ReIOS: Reflective Architecting in the Internet of Objects

Marina Mongiello, Gennaro Boggia and Eugenio Di Sciascio
Dept. of Electrical and Information Engineering, Politecnico di Bari
Via Orabona, 4, 70125, Bari, Italy

Keywords: Reflection, Middleware, IoT, Architectural Modeling.

Abstract: Self-adaptive systems are modern applications in which the running software system should be able to react on its own, by dynamically adapting its behavior, for sustaining a required set of qualities of service, and dynamic changes in the context or in user requirements. They are typically involved in Future Internet development such as the Internet of Things where interoperability, flexibility, and adaptability are key requirements. Convergence of contents, services, things and networks seems to be the cornerstone to fullfil these requirements. We propose a reflective approach to provide a common abstraction for automating the deployment of component based applications in the Internet of Things environment. The proposed framework allows the design of heterogeneous, distributed, and adaptive applications built on the component based software engineering paradigm. The framework considers a metamodel instantiated in a Rest middleware properly modified for allowing different implementations by using reflective design patterns. We are currently working to refine the framework metamodel and to validate it in several implementation domains.

1 INTRODUCTION

Model Driven engineering is a solid approach for the development of complex systems and applications since it enables modeling abstraction at different levels. But in modern applications the adaptation to changes in context and requirements implies an extension of the modeling use from design time to run time(Lehmann et al., 2011). In fact, changes in requirements, assumptions about the environment, and continuous changes in usage profiles are difficult to predict and to anticipate at design time. Thus, the *models@runtime* is the emerging paradigm aiming to manage the transfer of traditional modeling activities (such as verification, design and quality evaluation) to the running system. Several approaches and proposals in the field of self adaptive applications, with the purpose of modeling the architecture and verifying the system properties, have been proposed in the recent past (Salehie and Tahvildari, 2009). A common denominator in many of these works is the objective to give systems an acceptable level of reliability and of flexibility necessary to ensure adaptability, without compromising dependability.

A winning strategy to fulfill the main conflicting quality requirements of self adaptive applications and, at the same time, to enable the modeling extension to run time situation is the formalization. The use of a metamodel is fundamental in run time models to identify and, thus, separate descriptive features – description of the model – from prescriptive features – the prescription of how the model and hence the system should be. In fact, to use a run time model it is necessary to identify elements which are descriptive and elements that are prescriptive (that can be identified through the metamodel).

Among the numerous domains and applications, herein, we consider, as a significant and very attracting example of application, the Internet of Things. This emerging paradigm refers to a combined part of Future Internet – communication and network issues – and of physical objects or devices or virtual objects devices, that is information heterogeneous in nature and seamless integrated into the communication network. In this context, considering that there are no standardized modeling approach, we propose a new solution, namely, ReIOS: a **Re**flective framework for **I**nternet of **O**bjects and **S**ensor network. It should be useful in designing heterogeneous, distributed, and adaptive applications built on the component based software engineering paradigm. The framework is made up of an abstract level in which we model relevant building blocks of a distributed environment in which resources, events, applications are managed using a reflective controller, ensuring at the same time

Mongiello, M., Boggia, G. and Sciascio, E.
ReIOS: Reflective Architecting in the Internet of Objects.
DOI: 10.5220/0005800603840389
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 384-389
ISBN: 978-989-758-168-7

flexibility and adaptability. The abstract level is instantiated in a concrete level where the reflective controller enables application execution using reflective design patterns. The remaining of the paper is organized as it follows. In Section 2 main concepts useful to define our domain and the proposed methodology are drawn. Section 3 defines the proposal. An instantiation of the proposed metamodel is described in Section 4. Final discussion and future development of the proposal are reported in Section 6.

2 SOME BACKGROUND

2.1 Reflective Patterns

Reflective systems are characterized by a multi-level architecture separating meta-levels from the base level. In reflective object-oriented systems, certain aspects of the base level are represented or reified as meta-objects to provide a representation of the execution state as data. The role of meta-objects is to observe and modify the base objects they represent (their referents). The code dealing with meta-objects is called a meta-program and its interfaces are called meta-object protocols. By combining design patterns and reflective languages it is possible to obtain reflective design patterns, useful in managing flexibility issues in adaptive system implementation and modeling. A reflective enhancement of design patterns to improve flexibility is proposed in (Ferreira and Rubira, 1998; Maes, 1987a; Maes, 1987b). Improvement is performed by adapting standard design pattern (Gamma et al., 1994) to a reflective implementation obtaining reflective design patterns by extending the programming language with reflective operators. The approach provides a flexible implementation toward design patterns-based software evolution in which loose coupling and high cohesion are guaranteed.

2.2 Internet of Things

An increasing number of everyday machines and objects are now embedded with sensors or actuators and have the ability to communicate each other over the Internet. Collectively they make up the new and emerging Internet of Things (IoT) paradigm. In this scenario, individual devices are connected through Machine-to-Machine (M2M) communication interfaces. Potential applications and services in the IoT include: smart devices, smart cities, smart grids, automotive, eHealth, home automation and energy management, remote industrial process control (Tan and Wang, 2010; Guinard et al., 2012). But, smart objects produce large volumes of data that need to be managed, processed, transferred, stored securely and, hence, standardized. The use of standards ensures interoperable and cost-effective solutions, opens up opportunities in new areas, and allows the market to reach its full potential (Guinard et al., 2010; Hamida et al., 2012).

2.3 REST Middleware

Nowadays, many vertical M2M solutions have been designed independently for different applications, making the current M2M market very fragmented, which inevitably hinders a large-scale M2M deployment. To decrease the market fragmentation there have been many efforts from different standardization bodies to define horizontal service layers. Among others, The European Telecommunications Standards Institute (ETSI) has defined with the SmartM2M standard a middleware which has a RESTful architecture (Vogli et al., 2015). On the other side, OneM2M, where are collaborating more than 200 standardization bodies and companies, is defining a RESTful middleware which will have a global validity (Swetina et al., 2014). Also other solutions based on the use of CoAP proposes RESTful based approach (Palattella et al., 2013).

All the proposed solutions have a common denominator: they provide RESTful middlewares separating the application from the communication domain. Middlewares are accessible via open interfaces and enable the development of services and applications independently of the underlying network. In addition, they provide several service capabilities to enable machine registration, synchronous and asynchronous communication, resource discovery, access rights management, group broadcast, and so on.

All the resources in the RESTful middlewares are organized in standardized resource trees and can be uniquely addressed by a Uniform Resource Identifier (URI). Their representations can be transferred and manipulated with verbs (i.e., retrieve, update, delete, and execute).

3 THE PROPOSED ReIOS FRAMEWORK

The aim of this paper is the proposal of a new framework, namely ReIOS: a Reflective framework for Internet of Objects and Sensor network. Its goal is to support modeling and design of heterogeneous, distributed and adaptive applications starting from the component based software engineering paradigm and using a run time modeling approach. For this reason, we use a reflective architecture as metamodel. More precisely the metamodel comprises the following abstractions:

- modeling the state event properties of the domain;
- modeling the discovery of services or applications;
- managing a repository of available apps;
- reflective management of applications to enforce the flexibility to ensure adaptability.

Each abstraction is a conceptual building block formalizing the framework: an Event Processing node, a Finite State Machine to model the states and events, a Discovery node, a Protocol Layer, a Reflective Controller, a Knowledge Base, as shown in Figure 1. More in details, the Finite State Machine describes the states and the events of the domain. The event processing node manages the events without the need of storing them persistently on the storage device; it changes the behavior of the devices according to the data stream generated by the devices themselves, depending on the reflective execution of apps managed in the reflective controller. A discovery node is used to search for given applications and services.

Standardized resources are stored in a Knowledge base and each resource is uniquely addressable via a Universal Resource Identifier (URI) and has a representation that can be transferred and manipulated (retrieved, updated, deleted and executed). The use of a metamodel and its structure facilitates the future development and extension of the model itself being compositional and scalable thanks to its abstraction with more complex and effective reasoning method for resource discovery.

The protocol manager handles RESTful requests. It receives information requests, independent on the protocol, and responds with a confirmation always independent on the adopted protocol; it allows also direct access to the underlying protocols so that developers could abstract from the details without losing track of them. It enable connection among multiple servers residing in the cloud, on PCs or single-board computers (Raspberry Pi, BeagleBone, and so on), by creating networks of distributed IoT devices. Moreover, it creates a manageable image of all the connected devices for communicating with microcontrollers, such as Arduino and Spark Core giving each device a REST API, both locally and in the cloud.

The Reflective controller is the main feature of the framework since it exploits the reflective approach. The proposed idea is to design the reflective controller using a reflective enhancement of design patterns in order to prevent modification and changes of middleware codes used in the IoT domain subsequent to changes in contexts, environment and user profile. Enhancement of design pattern flexibility is performed by adapting standard design pattern (Gamma et al., 1994) to a reflective implementation: Reflective design patterns are obtained by extending the programming language with reflective operators according to the approach defined in (Ferreira and Rubira, 1998; Maes, 1987a; Maes, 1987b) this provides a flexible approach toward design patterns-based software evolution in which loose coupling and high cohesion are guaranteed. The reflective controller enables deployment and execution of the apps responding to detached changes and modifications.

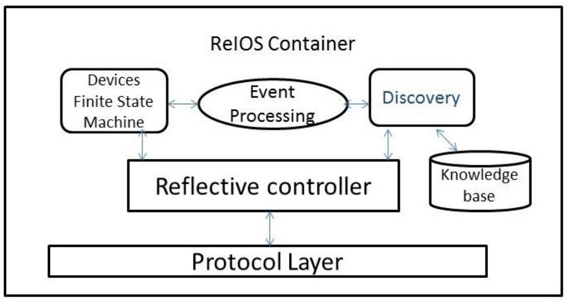

Figure 1: Graphical schema of the ReIOS framework.

4 ReIOS INSTANTIATION: A REFLECTIVE CONTROLLER IN MIDDLEWARE IMPLEMENTATION

The proposed reflection model, established with the help of proxies, provides a means of customizing application behavior and of adapting to new requirements without the need for modification. The reflection model supports customization of applications.

A meta-level architecture inherently exhibits a degree of separation of concerns, where the application core concerns (at the base level) are separated from the non-functional concerns (at the meta-level) in a natural way. To instantiate the metamodel, we build a reflective controller implementing the CRUD (Create, Retrieve, Update and Delete) methods for each resource. It carries out the checks required for operations, such as access permissions and verification of syntax resources. The functional architecture of the controller also manages the access of a resource from remote server to make the resource accessible to other machines.

To validate our proposal, we instantiated the metamodel in real implementation environment. We mapped our controller on a state of the art IoT middleware. We refer to well-known REST Middlewares (see Sec. II) and modified the standard class diagram by using Reflective pattern according to the methods proposed in in (Ferreira and Rubira, 1998; Maes, 1987a; Maes, 1987b). The main changes on the functional architecture and on the class design are performed on the controller of the Rest middleware on which we map the Resource controller of our framework. Here we briefly summarize these modifications:

- implementation of the Bridge pattern to decouple the implementations of classes AnnouncerController with the implementation of the Controller. In this way, the abstract class Announce Controller will inherit methods of the abstract controller class while maintaining a level of abstraction that allows to decouple implementations for various resources.
- Elimination of the use of two distinct classes to perform operations on a single resource or a collection of identical resources. In the code the two implementations are fundamentally different: some operations are valid for a single resource, the other for a collection, performing a preliminary operation to examine the passed parameter and decides which method to apply or to understand if it is a single resource or a collection.
- application of a proxy that stores temporarily calls to instantiate objects and then balance the trade-off flexibility / performance.

Figures 2 and 3 show the class design before and after reflective adaptation.

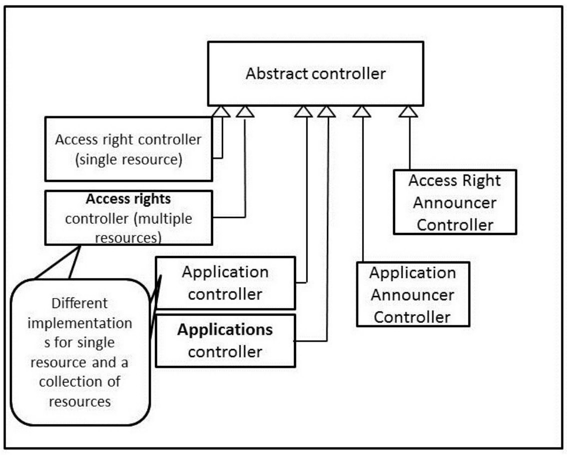
Abstract controller
Access right controller (single resource)
Access rights controller (multiple resources)
Access Right Announcer Controller
Application controller
Applications controller
Application Announcer Controller
Different implementations for single resource and a collection of resources

Figure 2: Class diagram of abstract controller.

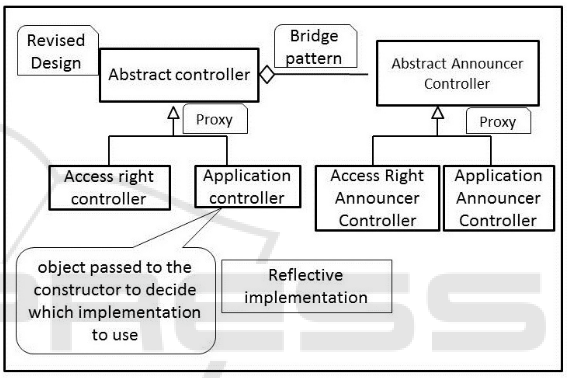
Revised Design
Abstract controller
Bridge pattern
Abstract Announcer Controller
Proxy
Proxy
Access right controller
Application controller
Access Right Announcer Controller
Application Announcer Controller
object passed to the constructor to decide which implementation to use
Reflective implementation

Figure 3: Revised Class diagram of abstract controller for reflective implementation.

5 RELATED WORK

Herein, we briefly recall state of the art in the field of adaptive modeling of software architecture considering two main categories of investigations: runtime modeling and/or verification for adaptive software systems and runtime composition and integration of applications in future internet and IoT domain. With respect to existing approach, we propose a formal framework to model adaptive architectures that can be used to instantiate architectural design of adaptive systems in various contexts at runtime.

5.1 Runtime Modeling and/or Verification for Adaptive Software Systems

Models for context-aware and self-adaptive systems have been widely studied in the last years. The work in (Baldauf et al., 2007) summarizes the common principles for the development of context-aware applications, in terms of architecture, middleware, and frameworks. Work (Strang and Linnhoff-Popien, 2004) provides a survey on methods for context modeling. Several surveys on self-adaptive systems have been proposed (see the work in (Cheng, 2009; R. de Lemos et al., 2013) and in (Weyns et al., 2012), just to name the most well-known). The work in (Salvaneschi et al., 2013) provides an exhaustive and structured state the art and compares three approaches to support the implementation of adaptive systems. A survey on architectural modeling in self-management may be found in (Kramer and Magee, 2007).

According to the cited works, formal approaches seem to be the most powerful instrument for modeling self-adaptive systems, while still preserving correctness and quality properties. Formal methods have been used to model MAPE-K loops in (Arcaini et al., 2015) and services composition in (Riccobene and Scandurra, 2015). In (Bucchiarone et al., 2015) graph transformation is used to model self-adaptation. Modeling of evolving systems based on components is instead adopted in (Pelliccione et al., 2008). Models for service choreography and composition is employed in (Autili et al., 2009) and by the same authors in (Autili et al., 2014), with specific reference to Future Internet applications. Composition of self-adaptive systems for dependability is proposed in (Cubo et al., 2014). The work in (Mongiello et al., 2015b) adopts semantic approach for run-time verification.

In (Calinescu, 2013), the authors overview emerging techniques for the engineering of high-integrity self-adaptive software; in the same paper, a service-based architecture aimed at integrating these techniques is introduced.

5.2 Runtime Composition and Integration of Applications in Future Internet and IoT Domain

Lightweight applications executable on gateways connecting heterogeneous devices are studied in (Vögler et al., 2015). Services are autonomous tasks that can be executed in run-time enviroments. A rigorous and lightweight theoretical foundation for representing the behavior of heterogeneous things is proposed in (Cubo et al., 2012). This work relies on a service-oriented paradigm. The approach proposed in (Torjusen et al., 2014) integrates run-time verification enablers in the feedback adaptation loop of the ASSET adaptive security framework. The scope of this integration is guaranteeing self-adaptive security and privacy properties in the eHealth settings. In (González et al., 2013), the authors present an approach dealing with the run-time verification of behavior-aware composition of things. They propose to check whether a mashup of things respects the specified behavior of the composed things. The approach is based on mediation techniques and complex event processing and is able to detect and inhibit invalid invocations. As a consequence, things only receive requests compatible with their behavior. Semantic approaches are in (Mongiello et al., 2015a) to model self-adaptive architectural model in IoT, the proposal uses a graph based model for data and processes to be exectued in IoT environment.

6 DISCUSSION AND FUTURE DEVELOPMENT

In this paper, we proposed a reflective framework for adaptive modeling of self-adaptive software in the internet of things domain. The framework is a metamodel to manage applications deployment and execution to enable software system to react on its own, by dynamically adapting its behavior, in response to changes in the environment, in the context or in user's experience. The proposed solution allows the design of heterogeneous, distributed and adaptive applications built on the component based software engineering paradigm. The metamodel has been instantiated in a Rest middleware example to modify its implementation by using reflective design patterns. We are currently working to refine the metamodel definition and its implementation in several real domain scenarios.

REFERENCES

Arcaini, P., Riccobene, E., and Scandurra, P. (2015). Modeling and analyzing MAPE-K feedback loops for self-adaptation. In *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18-19, 2015*, pages 13–23.

Autili, M., Benedetto, P. D., and Inverardi, P. (2009). Context-aware adaptive services: The PLASTIC approach. In *Fundamental Approaches to Software Engineering, 12th International Conference, FASE 2009, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, pages 124–139.

Autili, M., Inverardi, P., and Tivoli, M. (2014). CHOREOS: large scale choreographies for the future internet. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, Antwerp, Belgium, February 3-6, 2014*, pages 391–394.

Baldauf, M., Dustdar, S., and Rosenberg, F. (2007). A survey on context-aware systems. *Int. J. of Ad Hoc and Ubiquitous Computing*, 2(4):263–277.

Bucchiarone, A., Ehrig, H., Ermel, C., Pelliccione, P., and Runge, O. (2015). Rule-based modeling and static analysis of self-adaptive systems by graph transformation. In *Software, Services, and Systems*, pages 582–601.

Calinescu, R. (2013). Emerging techniques for the engineering of self-adaptive high-integrity software. In *Assurances for Self-Adaptive Systems*, pages 297–310. Springer.

Cheng, B. H. e. a. (2009). Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, pages 1–26. Springer-Verlag, Berlin, Heidelberg.

Cubo, J., Brogi, A., and Pimentel, E. (2012). Behaviour-aware compositions of things. In *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, pages 1–8. IEEE.

Cubo, J., Ortiz, G., Boubeta-Puig, J., Foster, H., and Lamersdorf, W. (2014). Adaptive services for the future internet. *J. UCS*, 20(8):1046–1048.

Ferreira, L. L. and Rubira, C. M. F. (1998). Reflective design patterns to implement fault tolerance. In *OOPSLA 1998*.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education.

González, L., Cubo, J., Brogi, A., Pimentel, E., and Ruggia, R. (2013). Run-time verification of behaviour-aware mashups in the internet of things. In *Advances in Service-Oriented and Cloud Computing*, pages 318–330. Springer.

Guinard, D., Ion, I., and Mayer, S. (2012). In search of an internet of things service architecture: Rest or ws-*? a developers perspective. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 326–337. Springer.

Guinard, D., Trifa, V., and Wilde, E. (2010). A resource oriented architecture for the web of things. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE.

Hamida, A. B., Kon, F., Oliva, G. A., Dos Santos, C. E. M., Lorré, J.-P., Autili, M., De Angelis, G., Zarras, A., Georgantas, N., Issarny, V., et al. (2012). An integrated development and runtime environment for the future internet. In *The Future Internet*, pages 81–92. Springer.

Kramer, J. and Magee, J. (2007). Self-managed systems: an architectural challenge. In *Future of Software Engineering, 2007. FOSE'07*, pages 259–268. IEEE.

Lehmann, G., Blumendorf, M., Trollmann, F., and Albayrak, S. (2011). Meta-modeling runtime models. In *Proceedings of the 2010 International Conference on Models in Software Engineering*, MODELS'10, pages 209–223, Berlin, Heidelberg. Springer-Verlag.

Maes, P. (1987a). Concepts and experiments in computational reflection. In *Conference Proceedings on Object-oriented Programming Systems, Languages and Applications*, OOPSLA '87, pages 147–155, New York, NY, USA. ACM.

Maes, P. (1987b). Concepts and experiments in computational reflection. *SIGPLAN Not.*, 22(12):147–155.

Mongiello, M., Grieco, A. L., Sciancalepore, M., and Vogli, E. (2015a). Adaptive architectural model for future internet applications. In *Proc. of the 5th International Workshop on Adaptive services for future internet*.

Mongiello, M., Pelliccione, P., and Siancalepore, M. (2015b). Ac-contract: run-time verification of context-aware systems. In *Software Engineering for Adaptive and Self-Managing Systems, 2015. SEAMS '15. ICSE Workshop on*, pages 106–115.

Palattella, M. R., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L. A., Boggia, G., and Dohler, M. (2013). Standardized protocol stack for the internet of (important) things. *IEEE Commun. Surveys & Tutorials*, 15(3):1389–1406. doi:10.1109/SURV.2012.111412.00158.

Pelliccione, P., Tivoli, M., Bucchiarone, A., and Polini, A. (2008). An architectural approach to the correct and automatic assembly of evolving component-based systems. *Journal of Systems and Software*, 81(12):2237–2251.

R. de Lemos et al. (2013). Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, pages 1–32. Springer Berlin Heidelberg.

Riccobene, E. and Scandurra, P. (2015). Formal modeling self-adaptive service-oriented applications. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pages 1704–1710.

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

Salvaneschi, G., Ghezzi, C., and Pradella, M. (2013). An analysis of language-level support for self-adaptive software. *ACM (TAAS)*, 8(2):7.

Strang, T. and Linnhoff-Popien, C. (2004). A context modeling survey. In *Workshop Proceedings*.

Swetina, J., Lu, G., Jacobs, P., Ennesser, F., and Song, J. (2014). Toward a standardized common m2m service layer platform: Introduction to onem2m. *IEEE Wireless Communications*, 21(3):20–26.

Tan, L. and Wang, N. (2010). Future internet: The internet of things. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 5, pages V5–376. IEEE.

Torjusen, A. B., Abie, H., Paintsil, E., Trcek, D., and Skomedal, Å. (2014). Towards run-time verification of adaptive security for IoT in eHealth. In *Proc. of the 2014 European Conf. on Software Architecture Workshops*, page 4. ACM.

Vögler, M., Li, F., Claeßens, M., Schleicher, J. M., Sehic, S., Nastic, S., and Dustdar, S. (2015). Colt collaborative delivery of lightweight iot applications. In *Internet of Things. User-Centric IoT*, pages 265–272. Springer.

Vogli, E., Ben Alaya, M., Monteil, T., Grieco, L. A., and Drira, K. (2015). An efficient resource naming for enabling constrained devices in smartm2m architecture. In *IEEE International Conference on Industrial Technology (ICIT) 2015*, pages 1832–1837.

Weyns, D., Iftikhar, M. U., de la Iglesia, D. G., and Ahmad, T. (2012). A survey of formal methods in self-adaptive systems. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, pages 67–79. ACM.