Michael Walfish, " Detecting failures in distributed systems with the FALCON spy network", in SOSP '11

[83] Manku, G. S., Bawa, M., and Raghavan, P., "Symphony: Distributed Hashing in a Small World", in Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS), pp. 127–140, March 2003.

[84] iCore deliverable D.1.1, "D1.1 Use cases definitions and scenarios", Eds. Roberto Minerva, Frank Berkers, Apr. 2012

[85] iCore deliverable D.2.1, "D2.1 Technical Requirements for the iCore Cognitive Management and Control Framework", Apr. 2012

[86] Ishii, H., and Ullmer, B. Tangible bits: Towards seamless interfaces between people, bits and atoms. In CHI (1997), pp. 234-241.

[87] Major magazine on practical do-it-yourself (DIY) projects. See: http://makezine.com/

[88] Community-operated spaces enabling amateurs to work on their projects collectively. See http://hackerspaces.org/

[89] Gershenfeld, N. FAB: The Coming Revolution on Your Desktop - From Personal Computers to Personal Fabrication. Basic Books, 2005.

[90] An electronic sketchbook for prototyping multimedia applications. See: http://processing.org.

[91] An open-source hardware platform for physical computing. See: http://arduino.cc

[92] Akyildiz, I., Su, W., Sankarasubramaniam, Y., and Cayirci, E. Wireless sensor networks: a survey. Computer Networks 38, 4 (2002), 393-422.

[93] Wang, M.-M., Cao, J.-N., Li, J., and Dasi, S. K. Middleware for wireless sensor networks: A survey. Journal of Computer Science and Technology 23, 3 (May 2008), 305-326.

[94] Yick, J., Mukherjee, B., and Ghosal, D. Wireless sensor network survey. Computer Networks 52, 12 (2008), 2292-2330.

[95] Tolle, G., Polastre, J., Szewczyk, R., Culler, D., Turner, N., Tu, K., S., B., Dawson, T., Buonadonna, P., Gay, D., et al. A macroscope in the redwoods. In Proceedings of the 3rd international conference on Embedded networked sensor systems (2005), ACM New York, NY, USA, pp. 51-63.

[96] Ye, W., Heidemann, J., and Estrin, D. An energy-e client MAC protocol for wireless sensor networks. In Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM) (2002), vol. 3, pp. 1567-1576.

[97] Van Dam, T., and Langendoen, K. An adaptive energy-e client MAC protocol for wireless sensor networks. In Proceedings of the 1st international conference on embedded networked sensor systems (Los Angeles, California, USA, 2003), SenSys '03, ACM, pp. 171-180. ACM ID: 958512.

[98] Demirkol, I., Ersoy, C., and Alagoz, F. MAC protocols for wireless sensor networks: a survey. Communications Magazine, IEEE 44, 4 (2006), 115-121.

[99] Bai, L., Dick, R., and Dinda, P. Archetype-based design: Sensor network programming for application experts, not just programming experts. In Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN 2009) (2009).

[100] Mainland, G., Morrisett, G., and Welsh, M. Flask: staged functional programming for sensor networks. In ICFP '08: Proceeding of the 13th ACM SIGPLAN international conference on functional programming (New York, NY, USA, 2008), ACM, pp. 335-346.

[101] Woo, A., Seth, S., Olson, T., Liu, J., and Zhao, F. A spreadsheet approach to programming and managing sensor networks. In Proceedings of the fifth international conference on information processing in sensor networks (IPSN) (New York, NY, USA, 2006), ACM, pp. 424-431.

[102] Srivastava, M., Hansen, M., Burke, J., Parker, A., Reddy, S., Saurabh, G., Allman, M., Paxson, V., and Estrin, D. Wireless urban sensing systems. CENS Technical Report 65, University of California Los Angeles, April 2006.

[103] Dunkels, A., and Vasseur, J. IP for Smart Objects Alliance. Internet Protocol for Smart Objects (IPSO) Alliance White paper, September 2008.

[104] Dunkels, A. Full TCP/IP for 8-bit architectures. In Proceedings of the 1st international conference on mobile systems, applications and services (2003), ACM, p. 98.

[105] Dunkels, A., Gronvall, B., and Voigt, T. Contiki - a lightweight and flexible operating system for tiny networked sensors. In Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I) (Tampa, Florida, USA, 2004).

[106] Durvy, M., Abeille, J., Wetterwald, P., O'Flynn, C., Leverett, B., Gnoske, E., Vidales, M., Mulligan, G., Tsiftes, N., Finne, N., and Dunkels, A. Making Sensor Networks IPv6 Ready. In Proceedings of the Sixth ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008), poster session (Raleigh, North Carolina, USA, Nov. 2008).

[107] Hui, J. W., and Culler, D. E. IP is dead, long live IP for wireless sensor networks. In SenSys'08: Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (New York,NY, USA, 2008), ACM, pp. 15-28.

[108] Priyantha, N. B., Kansal, A., Goraczko, M., and Zhao, F. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In SenSys '08: Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (New York, NY, USA, 2008), ACM, pp. 253-266.

[109] Berners-Lee, T. Information management: A proposal. Online at http://www.w3.org/History/1989/proposal.html

[110] O'Reilly, T. What is web 2.0. Online at http://oreilly.com/web2/archive/what-is-web-20.html

[111] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7 (OSDI '06), Vol. 7. USENIX Association, Berkeley, CA, USA, 15-15.

[112] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall and W. Vogels, “Dynamo: Amazon's Highly Available Key-Value Store”, in the Proceedings of the 21st ACM Symposium on Operating Systems Principles, Stevenson, WA, October 2007.

[113] Apache Cassandra. http://cassandra.apache.org

[114] Apache HBase. http://hbase.apache.org

[115] Apache CouchDB. http://couchdb.apache.org

[116] MongoDB. http://www.mongodb.org

[117] CouchBase Server. http://www.couchbase.com

[118] Windows Azure Platform, http://www.microsoft.com/azure/default.mspx

[119] Aneka, http://www.manjrasoft.com/products.html

[120] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation. Berkeley, CA, USA: USENIX Association, 2004, p. 10.

[121] Apache Hadoop, http://hadoop.apache.org

[122] Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. European Conference on Computer Systems (EuroSys), Lisbon, Portugal, March21-23, 2007

[123] Y. Becerra, V. Beltran, D. Carrera, M. González, J. Torres and E. Ayguadé. Speeding Up Distributed MapReduce Applications Using Hardware Accelerators. In the 38th International Conference on Parallel Processing (ICPP). Vienna, Austria. September 2009.

[124] J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguadé, M. Steinder, I. Whalley. Performance-Driven Task Co-Scheduling for MapReduce Environments. In the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS2010). April 19-23th, 2010, Osaka, Japan.

[125] J. Polo, D. Carrera, Y. Becerra, V. Beltran, J. Torres, E. Ayguadé. Performance Management of Accelerated MapReduce Workloads in Heterogeneous Clusters. Accelerators. In the 39th International Conference on Parallel Processing (ICPP2010). San Diego, CA. September 2010.

[126] D. Carrera, M. Steinder, I. Whalley, J. Torres and E. Ayguadé. Utility-based Placement of Dynamic Web Applications with Fairness Goals. In 11th IEEE/IFIP Network Operations and Management Symposium (NOMS 2008), April 7-11, 2008.

[127] D. Carrera, M. Steinder, I. Whalley, J. Torres and E. Ayguadé. Enabling resource sharing between transactional and batch workloads using dynamic application placement. In the ACM/IFIP/USENIX 9th International Middleware Conference (Middleware 2008), Dec. 2008.

[128] STORM: Distributed and fault-tolerant realtime computation. http://storm-project.net/

[129] S4: Distributed Stream Computing Platform. 2010 IEEE International Conference on Data Mining Workshops (ICDMW). Dec. 2010. Neumeyer, L., Robbins, B. ; Nair, A. ; Kesari, A. pp 170-177.

[130] memcached - a distributed memory object caching system. http://memcached.org

[131] B. G. Fitch, A. Rayshubskiy, M. C. Pitman, T. J. C. Ward, and R. S. Germain. 2009. Using the Active Storage Fabrics model to address petascale storage challenges. In Proceedings of the 4th Annual Workshop on Petascale Data Storage (PDSW '09). ACM, New York, NY, USA, 47-54.

[132] Apache Pig. http://pig.apache.org/

[133] Scala Framework. http://www.scala-lang.org/node/273

[134] H. Pfeffer, D. Linner, and S. Steglich: Dynamic Adaptation of Workflow Based Service Compositions, In Proceedings of the 4th international Conference on intelligent Computing: Advanced intelligent Computing theories and Applications - with Aspects of theoretical and Methodological Issues (Shanghai, China, September 15 - 18, 2008). D. Huang, D. C. Wunsch, D. S. Levine, and K. Jo, Eds. Lecture Notes In Computer Science, vol. 5226. Springer-Verlag, Berlin, Heidelberg, 763-774, 2008.

[135] Girish B. Chafle, Sunil Chandra, Vijay Mann, and Mangala Gowri Nanda. Decentralized orchestration of composite web services. pages 134–143, 2004.

[136] H. Pfeffer, D. Linner, and S. Steglich: Modeling and Controlling Dynamic Service Compositions, In Proceedings of the 2008 the Third international Multi-Conference on Computing in the Global information Technology (Iccgi 2008) - Volume 00 (July 27 - August 01, 2008). ICCGI. IEEE Computer Society, Washington, DC, 210-216, 2008.

[137] Bin Liu, Shaofei Wu, and Tao Lv. A distributed workflow model based on web service. In International Conference on Intelligent Computation Technology and Automation, 2008.

[138] I. Stoica et al., “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,” Proc. ACM SIGCOMMC. Pedrinaci, J. Domingue, and A. Sheth.

Handbook on Semantic Web Technologies, volume Semantic Web Applications, chapter Semantic Web Services. Springer, 2010.

[139] D. Fensel, H. Lausen, A. Polleres, J. de Bruijn, M. Stollberg, D. Roman, and J. Domingue. Enabling Semantic Web Services: The Web Service Modeling Ontology. Springer, 2007.

[140] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. Member submission, W3C, 2004. W3C Member Submission 22 November 2004.

[141] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, and K. Verma. Web service semantics – WSDL-S. http://www.w3.org/Submission/WSDL-S/, November 2005. W3C Member Submission.

[142] http://www.w3.org/Submission/wadl/

[143] L. Richardson and S. Ruby. RESTful Web Services. O'Reilly Media, Inc., May 2007.

[144] J. Kopecky, K. Gomadam, and T. Vitvar. hRESTS: an HTML Microformat for Describing RESTful Web Services. In The 2008 IEEE/WIC/ACM International Conference on Web Intelligence (WI2008), Sydney, Australia, November 2008. IEEE CS Press.

[145] http://www.w3.org/standards/semanticweb/data

[146] K. Fujii and T. Suda, "Dynamic Service Composition Using Semantic Information", Proc. of the 2nd ACM International Conference on Service Oriented Computing (ICSOC '04), November 2004.

[147] C. Pedrinaci and J. Domingue. Toward the Next Wave of Services: Linked Services for the Web of Data. Journal of Universal Computer Science, 16(13):1694–1719, 2010.

[148] L. C. A. Hately, C. von Riegen, and T. Rogers. UDDI Specification Version 3.0.2, 2004.

[149] T. Pilioura and A. Tsalgatidou. Unified Publication and Discovery of Semantic Web Services. ACM Trans. Web, 3(3):1–44, 2009.

[150] T. Kawamura, J.-A. D. Blasio, T. Hasegawa, M. Paolucci, and K. P. Sycara. Public Deployment of Semantic Service Matchmaker with UDDI Business Registry. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, International Semantic Web Conference, volume 3298 of Lecture Notes in Computer Science, pages 752–766. Springer, 2004.

[151] E. Toch, A. Gal, I. Reinhartz-Berger, and D. Dori. A semantic approach to approximate service retrieval. ACM Transactions on Internet Technology (TOIT), 8(1):2, 2007.

[152] C. Kiefer and A. Bernstein. The Creation and Evaluation of iSPARQL Strategies for Matchmaking. In Proceedings of the 5th European Semantic Web Conference (ESWC), Lecture Notes in Computer Science. Springer, February 2008.

[153] B. Adida, M. Birbeck, S. McCarron, and S. Pemberton. RDFa in XHTML: Syntax and Processing. http://www.w3.org/TR/rdfa-syntax/, October 2008.

[154] S. Dustdar and W. Schreiner, "A survey on web services composition," Int. J. Web Grid Serv., vol. 1, no. 1, pp. 1–30, 2005.

[155] M. ter Beek, A. Bucchiarone, and S. Gnesi, "A Survey on Service Composition Approaches: From Industrial Standards to Formal Methods," Technical Report 2006-TR-15, Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche, 2006.

[156] K. Fujii and T. Suda, "Semantics-based Dynamic Web Service Composition," the International Journal of Cooperative Information Systems (IJCIS), special issue on Service-Oriented Computing, Vol. 15, No. 3, pp. 293-324, Sep. 2006.

[157] Van der Aalst, M. Dumas, and A. ter Hofstede, "Web service composition languages: old wine in New bottles?," Euromicro Conference, 2003. Proceedings. 29th, pp. 298–305, Sep. 2003.

[158] G. Di Lorenzo, H. Hacid, H.-Y. Paik, and B. Benatallah, "Data integration in mashups," SIGMOD Rec., vol. 38, no. 1, p. 59, Jun. 2009.

[159] Mangala Gowri Nanda, Satish Chandra, and Vivek Sarkar. Decentralizing execution of composite web services. SIGPLAN Not., 39(10):170–187, 2004.

[160] Ronan Barrett and Claus Pahl. Model driven design of distribution patterns for web service compositions. page 23, 2006.

[161] http://chicagocrime.org

[162] Ericsson. "More than 50 billion connected devices". Ericsson white paper 284 23-3149 Uen | February 2011. http://www.ericsson.com/res/docs/whitepapers/wp-50-billions.pdf

[163] Stirbu, V.; , "Towards a RESTful Plug and Play Experience in the Web of Things," Semantic Computing, 2008 IEEE International Conference on , vol., no., pp.512-517, 4-7 Aug. 2008. doi: 10.1109/ICSC.2008.51

[164] Mattern, F. and Floerkemeier, C. 2010. "From the internet of computers to the internet of things. In "From active data management to event-based systems and more". Springer-Verlag, Berlin, Heidelberg 242-259.

[165] M. Maleshkova, G.A., Rey, A. Simov, A., B. Renie, D. Liu, "Service Provisioning Platform Second Prototype," Deliverable D2.1.4 of the project SOA4All, 2010, available at http://soa4all.eu/file-upload.html?func=startdown&id=229

[166] http://w3.org/TR/sawsdl

[167] M. Maleshkova, C. Pedrinaci, J. and Domingue, J. "Semantic Annotation of Web APIs with SWEET," 6th Workshop on Scripting and Development for the Semantic Web, colocated with the Extended Semantic Web Conference 2010, Heraklion, Greece.

[168] Maleshkova, M., Pedrinaci, C., Domingue, J.: "Supporting the Creation of Semantic RESTful Service Descriptions" 8th International Semantic Web Conference, Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2) workshop, 2009.

[169] N. Li, C. Pedrinaci, M. Maleshkova, J. Kopecky, and J. Domingue, "OmniVoke: A Framework for Automating the Invocation of Web APIs," 5th IEEE International Conference on Semantic Computing (ICSC), pp.39-46, 18-21 Sept. 2011.

[170] Miller, T, Monaghan, C.: "Apple Press Info: App Store Tops 40 Billion Downloads with Almost Half in 2012". http://www.apple.com/pr/library/2013/01/07App-Store-Tops-40-Billion-Downloads-with-Almost-Half-in-2012.html

[171] Ahmet, Z., Holmquist, L.: "Sharing mobile services: beyond the App store model" MobileHCI '10 Proceedings of the 12th international conference on Human computer interaction with mobile devices and services, Pages 383-384, 2010. doi: 10.1145/1851600.1851676

[172] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, V. Sundaresan, "SOOT – A Java Bytecode Optimization Framework", In Proceedings of the 1999 Conference of the Centre for Advanced Studies on Collaborative Research, CASCON'99, Mississauga, Ontario, Canada, IBM Press, 1999.

[173] C. Lattner and V. Adve, “LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation”, Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO’04), Palo Alto, California, 2004.

[174] D. Beyer, M. E. Keremoglu, “CPAchecker: A Tool for Configurable Software Verification”, *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011, Snowbird, UT, July 14-20)*, LNCS 6806, pages 184-190, 2011. Springer-Verlag, Heidelberg.

[175] http://wala.sourceforge.net/wiki/index.php/Main_Page

[176] A. C. Myers, B. Liskov, “A Decentralized Model for Information Flow Control”. In Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP), pages 129–142, Saint-Malo, France, October 1997.

[177] D. Schreckling, J. Köstler and M. Schaff. “Kynoid: Real-time enforcement of fine-grained, user-defined, and data-centric security policies for Android.” In Information Security Technical Report (ISTR), 2013.

[178] D. Schreckling, J. Posegga and D. Hausknecht. “Constroid: Data-Centric Access Control for Android.” In Proceedings of the 27th Symposium on Applied Computing (SAC): Computer Security Track, 2012.

[179] Erl, T. (2005). Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[180] http://w3.org/TR/wsdl20

[181] http://w3.org/TR/soap12

[182] Gomadam, K., Ranabahu, A., and Sheth, A. (2010). SA-REST: Semantic Annotation of Web Resources. W3C member submission, available at http://www.w3.org/Submission/SA-REST/

[183] Daniel Oberle, Alistair Barros, Uwe Kylau, Steffen Heinzl, A unified description language for human to automated services, Information Systems, Volume 38, Issue 1, March 2013, Pages 155-181, ISSN 0306-4379, 10.1016/j.is.2012.06.004.

[184] http://linked-usdl.org/

[185] Karthik Gomadam, Ajith Ranabahu, Meenakshi Nagarajan, Amit. P. Sheth and Kunal Verma, 'A Faceted Classification Based Approach to Search and Rank Web APIs', In Proceedings of 6th IEEE International Conference on Web Services (ICWS), 177-184, Beijing, China, Sep. 2008

[186] Pedrinaci, C., Liu, D., Maleshkova, M., Lambert, D., Kopecky, J., and Domingue, J. (2010). iServe: a Linked Services Publishing Platform. In Proceedings of 1st International Workshop on Ontology Repositories and Editors for the Semantic Web, ORES 2010, colocated with 7th ESWC.

[187] http://www.w3.org/2005/Incubator/ssn/

[188] http://www.w3.org/TR/rif-overview/

[189] Ngoc Chan, N., Gaaloul, W., "Collaborative Filtering Technique for Web Service Recommendation Based on User-Operation Combination", On the Move to Meaningful Internet Systems, 2010.

[190] webinos Apps Application Store. http://apps.webinos.org/developer/