EUROPEAN JOURNAL OF LAW AND TECHNOLOGY, VOL 4, NO 2 (2013)

Security in Pervasive Applications: A Survey

Chiara Bodei, [1] Pierpaolo Degano, [2] Gian-Luigi Ferrari,[3] Letterio Galletta, [4] Gianluca Mezzetti [5]

Cite as: Bodei, C., Degano, P., Ferrari, G-L., Galletta, L., & Mezzetti, G., Security in Pervasive Applications: A Survey, in European Journal of Law and Technology, Vol. 4, No. 2, 2013.

ABSTRACT

We survey some critical issues arising in pervasive applications, in particular the interplay between context-awareness and security. We shall outline the techniques adapted for guaranteeing applications to securely behave in the digital environment they are part of. [6]

1. INTRODUCTION

Modern software systems are designed to operate *always* and *everywhere*. Internet is *de facto* becoming the infrastructure providing us with wired or wireless access points for our digitally instrumented life. A great variety of activities and tasks performed by individuals are mediated, supported and affected by different heterogeneous digital systems that in turn often cooperate each other without human intervention. These digital entities can be any combination of hardware devices and software pieces or even people, and their activities can change the physical and the virtual environment where they are plugged in. For example, in a smart house, a sensor can proactively switch on the heater to regulate the temperature.

An emerging line is therefore integrating these entities into an active and highly dynamic digital environment that hosts end-users, continuously interacting with it. Consequently, the digital environment assumes the form of a communication infrastructure, through which its entities can interact each other in a loosely coupled manner, and they can access resources of different kinds, e.g., local or remote, private or shared, data or programs, devices or services. The name *pervasive computing* is usually adopted to denote these phenomena.

The structure of pervasive applications is therefore subject to continuous changes, which however could compromise the correct behaviour of applications and break the guarantees on their non-functional requirements. For instance in the case of applications manipulating classified information, it is essential among other things that this data maintain integrity, privacy and security.

Effective mechanisms are thus required to *adapt* software to the new added functionalities and to changes of the operational environment, namely the context in which the application is plugged in. Also, appropriate security models are mandatory to ensure secured information is not shared

intentionally in a way that compromises the security goals and they also protect against the unintentional release of information to the same ends or even by outside entities with harmful intent.

The notion of *context* is fundamental for pervasive software. It includes any kind of computationally accessible information coming both from outside (e.g., available devices, code libraries, data offered by the environment), and from inside the application boundaries (e.g., user profiles). Of course, the context assumes a different shape depending on the architectural model to which software adheres. E.g., in the Cloud Computing paradigm, the context contains at least the description of the computational resources offered by the cloud provider, and also a measure of the available portion of each resource, and the way these are partitioned for multi-tenancy.

Traditional software engineering methodologies adopt a *static* model of software development, where the boundaries between specification and development are rigidly fixed; the interactions with the operational environment are assumed *a priori*; and software reconfiguration usually occurs *off-line*. This approach becomes inadequate, since applications now run in a partially known, ever changing operational environment. It is thus crucial studying *evolutionary* models for addressing the challenges posed by pervasive software.

Some illustrative, yet largely incomplete, cases of computational models and technologies towards the effective implementation and usage of pervasive software have been already developed and deployed. Among these, the most significant are Service Oriented Computing, the Internet of Things and Cloud Computing. Each of them is fostered by and addresses different aspects of the implementation and the usage of ubiquitous computing as follows.

In the Service Oriented Computing approach, applications are open-ended, heterogeneous and distributed. They are built by composing software units called services, which are published, linked, and invoked on-demand by other services using standard internet-based protocols. Moreover, applications can dynamically reconfigure themselves, by re-placing the services in use with others. Finally, services are executed on heterogeneous systems and no assumptions can be taken on their running platforms. In brief, a service offers its users access remote resources, i.e. data and programs.

A further step towards ubiquitous computing is when software pervades the objects of our everyday life, e.g. webTV, cars, smart phones, eBook readers, to make a few examples. These heterogeneous entities often have a limited computational power, but are capable of connecting to the internet, coordinating and interacting each other, in the so-called *plug&play* fashion. The real objects, as well as others of virtual nature (programs, services, etc.), which are connected in this way, form the Internet of Things. Objects become points where information can be collected and where some actions can be performed to process it, so changing the surrounding environment.

Cloud computing features facilities that are present on both the approaches above. Indeed, it offers through the network a hardware and software infrastructure on which end-users can run their programs on-demand. In addition, a rich variety of dynamic resources, such as networks, servers, storage, applications and services are made available. A key point is that these resources are ``virtualized'' so that they appear to their users as fully

dedicated to them, and potentially unlimited.

Many different techniques and approaches are being proposed to tackle the security issues typical of the pervasive computing scenario. There is a very rich literature about pervasive computing, from different points of view, including social, political, forensics, technological and scientific ones. By only considering the approaches within the last two viewpoints, large communities grew in a mesh of mostly overlapping fields, each one with its own techniques. In this paper we will lightly review these techniques focusing on adaptivity and security. Adaptivity is the capability of digital entities to fit for a specific use or situation; in a pervasive computing scenario this is a key aspect. Security is mandatory because the apparent simplicity of use of the new technologies hides their not trivial design and implementation, that become evident only when something goes wrong. In general, the risk is exchanging simplicity for absence of attention.

2. STATE OF THE ART AND CHALLENGES

The development of pervasive adaptive software has been investigated from different perspectives (control theory, artificial intelligence, programming languages) and several proposals have been put forward; for a survey, see [85,61]. Moreover, the opportunities of exploiting the offered techniques have transformed the ways software architectures are designed and implemented. Also user experience, (namely how a person feels when interfacing with a pervasive application) has been extremely transformed. Below, we focus on three branches, and related technologies, that we consider pivotal in pervasive computing from both the developer and the user perspective. We think that security and context-awareness are among the main concerns of pervasive computing, and so we mainly report on the results on these aspects focusing on the software engineering standpoint.

Service Oriented Computing

Service Oriented Computing (SOC) is a well-established paradigm to design distributed applications [80, 79, 78, 47]. In this paradigm, applications are built by assembling together independent computational units, called *services*. Services are stand-alone components distributed over a network, and made available through standard interaction mechanisms.

The main research challenges in SOC are described in [79]. An important aspect is that services are *open*, in that they are built with little or no knowledge about their operating environment, their clients, and further services therein invoked.

Adaptivity shows up in the SOC paradigm at various levels. At the lower one, the middleware should support dynamically reconfigurable run-time architectures and dynamic connectivity.

Service composition heavily depends on several, possibly, conflicting features:

- which information about a service is made public;
- how those services are selected that match the user's requirements;
- the actual run-time behaviour of the chosen services.

Service composition demands then autonomic mechanisms also driven by business requirements. The service oriented applications, made up by

loosely coupled services, also require self management features to minimise human intervention: self-configuring, self-adapting, self-healing, self-optimising, in the spirit of autonomic computation [63].

A crucial issue concerns defining and enforcing non-functional requirements of services, e.g. security and service level agreement ones. In particular, service assembly makes security imposition even harder. One reason why is that services may be offered by different providers, which only partially trust each other. On the one hand, providers have to guarantee the delivered service to respect a given security policy, in any interaction with the open operational environment, and regardless of who actually called the service. On the other hand, clients may want to protect their sensible data from the services invoked. Furthermore, security may be breached even when all the services are trusted, because of unintentional behaviour due, e.g. to design or implementation bugs, or because the composition of the services exhibits some unexpected and unwanted behaviour, e.g. leakage of information.

Web Services [8, 87, 92] built upon XML technologies are possibly the most illustrative and well-developed example of the SOC paradigm. Indeed, a variety of XML-based technologies already exist for describing, discovering and invoking web services [42, 26, 12, 2]. There are several standards for defining and enforcing non-functional requirements of services, e.g. WS-Security [14], WS-Trust [11] and WS-Policy [27]. The kind of security taken into account in these standards only concerns end-to-end requirements about secrecy and integrity of the messages exchanged by the parties.

Assembly of services can occur in two different flavours: *orchestration* or *choreography*. Orchestration describes the interactions from the point of view of a single service, while choreography has a global view, instead. Languages for orchestration and choreography have been proposed, e.g. BPEL4WS [12, 68] and WS-CDL [66]. However these languages have no facilities to explicitly handle security of compositions, only being focused on end-to-end security. Instead, XACML [3] gives a more structured and general approach, because it allows for declaring access control rules among objects that can be referenced in XML.

It turns out that the languages mentioned above do not describe many non-functional requirements, especially the ones concerning the emerging behaviour obtained by assembling services.

The literature on formal methods reports on many (abstract) languages for modelling services and their orchestration, see [52, 25, 56, 69, 19, 74, 70, 94, 36, 32] just to cite a few; [21] is a recent detailed survey on approaches to security and related tools, especially within the process calculi framework, [73] provides a formal treatment for a subset of XACML. An approach to the secure composition of services is presented in [18, 19]. Services may dynamically impose policies on resource usage, and they are composed guaranteeing that these policies will actually be respected at run-time. The security control is done efficiently at static time, by exploiting a type and effect system and model-checking. The problem of relating orchestration and choreography has been addressed in [38, 43], but without focusing on security issues.

Recently, increasing attention has been devoted to express service contracts as behavioural or session types [60]. These types synthesize the essential aspects of the interaction behaviour of services, while allowing for efficient

static verification of properties of composed systems. Through session types, [62] formalises compatibility of components and [31] describes adaptation of web services. Security has also been studied using session types, e.g. by [24, 17, 16].

Internet of Things

In 1988, Mark Weiser described his vision about the coming age of ubiquitous computing:

> Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives

In this world, sensors and computers are commodities available everywhere and surrounding people anytime. This idea has given rise to what is now called the *Internet of Things* [15] or *Everyware* [54]. Due to the pervasive integration of connectivity and identification tags, real objects are represented by digital entities in the virtual environment supported by dynamic opportunistic networks [81] or by the Internet. This is the case, e.g., of a fridge that becomes an active entity on the Internet ready to be queried for its contents. Such an intelligent space is then made of smart things, physical and endowed with software pieces, or fully virtual, that is highly interconnected and mutually influences their behaviour.

The software of intelligent spaces has then to be aware of the surrounding environment and of the ongoing events, to reflect the idea of an active space reacting and adapting to a variety of different issues, e.g. arising from people, time, programs, sensors and other smart things. Besides being assigned a task, a device can also take on the responsibility of proactively performing some activities with or for other digital entities.

The *Ambient calculus* [39] is among the first proposals to formalise those aspects of intelligent, virtual environments that mainly pertain to the movement of (physical devices and) software processes. Processes are hosted in virtual, separated portions of the space, called *ambients*. Processes can enter and leave ambients, and be executed in them. This calculus has been used, e.g. [34] to specify a network of devices, in particular to statically guarantee some properties of them, e.g. for regulating rights to the provision and discovery of environmental information.

Security plays a key role in the Internet of Things, not only because any digital entity can plug in, but also because the smart things may be devices, with also specific physical dimensions. The key point here is that information and physical security became interdependent, and traditional techniques that only focus on digital security are inadequate [37]. For example, there are some papers facing these issues with suitable extensions of the Ambient calculus, among which [71, 35, 29, 28, 91], but these proposals do not address the protection of the physical layer of smart things.

In addition, since spaces are active, the digital entities composing them may easily collect sensible information about the nearby people and things. Privacy may therefore be violated, because people are not always aware that their sensible data may be collected, nor that their activities are always context-aware, either. Even worse: it is possible through data mining to

disclose pieces of information, possibly confidential, so originating a tension with a tacit assumption or an explicit guarantee of privacy. For example, the analysis of behavioural patterns over big data can infer the actual identity of the individuals, violating their assumed anonymity [88].

Although some workshops and conferences are being organized on the new security topics of the Internet of Things, to the best of our knowledge little work is done in the area of formal methods, except for studies on protocols that guarantee anonymity (for brevity, we only refer the reader to the discussion on related work and to the references of [96]).

Cloud computing

The US National Institute of Standards and Technology defines Cloud computing as follows:

> Cloud computing is a model for enabling convenient, on-demand network access a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction

Cloud computing refers therefore to both the applications therein deployed and made available on the Internet and to the hardware infrastructures that makes this possible. The key characteristics of Cloud computing include on-demand self-service, ubiquitous network access, resource pooling.

Also, Cloud systems are characterized by some peculiar adaptivity characteristics called *elasticity* and *measured services* [10]. These are related to the two different viewpoints of the Cloud that customers and providers have. Elasticity refers to the ability of the provider to adapt its hardware infrastructure with minimal effort to the requirements of different customers, by scaling up and down the resources assigned to them. Measured services indicate load and resource usage optimization. It refers then to scalability from the provider point of view, usually achieved by multi-tenancy, i.e. by the capability of dynamically and accurately partitioning an infrastructure shared among various consumers.

The Cloud offers different models of services: *Software as a Service*, *Platform as a Service* and *Infrastructure as a Service*, so subsuming SOC. These three kinds can be organised in a stack hierarchy where the higher ones are built on the lower ones. At the lowest level, the Cloud provider offers an infrastructure made up by virtual machines, storages, and network interconnections. The whole application environment is granted to the user, who takes the responsibility for it. When supplying a platform as a service, the provider gives a basic software stack, usually including the operating system and a programming environment. At the highest level, we have Software as a Service, i.e. the provider enables its users to run on-demand one of its software service.

These aspects have been recently tackled within the formal methods approach. A formal calculus for defining the architecture of virtualized systems has been proposed in [20]. Elasticity has been studied and formalized in [30] through a calculus of processes with a notion of groups. A core functional language extended with explicit primitives for computational resource usage has been proposed in [22]. A process calculus with explicit primitives to control the distributed acquisition of resources

has been presented in [23].

The most relevant security issues in Cloud computing deal with access to resources, their availability and confidentiality [89]. Due to the distributed nature of the computation carried on in the Cloud, one challenge is to ensure that only authorised entities obtain access to resources.

A suitable identity management infrastructure should be developed, and users and services are required to authenticate with their credentials. However such a feature may affect the level of interoperability that a Cloud needs to have, because of the management of the identity tokens and of the negotiation protocols. In addition, providers should guarantee the right level of isolation among partitioned resources within the same infrastructure, because multi-tenancy can make unstable the borders between the resources dedicated to each user. Virtualization usually helps in controlling this phenomenon.

Moreover users should protect themselves from a possibly malicious Cloud provider, preventing him from stealing sensible data. This is usually achieved by exploiting encryption mechanisms [10] and identity management.

It is worth noting that whenever a program running in the Cloud handles some encrypted data, and the attacker is the provider itself, encryption is useless if also the encoding key is on the Cloud. For a limited number of cases, this problem can be circumvented by using suitable encryption schemata [53, 6]. There are providers that support working groups (e.g. [1]) who are aiming at making efficient homomorphism encryption so to commercially exploit it in the Cloud [75].

Note in passing that the Cloud can be misused and support attacks to security. Indeed, the great amount of computational resources made easily available can be exploited for large-scale hacking or denial of service attacks, and also to perform brute force cracking of passwords [4].

In the currently available systems, the responsibility for dealing with security issues is often shared between customer and providers. The actual balance depends on the service level at which security has to be enforced [9].

3. ADAPTIVITY AND SECURITY

In the previous sections, we overviewed a few technological and foundational features of ubiquitous computing, focusing on the Service Oriented Computing, the Internet of Things and the Cloud paradigms, from the developers' perspective. An emergent challenge is integrating adaptivity and security issues to support programming of applications and systems in the ubiquitous setting.

Adaptivity refers to the capability of a digital entity, in particular of its software components, to dynamically modify its behaviour reacting to changes of the surrounding active space, such as the location of its use, the collection of nearby digital entities, and the hosting infrastructure [86, 33]. Software must therefore be aware of its running environment, represented by a declarative and programmable *context*.

The notion of context assumes different forms in the three computational models discussed earlier. The shape of contexts in Service Oriented

Computing is determined by the various directories where services are published and by the end-points where services are actually deployed and made available, as well as by other information about levels of service, etc.

In the Internet of Things, the context is a (partial) representation of the active space hosting and made of the digital entities. Hence each digital entity may have its own context.

In the Cloud, a context contains the description of the computational resources offered by the centralised provider, and also a measure of the available portion of each resource; note that the context has to show to the provider the way computational resources are partitioned, for multi-tenancy.

A very short survey of the approaches to context-awareness follows, essentially from the language-based viewpoint (see, e.g. SCEL [48]). Other approaches range on a large spectrum, from the more formal description logics used to representing and querying the context [41, 93, 55] to more concrete ones, e.g. exploiting a middleware for developing context-aware programs [83].

Another approach is Context Oriented Programming (COP), introduced by Costanza [46]. Also subsequent work [58, 5, 65, 13] follows this paradigm to address the design and the implementation of concrete programming languages. The notion of *behavioural variation* is central to this paradigm. It is a chunk of behaviour that can be activated depending on the current working environment, i.e. of the context, so to dynamically modify the execution. Here, the context is a stack of *layers*, and a programmer can activate/deactivate layers to represent changes in the environment. This mechanism is the engine of context evolution.

Usually, behavioural variations are bound to layers: activating/deactivating a layer corresponds to activating/deactivating a behavioural variation. Only a few papers in the literature give a precise semantic description of the languages within the Context Oriented Programming paradigm. Among these, we refer the reader to [45, 64, 59, 44, 49] that however does not focus on security issues.

Security issues, instead, have been discussed in [40], even though this survey mainly considers specific context-aware applications, but not from a general formal methods viewpoint. Combining security and context-awareness requires addressing two distinct and interrelated aspects. On the one side, security requirements may reduce the adaptivity of software. On the other side, new highly dynamic security mechanisms are needed to scale up to adaptive software. Such a duality has already been put forward in the literature [95, 37] and we outline below two possible ways of addressing it: *securing context-aware systems* and *context-aware security*.

Securing context-aware systems aims at rephrasing the standard notions of confidentiality, integrity and availability [82] and at developing techniques for guaranteeing them [95]. Contexts may contain sensible data of the working environment (e.g. information about surrounding digital entities), and therefore to grant confidentiality this contextual information should be protected from unauthorised access. Moreover, the integrity of contextual information requires mechanisms for preventing its corruption by any entity in the environment.

A trust model is needed, taking also care of the roles of entities that can vary from a context to another. Such a trust model is important also because

contextual information can be inferred from the environmental one, provided by or extracted from digital entities therein, that may forge deceptive data. Since information is distributed, denial-of-service can be even more effective because it can prevent a whole group of digital entities to access relevant contextual information.

Context-aware security is dually concerned with the definition and enforcement of high-level policies that talk about, are based on, and depend on the notion of dynamic context. The policies most studied in the literature control the accesses to resources and smart things; see among the others [95, 62, 97]. Some e-health applications show the relevance of access control policies based on the roles attached to individuals in contexts [7, 51].

Most of the work on securing context-aware systems and on context-aware security aims at implementing various features at different levels of the infrastructures, e.g. in the middleware [83], or in the interaction protocols [57]. Indeed, the basic mechanisms behind security in adaptive systems have been studied much less. Moreover, the two dual aspects of context-aware security sketched above are often tackled separately. We lack then a unifying concept of security.

4. A LANGUAGE-BASED APPROACH TO SECURITY

We propose here linguistic mechanisms for adaptivity, coupled with methodological issues. Indeed, we think that an effective development of pervasive software requires a strong synergy between the methodologies and the development tools, in particular programming languages. Crucial to our approach is the design and implementation of high-level constructs that enable programmers to directly program adaptation at a fine-grain level and to ensure consistency and security of the adaptation process.

Our proposal faces the challenges pointed out above, by formally endowing a programming language with linguistic primitives for context-awareness and security, provided with a clear formal semantics. We suitably extend and integrate together techniques from COP, type theory and model-checking.

In particular, we introduced in [50] a static technique ensuring that a program:

- i. Adequately reacts to context changes;
- ii. Accesses resources in accordance with security policies;
- iii. Exchanges messages, complying with specific communication protocols.

The kernel of our proposal was COML [49], an extension of ML with COP features. Our first concern is the context, an active and complex entity that evolves independently of the applications. A programmer specifies the contents and the changes of a context, using its own specific mechanisms and rules, that are typically different from those used in programming applications. Indeed, programming the context requires skills different from those needed for applications. This methodological issue, as well as separation of concerns motivates us to define a two-component language: a declarative constituent for programming the context and a functional one for computing.

The declarative approach allows programmers to express *what* information the context has to include, leaving to the virtual machine *how* this information is actually collected and managed. For us, a context is a knowledge base and we implement it as a Datalog program [77, 72]. With this representation, adaptive programs can query the context by simply verifying whether a given property holds in it, i.e. by checking a Datalog goal. During the needed deductions the relevant information is also retrieved.

As for programming adaptation, we propose two mechanisms. The first one takes care of those program variables that assume different values depending on the different properties of the current context. To make that explicit, we introduce the notion of *context-dependent binding*, a sort of dynamic binding.

The second mechanism is based on behavioural variations, the fundamental concept of the COP paradigm. Usually, behavioural variations are not first class constructs in COP languages, rather, they are expressed as partial definition of procedures or classes or methods or modules. Instead, we equip COML with first class higher-order behavioural variations, that can therefore be referred to by identifiers, and passed as arguments to, and returned by functions. This provides us with a natural hook for programming dynamic adaptation patterns, as well as reusable and modular code. Note in passing that a behavioural variation can be supplied by the context, and then composed with existing ones, so implementing the autonomic element of [67].

We now discuss our programming model. It assumes that the virtual machine of the language provides its users with a collection of system variables, values, functions and predicates through a predefined API. Consequently, the context is split in two parts: the *system* and the *application* context.

The first one is provided by the virtual machine through its API; while the other one stores specific knowledge of the application, and the programmer initially fills in its contents. Obviously, programs acquire information about the system context through system predicates, but we stress that the actual values holding therein are only available at runtime.

In our execution model, the compiler produces a triple (C,P,H), where C is the application context, P is the program object code and H is an approximation of the program, used to verify properties about the program. Given such a triple, at loading-time the virtual machine performs a *linking* and a *verification* phase. The linking phase resolves system variables and links the application context to the system context, so obtaining the initial context that, of course, is checked for consistency.

Note that linking itself makes a first adaptive step, because it enables the application to use the capabilities of the hosting system, be they resources, data or code. In the spirit of Proof-Carrying code [76] and of the Java Bytecode Verifier [84], the verification phase exploits the approximation H to check that the program P will adapt to *all* the contexts that may occur at runtime.

If both phases succeed program evaluation begins, otherwise it is aborted.

5. CONCLUDING REMARKS

We reviewed some techniques for designing and developing adaptive applications within the pervasive computing paradigm. We focused on the issues arising from handling security policies.

REFERENCES

1. Cloud cryptography group at Microsoft Research, http://research.microsoft.com/en-us/projects/cryptocloud/

2. UDDI technical white paper. Tech. rep., W3C (2000)

3. eXtensible Access Control Markup Language (XACML) Version 2.0}. Tech. rep., OASIS (2005)

4. The future of cloud computing. Tech. rep., European Commision, Information Society and Media (2010)

5. Achermann, F., Lumpe, M., Schneider, J., Nierstrasz, O.: PICCOLA --- a small composition language. In: Formal methods for distributed processing. pp. 403--426. Cambridge University Press (2001)

6. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: Proceedings of the 2004 ACM SIGMOD international conference on Management of data. pp. 563--574. SIGMOD '04, ACM, New York, NY, USA (2004).

7. Al-Neyadi, F., Abawajy, J.: Context-based e-health system access control mechanism. Advances in information security and its application pp. 68-77 (2009)

8. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures and Applications. Springer-Verlag (2004)

9. Amazon.com Inc.: Aws customer agreement., http://aws.amazon.com /agreement

10. Amazon.com Inc.: Overview of Amazon Web Services. http://aws.amazon.com/whitepapers (2010)

11. Anderson, S., et al.: Web Services Trust Language (WS-Trust) (2005)

12. Andrews, T., etal.: Business Process Execution Language for Web Services (BPEL4WS), Version 1.1 (2003)

13. Appeltauer, M., Hirschfeld, R., Haupt, M., Masuhara, H.: ContextJ: Context-oriented programming with java. Computer Software 28(1) (2011)

14. Atkinson, B., et al.: Web Services Security (WS-Security) (2002)

15. Atzori, L., Iera, A., Morabito, G.: The internet of things: A survey. Computer Networks 54(15), 2787--2805 (2010)

16. Barbanera, F., Bugliesi, M., Dezani-Ciancaglini, M., Sassone, V.: Space-aware ambients and processes. Theor. Comput. Sci. 373(1-2), 41--69 (2007)

17. Barbanera, F., Dezani-Ciancaglini, M., Salvo, I., Sassone, V.: A type inference algorithm for secure ambients. Electr. Notes Theor. Comput. Sci.

62, 83--101 (2001)

18. Bartoletti, M., Degano, P., Ferrari, G.L., Zunino, R.: Semantics-based design for secure web services. IEEE Trans. Software Eng. 34(1), 33--49 (2008)

19. Bartoletti, M., Degano, P., Ferrari, G.L., Zunino, R.: Local policies for resource usage analysis. ACM Trans. Program. Lang. Syst. 31(6) (2009)

20. Bhargavan, K., Gordon, A.D., Narasamdya, I.: Service combinators for farming virtual machines. In: Lea, D., Zavattaro, G. (eds.) COORDINATION. Lecture Notes in Computer Science, vol. 5052, pp. 33--49. Springer (2008)

21. Blanchet, B.: Security protocol verification: Symbolic and computational models. In: Principles of Security and Trust - First International Conference, POST 2012, Lecture Notes in Computer Science, 7215, pp. 3--29 (2012)

22. Bodei, C., Dinh, V.D., Ferrari, G.L.: Safer in the clouds (extended abstract). In: Bliudze, S., Bruni, R., Grohmann, D., Silva, A. (eds.) ICE. EPTCS, vol. 38, pp. 45--49 (2010)

23. Bodei, C., Dinh, V.D., Ferrari, G.L.: Predicting global usages of resources endowed with local policies. In: Mousavi, M.R., Ravara, A. (eds.) FOCLASA. EPTCS, vol. 58, pp. 49--64 (2011)

24. Bonelli, E., Compagnoni, A., Gunter, E.: Typechecking safe process synchronization. In: Proc. Foundations of Global Ubiquitous Computing. ENTCS, vol. 138(1) (2005)

25. Boreale, M., et al.: SCC: a service centered calculus. In: WS-FM. Springer Lecture Notes in Computer Science, vol. 4184 (2006)

26. Box, D., et al.: Simple Object Access Protocol (SOAP) 1.1. WRC Note (2000)

27. Box, D., et al.: Web Services Policy Framework (WS-Policy) (2002)

28. Braghin, C., Cortesi, A.: Flow-sensitive leakage analysis in mobile ambients. Electr. Notes Theor. Comput. Sci. 128(5), 17--25 (2005)

29. Braghin, C., Cortesi, A., Focardi, R.: Security boundaries in mobile ambients. Computer Languages, Systems & Structures 28(1), 101 -- 127 (2002),

30. Bravetti, M., Giusto, C.D., Perez, J.A., Zavattaro, G.: Adaptable processes (extended abstract). In: Bruni, R., Dingel, J. (eds.) FMOODS/FORTE. Lecture Notes in Computer Science, vol. 6722, pp. 90--105. Springer (2011)

31. Brogi, A., Canal, C., Pimentel, E.: Behavioural types and component adaptation. In: Proc. Algebraic Methodology and Software Technology (AMAST). Springer Lecture Notes in Computer Science, vol. 3116 (2004)

32 Bruni, R.: Calculi for service-oriented computing. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM. Lecture Notes in Computer Science, vol. 5569, pp. 1--41. Springer (2009)

33. Bruni, R., Corradini, A., Gadducci, F., Lluch-Lafuente, A., Vandin, A.: A conceptual framework for adaptation. In: de Lara, J., Zisman, A. (eds.) FASE. Lecture Notes in Computer Science, vol. 7212, pp. 240--254. Springer (2012)

34. Bucur, D., Nielsen, M.: Secure data flow in a calculus for context awareness. In: Degano, P., De~Nicola, R., Meseguer, J. (eds.) Concurrency, Graphs and Models, Lecture Notes in Computer Science, vol. 5065, pp. 439--456. Springer (2008)

35. Bugliesi, M., Castagna, G., Crafa, S.: Reasoning about security in mobile ambients. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR. Lecture Notes in Computer Science, vol. 2154, pp. 102--120. Springer (2001)

36. Caires, L., De Nicola, R., Pugliese, R., Vasconcelos, V.T., Zavattaro, G.: Core calculi for service-oriented computing. In: Results of the SENSORIA Project, Lecture Notes in Computer Science, 6852, pp. 153--188 (2011)

37. Campbell, R., Al-Muhtadi, J., Naldurg, P., Sampemane, G., Mickunas, M.D.: Towards security and privacy for pervasive computing. In: Proceedings of the 2002 Mext-NSF-JSPS international conference on Software security: theories and systems. pp. 1--15. ISSS'02, Springer-Verlag, Berlin, Heidelberg (2003),

38. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centred programming for web services. In: European Symposium in Programming Languages (ESOP). vol. 4421 (2007)

39. Cardelli, L., Gordon, A.: Mobile ambients. In: Nivat, M. (ed.) Foundations of Software Science and Computation Structures, Lecture Notes in Computer Science, vol. 1378, pp. 140--155. Springer Berlin (1998)

40. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Tech. rep., Dartmouth College, Hanover, NH, USA (2000)

41. Chen, H., Finin, T., Joshi, A.: An ontology for context-aware pervasive computing environments. The Knowledge Engineering Review 18(03), 197-207 (2003)

42. Chinnici, R., Gudgina, M., Moreau, J., Weerawarana, S.: Web Service Description Language (WSDL), Version 1.2 (2002)

43. Ciancia, V., Ferrari, G.L., Guanciale, R., Strollo, D.: Event based choreography. Sci. Comput. Program. 75(10), 848--878 (2010)

44 Clarke, D., Costanza, P., Tanter, E.: How should context-escaping closures proceed? In: International Workshop on Context-Oriented Programming. pp. 1:1--1:6. COP '09, ACM, New York, NY, USA (2009) }

45. Clarke, D., Sergey, I.: A semantics for context-oriented programming with layers. In: International Workshop on Context-Oriented Programming. pp. 10:1--10:6. COP '09, ACM, New York, NY, USA (2009)

46. Costana, P.: Language constructs for context-oriented programming. In: In Proceedings of the Dynamic Languages Symposium. pp. 1--10. ACM Press (2005)

47. Curbera, F., Khalaf, R., Mukhi, N., Tai, S., Weerawarane, S.: The next step in web services. Communications of the ACM, 46(10) (2003)

48. De Nicola, R., Ferrari, G., Loreti, M., Pugliese, R.: A language-based approach to autonomic computing. In: Formal Methods for Component and Objects 2011. Lecture Notes in Computer Science, 7542, Springer (2013)

49. Degano, P., Ferrari, G.L., Galletta, L., Mezzetti, G.: Typing context-

dependent behavioural variations. In: PLACES 2012. vol. to appear in EPTCS (2012)

50. Degano, P., Ferrari, G.L., Galletta, L., Mezzetti, G.: Typing for coordinating secure behavioural variations. In: Coordination Models and Languages. Lecture Notes in Computer Science, vol. 7274. Springer (2012)

51. Deng, M., Cock, D.D., Preneel, B.: Towards a cross-context identity management framework in e-health. Online Information Review 33(3), 422--442 (2009)

52. Ferrari, G., Guanciale, R., Strollo, D.: {JSCL}: A middleware for service coordination. In: Proc. FORTE, Springer LNCS, vol. 4229 (2006)

53. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st annual ACM symposium on Theory of computing. pp. 169--178. ACM (2009)

54. Greenfield, A.: Everyware: The dawning age of ubiquitous computing. Peachpit Press (2006)

55. Gu, T., Wang, X., Pung, H., Zhang, D.: An ontology-based context model in intelligent environments. In: Proceedings of communication networks and distributed systems modeling and simulation conference. vol. 2004, pp. 270--275 (2004)

56 Guidi, C., Lucchi, R., Gorrieri, R., Busi, N., Zavattaro, G.: SOCK: A calculus for service oriented computing. In: Proc. Service-Oriented Computing (ICSOC). Springer LNCS, vol. 4294 (2006)

57. Heer, T., Garcia-Morchon, O., Hummen, R., Keoh, S., Kumar, S., Wehrle, K.: Security challenges in the ip-based internet of things. Wireless Personal Communications pp. 1--16 (2011)

58. Hirschfeld, R., Costanza, P., Nierstrasz, O.: Context-oriented programming. Journal of Object Technology, March-April 2008, ETH Zurich 7(3), 125-151 (2008)

59. Hirschfeld, R., Igarashi, A., Masuhara, H.: ContextFJ: a minimal core calculus for context-oriented programming. In: Proceedings of the 10th international workshop on Foundations of aspect-oriented languages. pp. 19--23. FOAL '11, ACM, New York, NY, USA (2011)

60. Honda, K., Vasconcelos, V., Kubo, M.: Language primitives and type discipline for structured communication-based programming. Programming Languages and Systems pp. 122--138 (1998)

61. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing degrees, models, and applications. ACM Comput. Surv. 40(3), 7:1--7:28 (2008)

62. Hulsebosch, R., Salden, A., Bargh, M., Ebben, P., Reitsma, J.: Context sensitive access control. In: Proceedings of the tenth ACM symposium on Access control models and technologies. pp. 111--119. ACM (2005)

63. IBM: An architectural blueprint for autonomic computing. Tech. rep. (2005)

64. Igarashi, A., Pierce, B.C., Wadler, P.: Featherweight java: a minimal core

calculus for Java and GJ. ACM Trans. Program. Lang. Syst. 23(3), 396--450 (2001)

65. Kamina, T., Aotani, T., Masuhara, H.: Eventcj: a context-oriented programming language with declarative event-based context transition. In: Proceedings of the tenth international conference on Aspect-oriented software development. pp. 253--264. AOSD '11, ACM, New York, NY, USA (2011)

66. Kavantza, N., et al.: Web Service Coreography Description Language}, http://www.w3.org/TR/ws-cdl-10/

67. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer 36(1), 41--50 (Jan 2003)

68. Khalaf, R., Mukhi, N., Weerawarana, S.: Service oriented composition in BPEL4WS. In: Proc. WWW Conference (2003)

69. Lapadula, A., Pugliese, R., Tiezzi, F.: A calculus for orchestration of web services. In: European Symposium in Programming Languages (ESOP), LNCS vol. 4421, (2007)

70. Lazovik, A., Aiello, M., Gennari, R.: Encoding requests to web service compositions as constraints. In: Proc. Principles and Practice of Constraint Programming, Springer LNCS, vol. 3709 (2005)

71. Levi, F., Sangiorgi, D.: Mobile safe ambients. ACM Trans. Program. Lang. Syst. 25(1), 1--69 (2003)

72. Loke, S.W.: Representing and reasoning with situations for context-aware pervasive computing: a logic programming perspective. Knowl. Eng. Rev. 19(3), 213--233 (2004)

73. Masi, M., Pugliese, R., Tiezzi, F.: Formalisation and implementation of the xacml access control mechanism. In: Barthe, G., Livshits, B., Scandariato, R. (eds.) ESSoS. Lecture Notes in Computer Science, vol. 7159, pp. 60--74. Springer (2012)

74. Misra, J.: A programming model for the orchestration of web services. In: 2nd International Conference on Software Engineering and Formal Methods, (2004)

75. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM workshop on Cloud computing security workshop. pp. 113--124. ACM (2011)

76. mNecula, G.C., Lee, P.: Safe, untrusted agents using proof-carrying code. In: Mobile Agents and Security. pp. 61--91. Springer (1998)

77. Orsi, G., Tanca, L.: Context modelling and context-aware querying. In: Datalog Reloaded, LNCS, vol. 6702. Springer (2011)

78. Papazoglou, M.P.: Service-oriented computing: Concepts, characteristics and directions. In: WISE (2003)

79. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: a research roadmap. Int. J. Cooperative Inf. Syst. 17(2), 223--255 (2008)

80. Papazouglou, M., Georgakopoulos, D.: Special issue on service oriented

computing. Communications of the {ACM} 46(10) (2003)

81. Pelusi, L., Passarella, A., Conti, M.: Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. Communications Magazine, IEEE 44(11), 134 --141 (2006)

82. Pfleeger, C., Pfleeger, S.: Security in computing. Prentice Hall (2003)

83. Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R., Nahrstedt, K.: Gaia: a middleware platform for active spaces. ACM SIGMOBILE Mobile Computing and Communications Review 6(4), 65--67 (2002)

84. Rose, E.: Lightweight bytecode verification. J. Autom. Reason. 31(3-4) (2004)

85 Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. ACM Trans. Auton. Adapt. Syst. 4(2), 14:1--14:42 (May 2009)

86. Schilit, B., Adams, N., Want, R.: Context-aware computing applications. In: In Proceedings of the Workshop on Mobile Computing Systems and Applications. pp. 85--90. IEEE Computer Society (1994)

87. Stal, M.: Web services: Beyond component-based computing. Communications of the ACM 55(10) (2002)

88. Sweeney, L., et al.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty Fuzziness and Knowledge Based Systems 10(5), 557--570 (2002)

89. Takabi, H., Joshi, J., Ahn, G.: Security and privacy challenges in cloud computing environments. Security & Privacy, IEEE 8(6), 24--31 (2010)

90. Vallecillo, A., Vansconcelos, V., Ravara, A.: Typing the behaviours of objects and components using session types. In: Proc. of FOCLASA (2002)

91. Vitek, J., Castagna, G.: Seal: A framework for secure mobile computations. In: Bal, H.E., Belkhouche, B., Cardelli, L. (eds.) ICCL Workshop: Internet Programming Languages. Lecture Notes in Computer Science, vol. 1686, pp. 47--77. Springer (1998)

92. Vogels, W.: Web services are not distributed objects. IEEE Internet Computing 7(6) (2003)

93. Wang, X., Zhang, D., Gu, T., Pung, H.: Ontology based context modeling and reasoning using owl. In: Pervasive Computing and Communications Workshops, 2004. IEEE (2004)

94. Wirsing, M.. (ed.): Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing, Lecture Notes in Computer Science, vol. 6582. Springer (2011)

95. Wrona, K., Gomez, L.: Context-aware security and secure context-awareness in ubiquitous computing environments. In: XXI Autumn Meeting of Polish Information Processing Society (2005)

96. Yang, M., Sassone, V., Hamadou, S.: A game-theoretic analysis of cooperation in anonymity networks. In: Degano, P., Guttman, J.D. (eds.)

POST. Lecture Notes in Computer Science, vol. 7215, pp. 269--289. Springer (2012)

97. Zhang, G., Parashar, M.: Dynamic context-aware access control for grid applications. In: Grid Computing, 2003. Proceedings. Fourth International Workshop on. pp. 101--108. IEEE (2003)

---

[1] Dipartimento di Informatica, Universita' di Pisa

[2] Dipartimento di Informatica, Universita' di Pisa

[3] Dipartimento di Informatica, Universita' di Pisa

[4] Dipartimento di Informatica, Universita' di Pisa

[5] Dipartimento di Informatica, Universita' di Pisa

[6] This work has been partially supported by IST-FP7-FET open-IP project ASCENS, MIUR Project Security Horizons