ISSN 2007-9737

An IoT Autonomic Architecture for Self-replicating and Self-assembly Systems

Mario Siller*, David E. Ramirez

CINVESTAV, Electrical Engineering and Computer Science, Guadalajara, Mexico

mario.siller@cinvestav.mx

Abstract. The number and diversity of Internet of Things (IoT) application domains are continuously increasing as the connected devices are doing exponentially. This has led to new challenges for the management, maintenance, and evolution of IoT systems. These issues have been approached by enabling adaptability and self-management capability (autonomic computing) in such systems. However, the focus on this area has been more on the behavioral or functional side (algorithmic) of systems rather than on the architecture and evolution of the high-level system design. This paper addresses this aspect by studying and proposing two new self-* properties for autonomic systems: *Self-Assembly and Self-replication.* These systems are referred to as *extended autonomic systems.* The proposed architecture is based on a multi-agent approach, biologically inspired, and on the *principle of maximum potential functionality:* "installed hardware capacity, enabled software capacity", Although such behavior could lead to having source code that does not run on the devices (Passive Code) despite module beloging to the same community. The functional unit defined to perform self-assembly and self-replication of system functionality is located at the source code class level. The proposed architectural design and algorithms were validated through the implementation of "Quines" in Python, static code testing, and an agent-based simulation. The experimental test scenarios consist of cyber-physical hydroponic systems for urban agronomy which were simulated as agent communities. Results show that the number of simulation cycles required for code composition and distribution increases reasonably as a function of the agent's community size and the number of functional classes involved in Self-assembly and Self-replication. Finally, it is shown that an IoT system can replicate part of its functional design in another system that requires it.

Keywords. Autonomic properties, MAS, IoT, Architecture, self-Star, IoT design self-assembly, self-CHOP.

1 Introduction

The number and diversity of application domains for IoT systems are constantly increasing. This is reflected in the number of devices and technologies involved in the design of IoT systems. This has led to the identification of new challenges for the administration and evolution of these systems due to their complexity, in addition to raising the question of whether it is possible to automate the design of these systems to facilitate their deployment and operation. Different approaches have been employed to address the problem of IoT system administration and its implications. One such approach is that of autonomic computing, as proposed in [31] and [36]. The notion of autonomic computing, as described in [31], stipulates that autonomic systems must have the capacity to self-awareness to facilitate decision-making processes and thus minimize the need for human intervention and increase their capacity to self-management. This notion encompasses a range of functions, including the ability to automatically configure and reconfigure themselves, among other tasks. This notion is encapsulated in the four core autonomic properties outlined in [36], namely Self-Configuration, Self-Healing, Self-Optimization, and Self-Protection, collectively referred to as Self-CHOP. The autonomic properties previously mentioned are widely accepted in the literature and subsequent research on autonomic computing mostly focuses on the utilization or application of these properties. Only a few works have proposed expanding the autonomic properties by seeking to provide the systems with other additional functionalities. However, these properties are associated with the maintenance and operation (O&M) and evolution stages of the system. It is clear that the management of changes in a system, which enables new capacities and involves the reuse of architectural and functional elements through interactions with other systems, has not been addressed. The inclusion of properties that enable the reuse of architectural elements would extend

the self-management of systems in this context and would thus further reduce human intervention. This, in turn, would achieve a stronger and more extensive self-management of autonomous systems.

In this work, an autonomic architecture for IoT systems is proposed, in which two novel properties are defined: *self-replication* and *self-assembly*. The introduction of these properties would enable systems to change their configuration or architectural structure without the need for human intervention, therefore facilitating the management aspects, design, implementation, system scalability, and deployment of similar new systems or within the same application domain. These new self-* properties are intended to address software change and reuse at different levels of abstraction, ranging from microservices and services (functions) to entire architectural elements such as subsystems, modules, and components. *Self-replication* would allow systems to replicate change elements autonomously, thereby facilitating the evolution between similar systems. The capacity for *self-assembly* would allow systems to reorganise, mutate, adapt, and assemble functionalities or architectural elements from other systems, and change themselves autonomously in response to the discovery of new change elements such as functions or architectural elements in similar systems.

2 Background

First, the concepts of *self-assembly* and *self-replication* are specifically presented from multiple fields of knowledge. Then, self-management in autonomous and self-adaptive systems is addressed to frame the notion of *self-assembly* and *self-replication* that will give rise to our proposal in Section 5, in addition to identifying relevant information and design assets.

2.1 Self-assembly and Self-replication

This subsection reviews the concept of self-assembly and self-replication in the framework of different disciplines such as biology, manufacturing, robotics, computing and IoT.

In [55] Raphael Plasson defines Self-replication as *"a property of a system enabling it to make a functional and independent copy of itself"*. On the other hand, as defined in [55], self-assembly can be described as the autonomous organization of components into patterns or structures without external intervention. Another proposed definition is presented in [39] where it is described as *"the ability of a system to form itself from distributed elements automatically"*.

The properties of self-assembly and self-replication in a system are closely related to self-configuration or self-adaptive computing, since once any functionality is replicated and assembled in a system, the system is able to adapt to use this new feature. The remainder of this section introduces the concepts of self-replication and self-assembly through various fields of knowledge.

2.1.1 Biology

In the domain of biology, an instance of self-replication can be observed in the process of DNA replication and assembly. According to [46], DNA replication exhibits a semiconservative nature, which leads to the assembly of new DNA molecules occurring predominantly in complementary pairs. This complementarity is both significant and serves as a source of inspiration for the concepts proposed in this paper.

2.1.2 Manufacturing

In the manufacturing area, self-assembly of parts has been introduced; this stems from the way the parts themselves were designed, as explained in [43] and [42]. This means that for a given designed part it can only be assembled with one other part.

2.1.3 Robotics

Another important field with contributions in self-replication and self-assembly is robotics, where approaches such as the one presented in [60] is based on robotic enzyme which acts in a similar way as a biological enzyme in assembling the parts scattered throughout the environment to form a single structure or replica. This process is limited by the amount of parts or material available in the environment. In contrast, software systems are not subject to this limitation because the resource used for self-assembly is the source code itself and no other material is required. Therefore, in this context, as long as the source code remains available, self-replication and self-assembly can take place.

In [21] a comparable robotic self-replication strategy is proposed, wherein a machine has the capacity to replicate itself by assembling or constructing its components. This process involves the creation of a replica module by module until the system is complete, and this is executed by a module specifically designed for the purpose of self-replication.

2.1.4 Computation

In the area of computing, Von Neumann's cellular automaton [50] marked the foundational approach to self-replication, introducing a framework that inspired subsequent studies on self-replicated systems.

The term "self-replication" has been employed within the field of metaprogramming, which is defined as the capacity to generate programs capable of either writing or manipulating other programs, including themselves. Programs that exhibit this capacity are referred to as "quines" in [29].

2.1.5 IoT

In [39] Self-assembly in IoT has been explored primarily from the perspective of service composition, where services dynamically integrate to create higher-level functionalities.

However, this particular instance of self-assembly does not encompass the architectural intricacies of the systems but rather focuses exclusively on the functionality of a service that is derived from the composition process. This is discussed in detail in Section 4.

3 Related Work

This section presents the state of the art in the areas covered by our proposal. The first subsection addresses the field of study of autonomic properties.

The properties defined as self-CHOP are considered in conjunction with the 3.2 Subsection, which focuses on IoT architectures, to propose two new autonomic properties and an IoT architecture that implements them in its design. The subsequent subsection, "IoT Architectures," presents the approaches and technologies that were taken into account in the design of this type of architecture.

Furthermore, the subsection addressing the resusability of assets is presented. In this subsection, the works most related to the areas of our proposal are presented, and the considerations to be taken into account when reusing design assets are explored in depth.

3.1 Autonomic Properties

The existing literature on autonomic properties has focused primarily on defining the functionality of each property, maintaining the original nomenclature, or proposing an architectural design to provide a system with one or more of these properties. Consequently, there has been a proliferation of descriptions for each proposed property. Some definitions of autonomic computing and its properties are reviewed in [40] and [49] in which the commonalities and differences in these concepts are discussed. In [9] the operation of each defined autonomous property is formally described from a proposed formal model that represents a system as a graph where the vertices represent processes and the communication links at the edges. In [65] new autonomic properties are presented such as self-anticipating, self-assembling, self-awareness, self-configuring, self-critical, self-defining, self-governing, self-installing, self-managing, self-organized, self-reflecting, self-similar, self-simulation, and selfaware. However, there is no detailed description, justification, or formalization of the properties proposed in this paper. For instance, they defined self-assembly as *"Assembly of models, algorithms, agents, robots, etc.; selfassembly is often influenced by nature, such as nest construction in social insects. Also referred to as selfreconfigurable systems"*. However, although the definition considers the notion of assembling elements in a system, it is not clear how to perform this operation, in addition to presenting ambiguity in which elements can be assembled. Furthermore, any assembly capability must be incorporated and derived from the system architecture design, a point not addressed by the authors in [65]. While the definition establishes a link between the property of self-reconfiguration and self-assembly, it fails to provide a comprehensive explanation or substantiation of this relationship.

3.2 IoT Architectures

In recent years, there has been an exponential increase in the number of IoT systems. This proliferation can be attributed to two major factors: the increasing number of devices capable of connecting to the Internet and the growing demand for intelligent solutions in various sectors. However, challenges such as scalability, management, and swapping devices in the system, have emerged as key issues in these systems. In addressing these challenges, numerous architectures have been proposed to minimize human intervention and standardize their operation. Many of these architectures are based on the RAMI architecture [1] and take

a service-oriented approach. The development of these architectures was motivated by the objective of reducing the management effort required for applications deployed on IoT systems. An architecture of this type is presented in [16], which has the following layers: application layer, security layer, distributed execution layer, modeling and validation layer, connectivity layer, and physical layer.

In addition, [68] presents an IoT autonomic architecture proposal for IoT systems with a focus on self-* autonomic property of self-configuration. The objective of this architecture is to enable the transit from previous IoT semi-autonomic systems to full autonomic IoT systems. The authors autonomic self-configuration design is based in Utility Theory and validated on an urban agriculture testbed and simulation. Although results are presented from the case study proposed in urban agriculture, validation through said case is limited since other application domains of the IoT are not addressed, and complementary approaches to the utility theory used are not explored.

On the other hand, adaptive architectural designs have also been investigated in the quest for greater autonomy for systems. These architectures are based on the principle that a system can address certain challenges without human intervention by adapting its operation. From this perspective, [18] presents an architectural pattern for IoT systems that prioritizes adaptation. This architectural pattern uses the MAPE autonomic loop, as proposed in the Master/Slave (M/S) Pattern described in [72]. This M/S pattern is designed to facilitate self-adaptation and to distribute tasks between two entities: the regional planner, which is responsible for analysis and planning, and the local nodes, which perform monitoring and execution tasks. The distribution of tasks across different nodes centralizes control in the regional planner, facilitating the implementation of high-level instructions for local adaptations. However, reliability is not addressed in any of the patterns [18], [72] since no failure event of a regional (master) planner node is considered.

3.3 Reusability of Assets

Reusability has been addressed through the services-oriented approach, where the element to be reused can be considered a service provided by a software agent. For this reason, it is possible to relate agents to services and even generate one-to-one, or one agent to many services relationship models. In [28], an integration of MAS and SOA for industrial automation is presented. The elements of the control architecture are the orchestrator, the orchestration engine (EO), and the Decision Support System (DSS). The orchestrator is referred to as ”a composite toolchain” and is responsible for facilitating the resolution of any conflict that may arise between the DSS and the orchestration engine. The orchestration engine is a part of a service-oriented middleware that executes a sequence of services offered by different devices. The DSS is a component integrated through a MAS that provides support for the decision-making/conflict resolution process. Nevertheless, this architectural design proposal is devoid of both a validation mechanism and a case study.

Likewise, [33] mentions the mapping of the SOA approach and IT systems. Furthermore, some factors must be taken into consideration regarding services. The initial element to be considered is the specificity of the service in question, particularly its reusability. In [45], an analysis is conducted of the relationship between generality and reusability in an architecture based on self-adaptation. This analysis is based on the idea of minimizing human intervention in two main fields. When a solution with high generality is developed, it must be adapted to each specific case without requiring significant effort by the developers. Conversely, when developing with a focus on reusability, the objective is to ensure that the system can be utilized without the necessity for modification to accommodate another system.

In a similar vein, the seminal work cited in [33] studies the integration of the SOA approach with IT systems. This study suggests that a number of factors must be considered in regard to services. This includes the granularity of the services and the notion of service composition. These factors and approaches are critical to the design of IoT-related services and applications.

One approach to defining service granularity is outlined in [20]. The authors propose a framework, termed ”Snowball,” predicated on the iterative application of rules to determine which services (applications involving only software) should be integrated as part of the same task and another set of rules to map tasks into IT services (applications which include software and hardware). The primary objective of this framework is to minimize the number of messages between services in the same system. Consequently, highly coupled systems that are subject to this framework can result in coarse-grained services, making it difficult to configure them precisely. Conversely, fine-grained services could be assembled or replaced in a simpler and more controlled manner, with this change having a comparatively reduced overall impact.

The evaluation of Quality of Service (QoS) in systems with service composition is a crucial aspect that facilitates the parameterization of factors such as service performance and its comparison with other analogous composite services or with identical objectives. This type of metrics enables the evaluation of functionalities that are replicated and self-assembled between similar systems. In the context of cyber-physical-social systems, a method for assessing functionalities that share common goals yet vary in their implementation is presented in [70]. This method is similar to those proposed in [25] and [14], in which compositions with multiple objectives are also evaluated.

3.4 Related Work Discussion

A detailed review of the current state of the art indicates that the interventions facilitated by autonomic computing can be enhanced and executed across diverse scales upon incorporating novel properties. This observation reveals the importance of autonomic properties in IoT systems and puts their potential benefits into perspective with an expanded definition of autonomic computing.

Furthermore, the concept of asset reusability, as previously documented, suggests the potential benefits that systems might derive from the automatic reuse of assets.

4 Problem Description

As stated previously, the literature on the self-management of software systems has primarily addressed the concept of autonomic computing, as established by [31]. This self-management approach is primarily concerned with fostering autonomy from human intervention, particularly during the maintenance and operation (O&M) phase. This approach does not consider the management of changes in the system of an architectural nature that can be used to enable new capacities in the system through the reuse of architectural and functional elements obtained from the interaction with other systems. This novel approach aims to facilitate the extension of self-management in this context, thus reducing the need for human intervention and enhancing the autonomy of the systems in question. Without this approach, such systems would possess only endogenous self-management capacities; with this approach, however, they gain both endogenous and exogenous self-management capacities, which may be referred to in this work as hybrid self-management in terms of the reuse of architectural and functional elements from other systems. In order to address this approach, it is proposed that the definition of autonomic computing be extended by the addition of two new properties: The term “self-replication” is defined for the purposes of this work as “the ability of a system to make copies of itself or parts of it.” Similarly, the term “self-assembly” is defined as “the ability of a system to take another system or parts of it and add all or some of the functionalities of this system to itself.” The two new properties are fundamental to enabling architectural modifications in a system through the reuse of architectural elements and functions from other systems. This process entails the automated integration of these components within the receiving system and the automated replication of the necessary elements by the sending systems.

From an engineering perspective, Internet of Things (IoT) systems are particularly conducive to testing and developing the characteristics of self-replication and self-assembly due to several key factors. In addition, there are already proposals for integrating the known autonomic properties into IoT systems. This approach has led to the emergence of a new category of systems, known as autonomic IoT systems, which are capable of autonomous operation and self-management.

The initial key factor to be considered is that self-replication and self-assembly aim to provide systems with the capacity to assemble architectural or functional components autonomously.

Consequently, autonomic IoT++ systems (i.e. autonomic IoT systems with these new properties) will be able to increase or change their operation in accordance with the architectural or functional elements they automatically assemble, thereby enhancing the flexibility of the system.

This is a significant consideration in light of the distributed nature of IoT systems and the inherent challenges associated with modifying the operational parameters of an already deployed autonomic IoT system.

A further significant consideration is the heterogeneity of IoT systems and the dynamic environments in which they are deployed. Concerning the heterogeneity of IoT systems, these systems can be widely benefited since, despite the heterogeneity, new devices can be integrated into the system with minimal human intervention.

Similarly, in the event of a change in the environment, the system will be able to modify its operation by assembling and replacing components almost independently from the system administrator. For these reasons, IoT systems are an ideal platform for the development and testing of these new autonomic properties.

5 Proposal

This section proposes an extension of autonomic computing in IoT systems by introducing two new properties: self-assembly and self-replication. These properties supplement the conventional properties of autonomic computing, self-configuration, self-optimization, self-healing, and self-protection. The proposal is structured into two main sub-sections. In the initial section, a formal definition of extended autonomic computing is presented, wherein the theoretical and formal bases delineating how self-assembly and self-replication capacities extend the concept of autonomic computing in IoT systems. The second subsection provides a detailed description of the extended IoT autonomic architecture, describing its constituent components and explaining the algorithms that facilitate the implementation of these new autonomic properties in IoT environments.

5.1 Definition of Autonomic Computing ++

A formalization of the self-assembly and self-replication properties is presented in this subsection. The objective of these properties is to facilitate self-management and adaptation in IoT systems with regard to the integration of new devices or source code updates in these systems, in addition to enabling the replication of their functionalities.

5.1.1 Formal Model

A formal and modified model based on [9] is presented below to specify the autonomic properties proposed in this work.

5.1.2 Base Model

An autonomic++ IoT system consists of n processes P $(n \geq 1)$ and an undirected graph $G = (V, E)$ where G represents the current version of the system and the vertices V depicts the *processes* of the system, and the edges E represent the *communication links* between processes $E \subseteq p \times p$. For each process $p_i \in V$, let $N(i)$ be the representation of the neighbors of i: therefore, $(i, j) \in E \Leftrightarrow j \in N(i)$ and $i \in N(j)$, for each process p_i a parameter vector $op_k \in \begin{pmatrix} \cdot \\ \cdot \\ \cdot \end{pmatrix}_k$ is associated representing the *operational parameters* of the type k process, where k is an index of *process types* that are present in the system. The association of process p_i and its corresponding operational parameters op_k is performed with the function $par : p_i \rightarrow op_k$ and the size of the operating parameter vector can vary for each process type. The set of all operating parameters of the system in its current state $OP = \{op_1\} \cup \{op_2\} \cup \{op_3\}...\{op_k\}$ is composed of the union of the operating parameters of each process type.

Each process p_i executes an algorithm consisting of a set of *actions*, $a \in A$, where sa is a predicate involving the variables of p_i and its neighbors, and when executed it updates the local state, ls_i, obtained from $s(p_i)$, where $s : p_i \rightarrow ls_i$.

The *global state* S (also known as Global Configuration) consists of the *local states*, ls_i of all processes p_i and corresponding op_i; this is represented as $S = \{ls_i, op_i\}$. The set of possible actions A is composed of two sets of actions: the set of *controlled actions*, CA and the set of *uncontrolled actions*, UA. An action can also be of type IA, *internal action*, or EA *external action*. All CA are IA while UA can be either IA or EA. An internal action can change an op_i. All CA are of type IA, and under normal conditions do not cause a failure in the system, since they must be valid and correct. An external action EA is an action originating within the system's environment and can be generated by an external entity. This action can cause changes in the external environment $ExEnv$.

The external environment $ExEnv$ consists of parameters that the system can only read, but cannot modify or intervene through IA. An UA can produce changes to network topology, ad hoc fault induction, arbitrary system configuration, or security attacks. Therefore, they can affect system performance or even partially or completely disable it. These actions can also be called fault actions or adversarial actions.

Correct system operation can be interpreted as "The system operates and behaves according to the specifications established in the different replication sources, assuming the existence of compatibility between the replicating systems". This is represented by the invariant OK. Furthermore, the system is defined to be in a legal or consistent configuration concerning a given external environment $ExEnv$ when the predicate OK is true.

5.1.3 Self-Healing

The system exhibits the property of self-healing with respect to a subset of uncontrolled actions $SH \subseteq UA$ if the occurrence of actions SH results in at most a temporary incorrect functioning or behavior in the system, but subsequently leads to the fulfillment of predicate OK.

5.1.4 Self-Configuration

The system exhibits the property of self-configuration with respect to a subset of controlled actions SC that are defined as $SC \subset CA$. This property is characterized by the following conditions: first, the occurrence of an action SC must cause a change in the operating parameters op_i of a process i; second, this change must result in a transition from S to S'; and third, the predicate OK must be satisfied. It is important to note that S and S' are regarded as two successive global states.

5.1.5 Self-Protection

The system has the property of self-protection with respect to a subset of uncontrolled actions $SP \subseteq UA$, if the occurrence of the actions SP is managed by the system with the objective of maintaining compliance with the predicate OK.

5.1.6 Self-Optimization

The system has the property of self-optimizing with respect to a set of operating parameters op_i associated with the processes p_i such that by changing the values of the operating parameter vector, a pre-defined objective function in p_i is maximized (or minimized), and the predicate OK is satisfied.

5.1.7 Self-Assembly

The system has the self-assembly property if it is able to receive a new vertex p_{new}, expand E with the pairs $\{(p_{new}, p_i)\}$ where i corresponds to all the vertices with which p_{new} would have communication, as well as the set of operating parameters with $OP = OP \cup \{op_{new}\}$, generating a transition from G (current version) to a new version of the system G' that satisfies the predicate OK.

5.1.8 Self-Replication

Consider a system, denoted by G, capable of communicating with any other system, H, that complies with the IoT Autonomic++ architecture. The system G is said to possess the self-replicating property if it can transfer (replicate) one or more processes or functions, p_i, and the associated operating parameters, op_i, while ensuring the fulfillment of the predicate OK on the receiving system H.

5.2 Description of the IoT Autonomic++ Architecture

This subsection describes the proposed architecture for an IoT Autonomic++ System. It outlines the main components, communication mechanisms, functional layers, and algorithms involved in self-assembly and self-replication. In addition, it provides a comprehensive technical specification that will guide the implementation of this architectural design.

5.2.1 Design Fundamentals

This sub-subsection titled “Design Artifacts and Fundamentals” presents the theoretical concepts and design elements that form the foundation of this paper’s IoT Autonomic++ architectural proposal.

MAS The field of multi-agent theory offers a foundational approach to comprehending and designing interactions among autonomous entities, or agents, that collaborate to accomplish shared objectives or goals. Each agent in a multi-agent system is endowed with the capacity to act autonomously, make local decisions, and adapt to the dynamic environment surrounding it.

Numerous works, including [59] and [28], have modeled functions associated with the control of a cyber-physical system or an IoT system as Multi-Agen System (MAS). Similarly, in other works (e.g., [3], [53], [23], [69], and [37]), tasks related to providing a system service are performed by agents that execute actions associated with those tasks. This implies a close relationship between multi-agent systems theory, services in service-oriented architectures (SOA), and the design and implementation of IoT systems.

SDN The theory of software-defined networking (SDN) represents a fundamental approach to the flexible and dynamic management of network infrastructures, whereby the control plane and the data plane are separated. This paradigm enables the centralization and programming of network control decisions while facilitating the decentralized management of routing and data packet forwarding. In the context of extended autonomic IoT architectures, the SDN paradigm is an essential tool, as it allows us to separate the data plane (core system functionality) and its control plane (autonomic computing extended), as illustrated in references [13], [22], [4]. This notion enables the management of the behavior of specific services for the administration of the IoT system and the IoT devices, which are mapped one-to-one with agents

of a multi-agent system in our architectural model. In the context of our research, the SDN approach offers a paradigm wherein IoT nodes can be regarded as programmable components, thereby enabling the system to self-configure in run time. This flexibility is of paramount importance in IoT environments, where nodes can be added or removed on a dynamic basis and where functions must be replicated or assembled without service interruptions. By centralizing control and the ability to reprogram the system at the logical level, the SDN approach can be employed to achieve a more flexible and cost-effective solution.

Architectures In formulating our proposal, we have considered the architectures of cyber-physical systems and IoT systems as presented in the existing literature. These architectures define the structure, components, and interactions of the devices and services that comprise a cyber-physical ecosystem. In light of the exponential growth of IoT devices and the increasing complexity of their applications, IoT architectures must evolve to incorporate autonomic capacities that enable the self-management of systems. The design of an extended autonomic IoT architecture, as explored in this research, is based on three key principles: flexibility, modularity, and distributed autonomy. These principles allow devices to not only interact with each other but also evolve during any operating condition change.

The majority of the presented architectures propose a layered model with a structure analogous to an infrastructure layer (sensing), a controller and virtualizer layer, and an application layer, among other layers such as storage or security. This is illustrated in references [3], [58], [32], [16], [8], [24], [11], [41], [2], and [66]. Similarly, in other references, the layers are the physical layer (sensing), network layer, and service layer (see references [56], [35], [54], and [67]). The primary distinction between these architectures is the location of data processing, which is either situated in the intermediate layer (edge computing, information management, central data management, or data layer) alongside network control or located within the application layer. Conversely, some architectural proposals concentrate on the operation of systems based on the theory of multi-agent systems, as described in [28], [53], and [26].

Another fundamental aspect identified in autonomic IoT architectures is the importance of the autonomic control loop (ACL MAPE-K), which is present in multiple works, including [5], [47], [68], and [18].

Autonomic Computing This approach is particularly pertinent in contexts such as the IoT, where the proliferation of devices and their heterogeneity render manual management impractical. In the context of this research, the concept of autonomic computing is expanded. The majority of recent research in autonomic computing has focused on implementing existing autonomic principles and concepts, rather than extending them or contributing to the underlying theory in the field.

This is shown in the following references: [68], [40], [10], [51], [15], [9], [65], [63], and [6]. The expansion of the concept of autonomic computing is achieved through the introduction of two new additional properties: self-assembly and self-replication. The introduction of these new capacities enables IoT nodes to self-manage and autonomously assemble new functionalities and replicate capacities between different devices in the system. This facilitates the scalability and continuous evolution of the IoT ecosystem. In this work, the approach taken is to conceptualize IoT devices as autonomous agents or elements, as proposed in [49].

Autonomic computing not only enhances the resilience of IoT systems against failures or unforeseen changes, but also facilitates greater operational efficiency by optimizing resources without external intervention. This can be viewed as an evolution of self-adaptation, as previously discussed in [27]. This is of particular importance in scenarios where systems must rapidly adapt to changing environments, such as those encountered in smart cities or Industry 4.0 or beyond. A critical aspect of incorporating autonomic computing into a system design involves a thorough evaluation of its potential effects and advantages. This is addressed in [44], [48] and [19] where metrics such as quality of service, cost, granularity/flexibility, robustness, degree of autonomy, efficiency, accuracy of awareness, interoperability, intelligence granularity, reaction time, among others. It is important to note that a number of these metrics are considered in the evaluation of the proposed architecture and the case study presented in Section 6.5.

Adaptability The capacity for self-adaptation represents a key capability for systems operating in environments subject to change. This property enables them to undergo dynamic adjustments in response to alterations in the surrounding environment or their intrinsic internal states, without needing external intervention. As mentioned in [71], the concept of self-adaptation has been primarily linked to automatic adjustments to a system's behavior due to changes

in its environment. Self-adaptation is crucial for optimizing system performance, enhancing resilience, and extending the lifespan. As stated in [12] and [6], a system with self-adaptive capacities can continuously monitor its environment and internal parameters to identify deviations or anomalous conditions and implement necessary adjustments at runtime.

This research explores the concept of self-adaptation within the context of an extended autonomic IoT architecture, which incorporates the additional properties of self-assembly and self-replication. These properties enhance the system's capacity for self-adaptation. This, in turn, enables the system to reconfigure its functions, as well as to assemble new capacities and replicate functionalities to other nodes in the system when necessary. Consequently, the self-management enabled by self-adaptation encompasses more than mere fundamental operational adjustments. When coupled with autonomic computing and the two proposed novel properties, it enables the system to proactively evolve and respond to the evolving demands of the environment. Furthermore, it provides flexibility to systems that assemble functionalities from other systems into themselves. This is particularly crucial in contexts such as smart cities and industrial environments, where conditions can undergo rapid change and where the capacity to adapt can be a determining factor in the success and sustainability of the system.

Other Design Considerations In the design of our proposal, we have considered some non-functional requirements, including aspects such as scalability, security, interoperability, and system adaptation. These have been previously identified in references such as [61] and [17]. On the other hand, in terms of stakeholders, the standard ISO IEC IEEE 280 [34] states that the architecture description shall identify the system stakeholders whose concerns are considered fundamental to the design. This group may include users, operators, acquirers, owners, suppliers, developers, builders, and maintainers.

5.2.2 Design Assets

The autonomic architecture proposed by Villela [74] (based on the level 4 architecture in [7]) was modified to include the properties of self-assembly and self-replication. In order to implement this, it is necessary to modify, add, and delete some modules present in the base architecture. In the architecture redesign, the following principles will be taken into account: the replication of programs (Quines), as outlined in[30]; the notions of Autonomic Element (AE), as described in [49] and [62]; the principles of distributed systems; the autonomic properties defined for the baseline architectural design; the MAPE-K control loop; and the autonomic properties proposed in [36]. In order to achieve this objective, it is necessary to modify the IoT architectural baseline [74] [7] and the functional algorithms. This modification will enable self-assembly and self-replication between systems through code replication. The result of this process will be an open system that can be reconfigured in a number of ways. These include the input or output of the devices, the configuration of system settings, objectives, or executable code updates. Code change is related to the field of [29] metaprogramming, which is characterized by programs that write or manipulate other programs or themselves (Quine [30]). In this work the objective is to achieve code composition by deploying new source code updates between devices belonging to the same community. This facilitates the integration of new features into the system once the new code has been compiled with the existing source code.

A fundamental aspect of this process is the management of self-replication and self-assembly. The replication and assembly of new code may involve the configuration of new functionalities, which could be achieved through a global configuration for devices within the same community. The configuration of new functionalities is addressed in multi-agent systems theory through control policies. These are defined as a set of actions, activities, plans, standards, procedures, and operating methods that govern the actions of agents [73]. The implementation of these policies is often facilitated by hierarchies or communities comprising sets of agents that share one or more common characteristics. An agent may be affiliated with one or more communities, as it may share characteristics or cooperative capacities with some of them and others with other communities.

The following sections present the architectural design hat has resulted from the proposed changes, see Figure 1 (proposed changes and new elements are coloured in gray and black, respectively).

Legacy Components

— App: The specification of this architectural element remains the same as in the original architectural baseline design. The principal objective of the

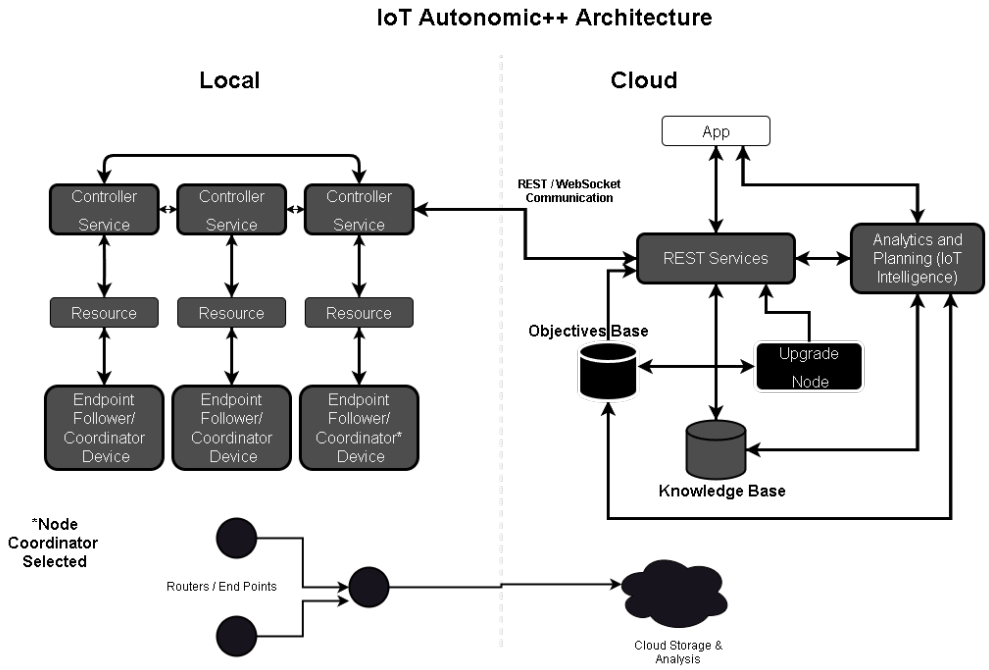

Fig. 1. Autonomous, Self-Assembling and Self-replicating IoT Architecture.

app element is to provide an interface that enables users to interact with the IoT system.

Proposed Modified Components

— Endpoint Follower/Coordinator: An Endpoint device can be either a Follower or a Coordinator. There is a coordinator node for every device community. A single node is designated as the coordinator based on a consensus algorithm to be chosen by the designer. The selection of a coordinator occurs when a system is started and there is no coordinator in the IoT device or when a failure occurs. In any case, both the Follower and Coordinator node roles/profiles must be implemented so that any node can assume the corresponding role as required. The coordinator of each device community will be responsible for facilitating communication between the IoT devices and the cloud layer for various purposes, including the transmission of sensed data from the IoT devices to the cloud and the modification of control policies from the objective base to the device communities. Moreover, the coordinator is responsible for the assembly of new versions of source code and its distribution. Conversely, the Follower device will possess functions associated with the primary functionality of the system, including data collection, processing, interaction with actuators, and communication with the coordinating device for the transmission of collected data, as well as the reception of control policies or new versions of source code.

— REST Services: This service facilitates communication between IoT devices and all elements in the cloud layer. Additionally, it is responsible for the monitoring of virtual nodes and the reporting of the status of Local (physical) nodes to detect any anomalies.

— Controller Service: it provides a communication channel among IoT devices. This channel enables collaboration between devices and the transfer of new data, facilitating the transfer of information between nodes of the same community and between nodes of different communities. Furthermore, the controller service provides access to the configuration and source code of the devices in communities. Thereby enabling the replication of source code or parts of it (using metaprogramming methods, such as Quine programs) to achieve a composition of a new source code that the receiving

(assembler) device itself will execute. The operation of this node is governed by the control policies defined in the objectives base. The Controller Service is responsible also for the the process of self-monitoring which is guided by high-level policies and inspired by multi-agent systems (MAS), as outlined in [73] and [64]. This functionality enables the device to report faults and attempt to correct them, otherwise abandoning or removing itself from the device communities to which it belongs.

— Knowledge Base: This element exhibits dual functionality. The principal function of this element is to store historical data collected from devices. It is also responsible for managing source code version control for IoT device communities in the context of self-replication and self-assembly, facilitating traceability of system evolution, and allowing for rollback if necessary.

— Analysis and Planning (IoT Intelligence): This architectural element is responsible for the analysis of historical data from the main functionality of the IoT system, derived from its application domain. The purpose of this analysis is to propose new control policies that improve overall system performance. These control policies will be incorporated into the Objective Base for subsequent implementation through the REST service node.

5.2.3 New Proposed Architectural Elements

— Objectives Base: This element is responsible for managing the control policies that govern the behavior of the communities of agents and their interactions. In the event that the IoT system administrator seeks to modify its configuration, the system is able to execute these changes without any interruption. Every configuration change involves generating a new version of the control policies. Once generated, these policies are distributed to the other devices through the REST service node and the Coordinator device.

— Updater Node: This element enables the IoT system administrator to modify Control Policies in the Objectives Base. Additionally, it facilitates remote source code updates and connection to other IoT systems that comply with the autonomic++ architecture. This enables self-assembly and self-replication between similar systems. This objective is met through the communication of the new source code to IoT community coordinators via the REST node.

5.2.4 MAS Architecture Representation

The agents that make up the proposed architecture are illustrated below, beginning with the definition of their nomenclature and acronyms. Each agent's nomenclature is associated with the function it performs in the architecture. These agents are: Follower Device Agent (FD-A), Coordinator Device Agent (CD-A), REST Service Agent (RS-A), Objectives Base Agent (OB-A), Knowledge Base Agent (KB-A), Update Agent (U-A), Analytics and Planning Agent (AP-A), and Application agent (A-A).

Figure 2 presents two perspectives of the multi-agent architecture. The left side of the figure provides a topological representation of the communication between the different modules and the system's interaction with a user and a system administrator. The red lines represent the system's internal communications, the blue rectangles symbolize the system elements, and the red circles indicate the external elements. A blue line represent the interconnectivity between internal elements and external elements. Figure 2 illustrates that communication between a user and the system is facilitated through the A-A, which, in turn, exchanges information with the RS-A, which is responsible for communication between most agents. The U-A agent provides a communication gateway with the system administrator, through this communication the administrator can make modifications to the source code and the objective base. The number of agents displayed on the right side of the figure is application-dependent. The elements highlighted in yellow correspond to unique agents in the system. In contrast, the elements indicated in green, purple, and orange correspond to non-unique agents that form communities of agents. Each community is characterized by a coordinator (CD-A) and at least one follower (FD-A).

It is imperative to underscore that with the functions delineated in each agent type proposed in the architecture, a correspondence with the functions established in an SDN/NFV architecture [52] can be established. Consequently, each agent can be located in the layer illustrated in Figure 3.

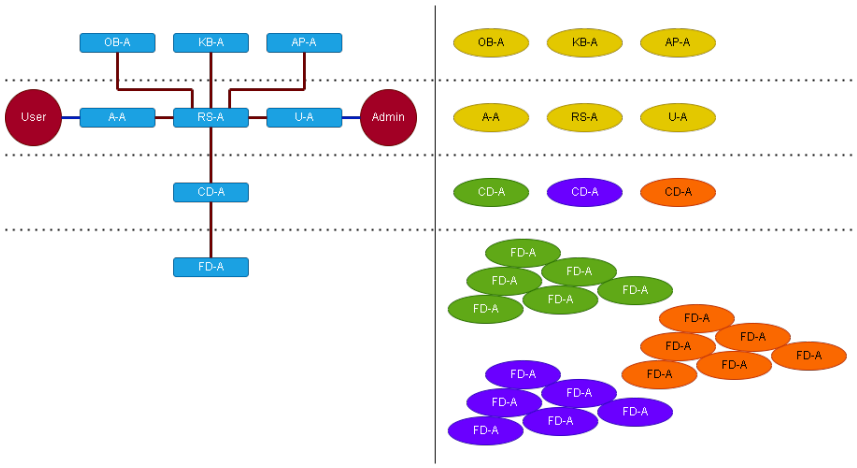

Fig. 2. Representation of the proposed architecture based on MAS.

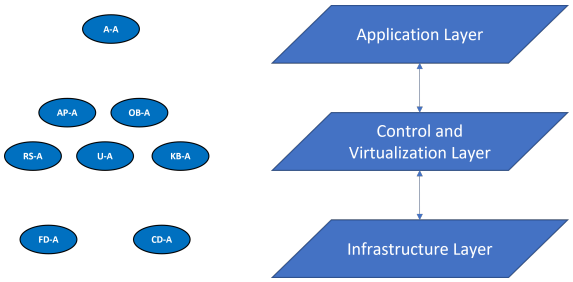

Fig. 3. Comparative of agents vs basic concepts of SDN.

5.2.5 Source Code Structure

Standardizing the source code structure across end-point/coordinator devices facilitates the comprehension and identification of functions/services and their dependencies.

This is pivotal to enabling self-assembly and self-replication of the source code. The proposed structure is based on the principle of modularity and it is shown as a template in Figure 4.

The organization of the code may not be limited to this structure; however, standardizing it and covering at least each aspect established in the proposed sections would be necessary for any deployment or application of the proposed architecture. All the subsections of the proposed structure are described below.

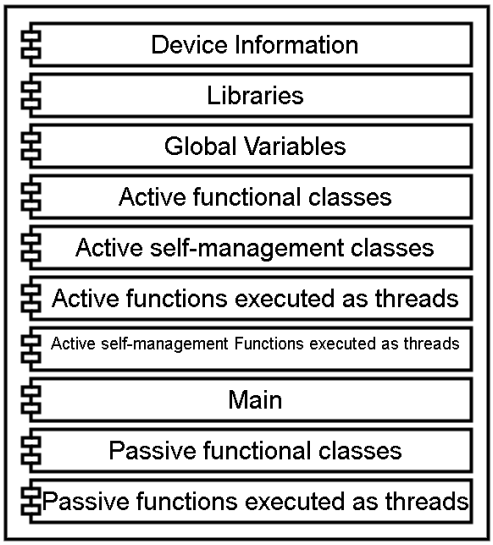

Fig. 4. Source code structure divided by sections.

Device Information The initial section, designated "device information", encompasses the ensuing sections: "SW Version", "HW Version", and "Capacity list". Sections *SW Version* and *HW Version* contains the corresponding version identifier, expressed as a numerical sequence. The capacities list contains the elements that comprise the IoT device, including actuators and sensors, which are specialized for a particular task.

Libraries The "Libraries" section contains the libraries employed in the source code for the "Endpoint/Coordinator" devices. The purpose of this source code module is to facilitate the analysis and comparison of libraries between two source codes that are assembled into a new source code version.

Global Variables The section contains the global variables for the "Endpoint/Coordinator" devices. These global variables can be used to modify the configuration of the IoT device functionalities.

Active Functional Classes The Active Functional Classes section contains the definitions of basic classes and functions related to each task that the system's core functionality can perform. Moreover, each functional class provides a list of capacities required for its execution within the system. The layout of the source code has been designed to facilitate maintaining order during automated analysis. This structure is shown in Figure 5. These classes are referred to as *"Active **functional** classes"* due to their association with the supported **functionality** of given IoT device.

Active Self-management Classes The Active self-management classes section contains all classes and self-management functions related to self-replication and self-assembly. It's worth noting that the system incorporates two categories of functions: first, those that facilitate source code management, including reading, interpreting, and generating new source code files. Second, functions related to system communication and control, including leader search, general follower listening, specific coordinator listening, response to survival messages, and survival queries.

Active Functions Executed as Threads The Active functions executed as threads section contains a function with algorithms for executing all of the system's primary tasks as threads, which may involve executing multiple classes, as well as the configuration variables associated with these tasks. Consequently, these algorithms utilize the fundamental functions provided in the Active Functional Classes.

Active Self-management Functions Wxecuted as Threads The Active Self-Management functions executed as threads section contains algorithms related to system self-management (autonomic properties), such as enabling follower or coordinator functions and their communication to enable self-replication and self-assembly. This section also describes the sequential or parallel execution of the tasks described in the 'Active Self-Management Classes' section related to the role of the follower device or the coordinator device.

Main The Main module is to contain exclusively the invocations of the core task and the autonomic computing threads. It should be noted that only invocations to functions executed as threads that are defined as active code are included. Consequently, in the event that a task cannot be executed due to a lack of capacities, a thread will not be enabled for that task.

Passive Functions Executed as Threads In contradistinction to Active Functional Classes, this module incorporates all the functional classes that the system has assembled in its source code, yet which cannot be executed because they lack the necessary capacities for their use (sensors, actuators, etc.); however, this passive source code follows the same structure as the active functional classes shown in the figure 5. These classes are referred to as *"Passive **functional** classes"* due to their association with the supported **functionality** of given IoT device.

Passive Functions Executed as Threads Passive functions executed as threads, on the other hand, incorporate the execution sequence that cannot be executed due to insufficiency of capacity on the device where the new source code was assembled. The code is stored, and if required, capacity (sensors, actuators, etc.) is added to the device. In this case, the code is converted into active code.

5.2.6 Architectural Views

In this Section, the proposed architecture is specified using the 4+1 view model of [38]. To facilitate comprehension of the various aspects of the architecture from multiple perspectives, several diagrams are presented for each view.

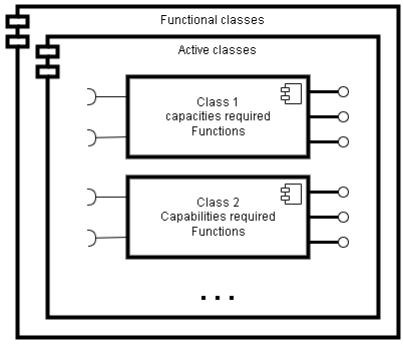

Fig. 5. Functional class module structure.

Logical view For this view the Figure 6 is presented. The diagram illustrates the data and functions associated with each node of the proposed architecture. Given that our proposal is a high-level architecture, it is not feasible to establish an implementation-level structure for the data and functions represented here.

Deployment View For this view Figure 7 shows a visual representation of the data provided and consumed by each node in the specified interaction. Figure 7 shows the communication flows between the nodes of the architecture and the content of the communication (control policies, source code, sensed data, etc.).

Process View For this view Figure 8 depicts the operation of the system when a device receives a new source code version or update *(updating source device)*. The same code reception process is also applicable in the event that an unknown device arrives in the community with a different code version, also referred to as updating source device. The process is described as follows: **the coordinating device** receives the source code from the *updating source code device* and analyzes it. Subsequently, it assembles (copies) the classes not present in its current code and generates a new version. Subsequently, the code is distributed to the community for compilation, and the execution of the new code is requested with a restart of all devices in the community.

Physical View This view is where the physical deployment of the system is illustrated. However, since this work is a high-level architectural design, it was decided to refer to Figure 1, which shows the separation between the local deployment with its devices and connections, and the cloud deployment with its nodes and connections.

Scenarios View (+1) For this view Figure 9 represents the high-level interaction scenarios between the user of the system, the administrator, and external Internet of Things (IoT) systems.

6 Experimental Work

6.1 Objective

The following experiments constitute a proof of concept, designed to validate the functionality of the proposed architectural design. The objective is to demonstrate that an implemented system based on the proposed architecture is capable of self-assembling and self-replicating functional architectural elements from other systems within the same IoT application domain, given that they follow the source code structure defined in 4. In these experiments, the self-assembly and self-replication duration of the system are measured in order to evaluate the scalability of these properties based on the number of IoT nodes belonging to the same domain.

6.2 Experimental Scenario Description

The experimental test bench compares the source code of two IoT devices from different systems running device software developed in Python. This software is uniform across all devices within a single IoT system, yet may exhibit variation between devices across different systems. In the experimental phase, the number of IoT devices utilized in system 1 is varied, employing 1, 4, 9, 19, and 49 devices/nodes, while a single device/node equipped with device software from system 2 is employed.

The devices were implemented using Docker images and containers, with one virtual machine per IoT device. In this sense, each virtual machine functions as an autonomic IoT device/node, with processing and communication capacities. This configuration emulates an IoT environment. The coordinator role is assigned to a device in System 1 (a system where the new source code will be self-assembled and replicated).

The device in System 2 establishes a connection with the System 1 network, thus initiating its interaction with the coordinator node of that system. In this scenario, the System 2 device possesses a new version of code. Consequently, the version is sent to the coordinator node of System 1, which initiates its self-assembly. The new

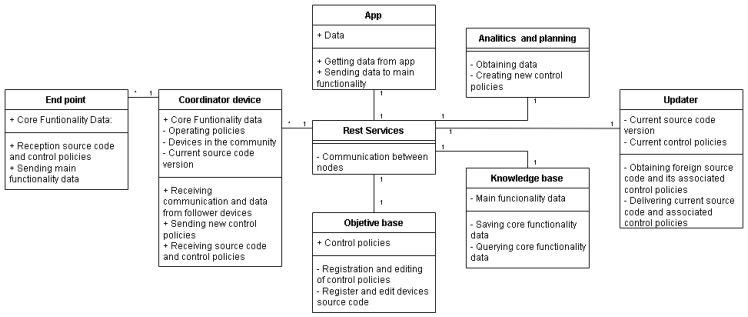

Fig. 6. Class diagram of the “autonomic IoT++ architecture”

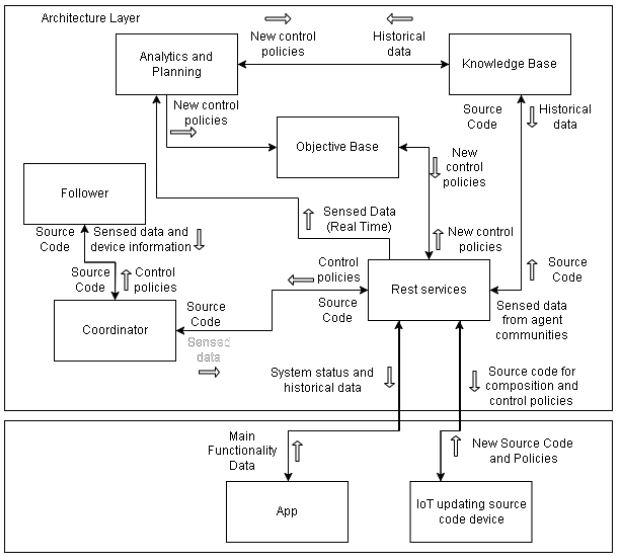

Fig. 7. Flowchart of the ”autonomic IoT++ architecture”

assembled code is then replicated to all network devices, including those that may belong to a different IoT system.

The specifications of the hardware computer used to run these virtual machines are: CPU (Intel Core i7-4710HQ, 4 cores, 3.5 GHz), RAM (8 GB DDR5, 1600 MHz), Storage (512 GB NVMe SSD), OS (Ubuntu 25.10), GPU (NVIDIA GTX860M, 2 GB), Network (MediaTek MT7902), Software (Docker 28.5.1). The source code of this experiment is available through GitHub in [57].

6.3 Self-assemble Implementation

Figure 10 and the following text describe the source-code-level self-assembly process executed by an IoT

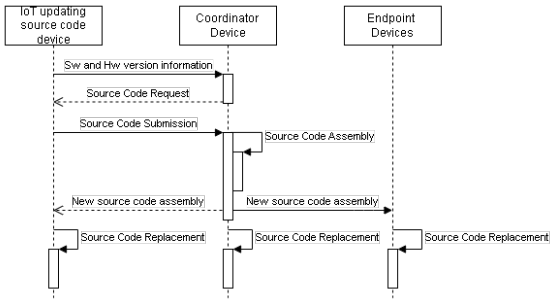

Fig. 8. Sequence diagram of the process of receiving a new version of source code

Coordinator Device when integrating functional classes from an IoT updating source code device.

The process begins with the reading of the section device information (see Figure 4), libraries, Active functional classes, Passive functional classes, Active functional threads, Passive functions executed as threads, Active self-management classes, and Active self-management functions executed as threads from the source code files of both the IoT Coordinator Device and the IoT updating source code device.

The subsequent assembly steps are performed: (1) A copy of the device information section from the IoT Coordinator Device source code is generated; (2) The libraries section is generated as the union (without duplication) of the libraries defined in the libraries sections of both the IoT Coordinator Device and the IoT updating source code device; (3) The global variables section is generated as the union (without duplication) of the global variables defined in the corresponding sections of both devices; (4) The Active functional classes section is generated by fully copying to memory the Active functional classes of the IoT Coordinator

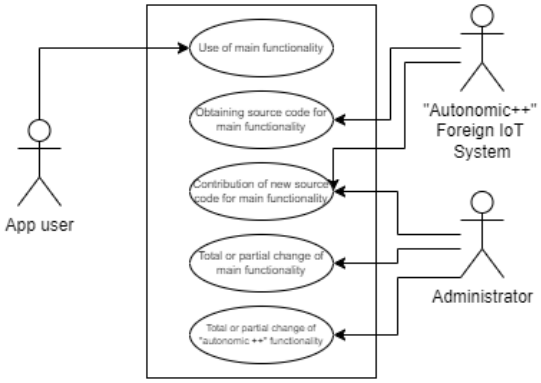

Fig. 9. Use case diagram of the "Autonomic IoT++ architecture"

Device and selectively copying only the previously unknown Active and passive functional classes from the IoT updating source code device that can be executed given the available hardware capacities; (5) The Active self-management classes section is generated by fully copying the Active self-management classes of the IoT Coordinator Device and selectively copying only the previously unknown Active self-management classes from the IoT updating source code device; (6) The Active functions executed as threads section is generated by fully copying to memory the Active functions executed as threads of the IoT Coordinator Device and selectively copying only the previously unknown Active and passive functions executed as threads from the IoT updating source code device that can be executed given the available hardware capacities; (7) The Active self-management functions executed as threads section is generated by fully copying to memory the Active self-management functions executed as threads of the IoT Coordinator Device and selectively copying only the previously unknown Active self-management functions executed as threads from the IoT updating source code device; (8) The main section is generated, including the invocation of the functions defined in the Active functions executed as threads and Active self-management functions executed as threads sections; (9) A section for Passive functional classes is generated, containing the classes that cannot be executed due to insufficient hardware capacities; (10) A section is generated for Passive functions executed as threads, containing the functions that cannot be executed due to insufficient hardware capacities; and (11) Finally, the new source code file is written by copying from memory all sections following the structure illustrated in Figure 4.

6.4 Experimental Implementation

A detailed specification of the functions is presented in Table 1. The following is a list of system sensors and actuators considered for the experiments:

Sensors:

- DHT22 (dht): Sensor to measure temperature and relative humidity.
- LDR or Light Sensor (ldr): To measure light intensity.
- PH Meter (ph): Sensor to measure pH levels of the solution.
- EC Sensor (ec): To measure nutrient concentration in the water.
- Water Level Sensor (wls): To monitor the amount of water in the reservoir.

Actuators:

- Water Pump (wp): To recirculate water in the system.
- Solenoid Valves (sv): To control the flow of water or nutrients.
- Nutrient and pH Dosers (nd): To add acidic, alkaline or nutrient solutions as needed.
- Fans (fans): To control ventilation and ambient temperature.
- LED Grow Lights (led): Act as an additional light source when light intensity is low.

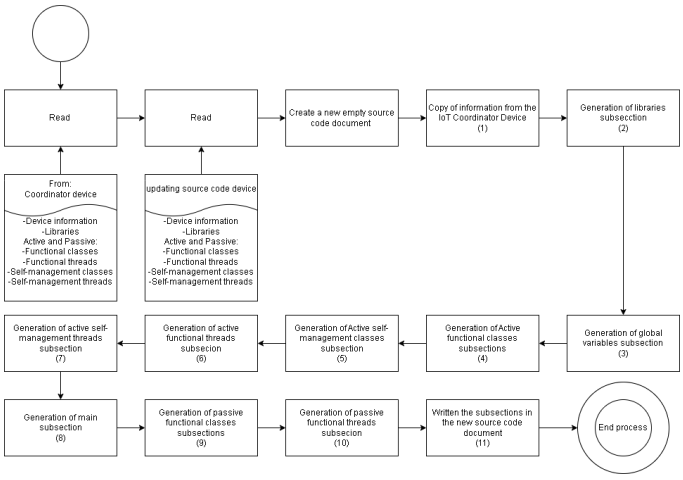

Fig. 10. Source-code-level self-assembly process executed by an IoT Coordinator Device

Each IoT device is equipped with an initial set of classes and capacities that is identical for all devices belonging to the same IoT system. Both are specified in the respective column of Table 2. Acronyms denoting classes (SW) are written in uppercase, whereas those denoting capacities (HW) are written in lowercase. The associated functions hardware dependencies expressed as **Function_name(dependent_HW)**, include: Measure temperature (dht), Measure humidity (dht), Water pump control(wp, wls), Water level (wls), Measure light intensity (ldr), Ph control(ph, sv), Nutrients control(ec, nd), Turn onLight (ldr, led), and Fan control (dht, fans).

To test the passive code classification performed by the Coordinator Device as specified in Figure 8, a dummy functional class and its corresponding function were incorporated. This functional class was named ”DHT-F” and its associated function ”MeasureTemperature-F”. In all experiments, both the class and the function are assumed to be part of passive code section of devices of systems 1. It is important to note that none of the IoT systems have hardware support for this class and function. In contrast, 3 specifies the functions classified as active on the devices of systems 1 and 2.

The tests were performed using a Docker Compose file that describes 1, 4, 9, 19, or 49 applications, a container name, and an IP address for each container. Each of these containers corresponds to an IoT device in System 1. A separate Docker Compose file was also utilized to execute a single container corresponding to the IoT device (from system 2) with a different source code version. Furthermore, a Python application was implemented, operating directly on the computer hardware.

This application generates a log of the following types of broadcast messages that occur: leader search, leader application, new coordinator notification, and restart notification. These messages were stored along with the IP address of the device transmitting the message and a timestamp for later analysis.

For analyzing and validating the Extended Autonomic IoT architecture (IoT Autonomic++), the independent and dependent variables are defined below. The independent variables correspond to the parameters intentionally manipulated during the experimental evaluation in order to observe and quantify their effect on the dependent variables.

Independent variables are as follows:

1. The capacities of the systems: The sensors and actuators available for the IoT devices participating in the experiment.
2. Number of classes and functions to replicate/assemble: The number of functionalities that will be attempted to self-replicate and self-assemble during a specific time interval.
3. Number of IoT devices: The total number of active nodes in the experimental network that are capable of participating in the self-replication and self-assembly processes.
4. The required capacities of each function to be replicated: The specific resources required for the execution of each functionality.

Table 1. Specification of the threads-functions.

Function	Classes needed for execution of the thread	Description
Measure temperature	DHT	The temperature measurement function relies exclusively on the DHT class, as this class encapsulates the methods required to acquire temperature data from the corresponding sensor.
Measure humidity	DHT	The humidity measurement function depends solely on the DHT class, which provides the necessary methods to retrieve humidity readings from the sensing hardware.
Water pump control	WP, WLS	The water pump control function depends on the WP and WLS classes. The WLS class is responsible for obtaining the current water level in the tank, while the WP class provides the mechanisms to activate or deactivate the water pump based on this information.
Water level	WLS	The water level monitoring function depends exclusively on the WLS class, which contains the methods required to sense and report the current water level.
Measure light intensity	LIGHT INTENSITY	The light intensity measurement function relies solely on the LIGHT INTENSITY class, which provides the functionality required to obtain ambient light intensity measurements.
Ph control	PH, SV	The Ph control function depends on the PH and SV classes. The PH class is used to obtain Ph measurements, while the SV class enables control of the solenoid valve, allowing the system to adjust the pH level accordingly.
Nutrients control	EC, ND	The Nutrients control function depends on the EC and ND classes. The EC class is used to obtain nutrient measurements, and the ND class is responsible for issuing the command to activate the nutrient dosers.
Turn OnLight	LIGHT INTENSITY, LED	The lighting activation function depends on the LIGHT INTENSITY and LED classes. The LIGHT INTENSITY class is used to determine the ambient light level, and based on this measurement, the LED class is responsible for issuing the command to activate the lighting system.
Fan control	DHT, FANS	The fan control function depends on the DHT and FANS classes. Temperature measurements obtained through the DHT class are used to determine whether the fans should be activated or deactivated via the FANS class.

Dependent variables are:

1. Total number of classes and functionalities running on the system: This variable represents the number of functions that are active and operational on IoT devices after they have been self-assembled and self-replicated.
2. Successful Replications and Assemblies: This variable refers to the total number of successful self-replication and self-assembly operations completed.
3. Components reused: This variable denotes the total number of elements reused in the self-assembly and self-replication operations.
4. Reuse rate: The proportion of reused components in relation to total number of functionalities.
5. System response rate based on capacities: The average time that the system requires to respond to coordinator failure, restarts, and input of new source code, based on the capacities of the devices.

These variables provide a quantitative foundation for evaluating the self-assembly, self-replication, and autonomic ++ computing processes regarding functionality and efficiency. This contributes to a comprehensive validation of the proposed architecture.

6.5 Experimental Process, Scenario Setup and Results

The experimental process focuses, among other aspects, on the interaction between the IoT Coordinator Device of System 1 and an IoT updating source code device from system 2. During this interaction, autonomic self-replication and self-assembly are performed. The devices engage in the exchange of active and passive functional classes, as well as active and passive functions that are executed as threads. The original functional classes and functions executed as threads of each device are presented in Tables 2 and 3.

Table 2. Original systems classes and capacities

Classes (Software)	Capacity (Hardware)	Classes of the devices in system 1	Capacities of the devices in system 1	Classes of the devices in system 2	Capacities of the devices in system 2
DHT	dht		✓	✓	✓
LIGHT INTENSITY	ldr	✓	✓	✓	✓
PH	ph	✓	✓	✓	✓
EC	ec	✓	✓	✓	✓
WLS	wls	✓	✓		✓
WP	wp	✓	✓		
SV	sv	✓	✓	✓	✓
ND	nd	✓	✓		✓
LED	led		✓	✓	✓
FANS	fans			✓	✓
DHT-F	dht-F	✓			

Table 3. Original systems active functions execute as threads

Function	Functions executed as threads of the devices in system 1	Functions executed as threads of the devices in system 2
Measure temperature		✓
Measure Humidity		✓
Water Pump	✓	
Water level	✓	
Measure light intensity	✓	✓
Ph	✓	✓
Nutrients	✓	
Turn onLight		✓
Fan Control		✓
Temperature-F		

The devices of System 1 receive the new source code file from their IoT coordinator device. This file contains the new added active functional classes named DHT and LED, the functions executed as threads Measure temperature and Turn onLigth, the passive functional classes FANS, and the passive function Fan Control.

This is in accordance with the process delineated in Figure 10 and the result is reflected in Tables 4 and 5.

To evaluate the impact of network conditions on the self-assembly and self-replication process, the *elapsed time* was measured from the moment the IoT device originating from system 2 accessed the network of System 1. This interval corresponds to the time interval between the transmission of the ”coordinator search” message and the subsequent transmission of the ”reboot” message by the IoT Coordinator Device of System 1. The sequence of steps that takes place between the transmission of these two messages unfolds as described in the figure 8.

The interaction between the IoT Coordinator Device of System 1 and the IoT updating source code device of System 2 is studied in three different scenarios representing distinct network conditions. These scenarios are: (i) a network without degradation, (ii) 5% packet loss with a 100-millisecond delay, and (iii) 10% packet loss with a 200-millisecond delay. The generation of the different network conditions is facilitated by the Linux Traffic Control (TC) tool. Each network scenario is evaluated with 2, 5, 10, 20, and 50 IoT devices in System 1 (Docker containers). For each evaluation, the experiment is repeated ten times.

The average *elapsed time* results in seconds are shown in Table 6. A notable observation is that the mean *elapsed time* is comparable across the three distinct network conditions in each evaluation, irrespective of the number of devices. This can be attributed to the fact the messages used by IoT devices are usually very small and are not substantially influenced by alterations in the quality of service of the network. Furthermore, the increase in time needed for the source code self-assembly and self-replication process is approximately 3 seconds per device, regardless of network conditions.

Table 4. Classes and capacities in devices of system 1 after self-assembly of source code

Classes (Software)	Capacity (Hardware)	Classes of the devices in system 1	Capacities of the devices in system 1
DHT	dht	✓	✓
LIGHT INTENSITY	ldr	✓	✓
PH	ph	✓	✓
EC	ec	✓	✓
WLS	wls	✓	✓
WP	wp	✓	✓
SV	sv	✓	✓
ND	nd	✓	✓
LED	led	✓	✓
FANS	fans	✓	
DHT-F	dht-F	✓	

These 3 seconds can be explained by the implementation approach; a 1-second waiting period is defined to allow the coordinator (socket) to complete the receiving process upon receiving the new source code file, and another 1 second on the IoT devices when the coordinator notifies them that the newly generated source code file will be replicated. The remainder is associated with processing times.

It is important to emphasize that the presented implementation constitutes a proof of concept aimed at demonstrating the feasibility of architectural self-assembly and self-replication. The measured execution times are therefore specific to the prototype and do not represent optimized performance benchmarks. Alternative implementations and communication strategies may significantly reduce processing times. Performance optimization, however, is beyond the scope of this study, whose primary objective is to establish the conceptual and architectural viability of the proposed self-* properties.

6.6 Discussion of Results

The experimental evaluation demonstrated the feasibility at the software implementation level of self-replication and self-assembly between devices belonging to systems deployed under the same architecture and application domain. The results confirm that functional and self-management capacities can be autonomously replicated and assembled across devices without human intervention, validating the proposed Extended Autonomic IoT architecture at the software implementation level.

Table 5. Systems active Functions execute as threads in the new source code version

Functions executed as threads	Functions executed as threads of the devices in system 1
Measure temperature	✓
Measure Humidity	✓
Water Pump	✓
Water level	✓
Measure light intensity	✓
Ph	✓
Nutrients	✓
Turn onLight	✓
Fan Control	
Temperature-F	

Table 6. Average *elapsed time* results in seconds for each evaluation.

Devices	Without degradation	5% loss	10% loss
2	6.0130	6.0110	6.0115
5	18.0229	18.0211	18.0228
10	33.0351	33.0345	33.0357
20	57.3608	63.0701	63.0641
50	153.1422	153.1556	153.1427

The limited impact of adverse network conditions on the measured self-assembly and self-replication elapsed time observed during the experiments can be reasonably explained by the communication pattern characteristics involved in the self-assembly and self-replication processes. Message exchanges are triggered only during specific events, such as system updates or the incorporation of a new device into the community, which are infrequent in the type of controlled IoT environments considered in this study. In the presented experimental setup, IoT devices execute relatively simple algorithms and functions, resulting in small source code fragments and minimal communication overhead. Consequently, variations in network latency and packet loss do not significantly affect the overall behavior of the proposed mechanisms. The observed increase in self-assembly and self-replication process time, approximately three seconds per additional device on average, remains consistent across different network conditions.

Although the proposed approach is described in terms of architectural self-replication, the experimental instantiation focuses on software-level elements. In this work, classes and functions executed as threads are treated as the minimal executable architectural units within the scope of this implementation that can be autonomously replicated and assembled without external human intervention, making them suitable for validating architectural self-replication.

While the experiment instantiates only these two elements, the proposed model is not restricted to these constructs and can be extended to support higher-level artifacts, such as services, components, or containers, in future implementations.

7 Conclusion

In this work, an autonomic IoT architecture that incorporates the novel properties of self-assembly and self-replication was proposed. These formalized properties enable the composition of new versions or updates to the source code of IoT systems developed under the proposed architecture that belong to the same application domain. This is an extension and contribution to the available self-management capacities within the Autonomic Systems Theory.

In this sense, the experiment demonstrated the feasibility of reducing or eliminating human intervention while improving the control and elasticity of the deployed IoT systems. Consequently, this would increase their scalability and extend their lifecycle.

In light of the experimental work conducted and the novel capacities implemented within the proposed conceptual framework, potential security vulnerabilities were identified in the event of an attack on any of the IoT system's devices, resulting in infection with malicious code capable of self-assembly and self-replication. It is important to acknowledge that this aspect exceeds the objectives proposed in this paper and will be addressed in future work. T

he processes of self-replication and self-assembly necessitate a fundamental comprehension and acknowledgment of the capacities inherent in an IoT device, as well as the functionalities or services embedded within its code.

References

1. **4.0, P. I. (2014).** Plattform industrie 4.0: Rami4.0 - a reference framework for digitalisation.. Plattform Industrie 4.0, pp. 32.
2. **Abreu, D. P., Velasquez, K., Curado, M., Monteiro, E. (2017).** A resilient internet of things architecture for smart cities. Annals of Telecommunications, Vol. 72, pp. 19–30.
3. **Arzo, S. T., Bassoli, R., Granelli, F., Fitzek, F. H. (2021).** Multi-agent based autonomic network management architecture. IEEE Transactions on Network and Service Management, Vol. 18, No. 3, pp. 3595–3618.
4. **Arzo, S. T., Naiga, C., Granelli, F., Bassoli, R., Devetsikiotis, M., Fitzek, F. H. (2021).** A theoretical discussion and survey of network automation for iot: Challenges and opportunity. IEEE Internet of Things Journal, Vol. 8, No. 15, pp. 12021–12045.
5. **Ashraf, Q. M., Habaebi, M. H., Sinniah, G. R., Ahmed, M. M., Khan, S., Hameed, S. (2014).** Autonomic protocol and architecture for devices in internet of things. 2014 IEEE Innovative Smart Grid Technologies - Asia (ISGT ASIA), pp. 737–742. DOI: 10.1109/ISGT-Asia.2014.6873884.
6. **Ashraf, Q. M., Tahir, M., Habaebi, M. H., Isoaho, J. (2023).** Toward autonomic internet of things: Recent advances, evaluation criteria, and future research directions. IEEE Internet of Things Journal, Vol. 10, No. 16, pp. 14725–14748.
7. **Bahga, A., Madisetti, V. (2014).** Internet of Things. A Hands-On Approach.
8. **Barriga, J. K. D., Romero, C. D. G., Molano, J. I. R. (2016).** Proposal of a standard architecture of iot for smart cities. Learning Technology for Education in Cloud–The Changing Face of Education: 5th International Workshop, LTEC 2016, Hagen, Germany, July 25-28, 2016, Proceedings 5, Springer, pp. 77–89.
9. **Berns, A., Ghosh, S. (2009).** Dissecting self-* properties. 2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems, pp. 10–19. DOI: 10.1109/SASO.2009.25.
10. **Bezerra, R. M., Nascimento, F. M., Martins, J. S. (2015).** On computational infraestruture requirements to smart and autonomic cities framework. 2015 IEEE First International Smart Cities Conference (ISC2), IEEE, pp. 1–6.

11. **Calderoni, L., Magnani, A., Maio, D. (2019).** Iot manager: An open-source iot framework for smart cities. Journal of Systems Architecture, Vol. 98, pp. 413–423.
12. **Camara, J., Papadopoulos, A. V., Vogel, T., Weyns, D., Garlan, D., Huang, S., Tei, K. (2020).** Towards bridging the gap between control and self-adaptive system properties. arXiv preprint arXiv:2004.11846.
13. **Caraguay, A. L. V., Gonzalez, P. L., Tandazo, R. T., Lopez, L. I. B. (2018).** Sdn/nfv architecture for iot networks. WEBIST, pp. 425–429.
14. **Chattopadhyay, S., Banerjee, A. (2020).** Qos-aware automatic web service composition with multiple objectives. ACM Transactions on the Web (TWEB), Vol. 14, No. 3, pp. 1–38.
15. **Chatzigiannakis, I., Hasemann, H., Karnstedt, M., Kleine, O., Kröller, A., Leggieri, M., Pfisterer, D., Römer, K., Truong, C. (2012).** True self-configuration for the iot. 2012 3rd IEEE International Conference on the Internet of Things, IEEE, pp. 9–15.
16. **Chehida., S., Bensalem., S., Conzon., D., Ferrera., E., Tao., X. (2022).** Brain-iot architecture and platform for building iot systems. Proceedings of the 7th International Conference on Internet of Things, Big Data and Security - IoTBDS,, INSTICC, SciTePress, pp. 67–77. DOI: 10.5220/0011086000003194.
17. **da Silva, W. M., Alvaro, A., Tomas, G. H. R. P., Afonso, R. A., Dias, K. L., Garcia, V. C. (2013).** Smart cities software architectures: A survey. Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC 13, ACM Press, pp. 1722–1727. DOI: 10.1145/2480362.2480688.
18. **de Ribamar Lima Ribeiro, A., de Almeida, F. M., Moreno, E. D., Montesco, C. A. (2016).** A management architectural pattern for adaptation system in internet of things. 2016 International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 576–581. DOI: 10.1109/IWCMC.2016.7577121.
19. **Dehraj, P., Sharma, A. (2021).** A review on architecture and models for autonomic software systems. The Journal of Supercomputing, Vol. 77, No. 1, pp. 388–417.
20. **Elkholy, M., Baghdadi, Y., Marzouk, M. (2022).** Snowball framework for web service composition in soa applications. International Journal of Advanced Computer Science and Applications, Vol. 13, No. 1.
21. **Ellery, A. (2017).** Bioinspiration lessons from a self-replicating machine concept in a constrained environment. 2017 IEEE International Conference on Robotics and Biomimetics (ROBIO), pp. 1874–1879. DOI: 10.1109/ROBIO.2017.8324692.
22. **ETSI, N. (2015).** Report on sdn usage in nfv architectural framework. ETSI GS NFV-EVE, Vol. 5, pp. v1.
23. **Fortino, G., Russo, W., Savaglio, C., Shen, W., Zhou, M. (2017).** Agent-oriented cooperative smart objects: From iot system design to implementation. IEEE Transactions on Systems, Man, and Cybernetics: Systems, Vol. 48, No. 11, pp. 1939–1956.
24. **Gavrilović, N., Mishra, A. (2021).** Software architecture of the internet of things (iot) for smart city, healthcare and agriculture: analysis and improvement directions. Journal of Ambient Intelligence and Humanized Computing, Vol. 12, No. 1, pp. 1315–1336.
25. **Gavvala, S. K., Jatoth, C., Gangadharan, G., Buyya, R. (2019).** Qos-aware cloud service composition using eagle strategy. Future Generation Computer Systems, Vol. 90, pp. 273–290.
26. **Giordano, A., Spezzano, G., Vinci, A. (2016).** Smart agents and fog computing for smart city applications. **Alba, E., Chicano, F., Luque, G.**, editors, Smart Cities, Springer International Publishing, Cham, pp. 137–146.
27. **Group, I. C. S. (2002).** The tivoli software implementation of autonomic computing guidelines.
28. **Herrera, V. V., Bepperling, A., Lobov, A., Smit, H., Colombo, A., Lastra, J. M. (2008).** Integration of multi-agent systems and service-oriented architecture for industrial automation. 2008 6th IEEE International Conference on Industrial Informatics, IEEE, pp. 768–773.
29. **Hill, P., Gallagher, J. (1998).** Meta-programming in logic programming. Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 5, pp. 421–497.
30. **Hofstader, D. (1979).** Godel, escher, bach: an eternal golden braid.
31. **Horn, P. (2001).** Autonomic computing: Ibm's perspective on the state of information technology. IBM.
32. **Hu, P., Chen, W. (2019).** Software-defined edge computing (sdec): Principles, open system architecture and challenges. 2019

IEEE SmartWorld, Ubiquitous Intelligence y Computing, Advanced y Trusted Computing, Scalable Computing y Communications, Cloud y Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), pp. 8–16. DOI: 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00047.

33. **ISO/IEC (2014).** Iso/iec jtc 1 smart cities. Technical report, ISO/IEC.
34. **ISO/IEC/IEEE (2011).** Iso/iec/ieee systems and software engineering – architecture description. ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000), pp. 1–46. DOI: 10.1109/IEEESTD.2011.6129467.
35. **Kang, B., Kim, D., Choo, H. (2017).** Internet of everything: A large-scale autonomic iot gateway. IEEE Transactions on Multi-Scale Computing Systems, Vol. 3, No. 3, pp. 206–214.
36. **Kephart, J. O., Chess, D. M. (2003).** The vision of autonomic computing. Computer, Vol. 36, No. 1, pp. 41–50.
37. **Krivic, P., Skocir, P., Kusek, M., Jezic, G. (2017).** Microservices as agents in iot systems. Agent and Multi-Agent Systems: Technology and Applications: 11th KES International Conference, KES-AMSTA 2017 Vilamoura, Algarve, Portugal, June 2017 Proceedings 11, Springer, pp. 22–31.
38. **Kruchten, P. B. (1995).** The 4+1 View model of architecture. IEEE Software, Vol. 12, No. 6, pp. 42–50. DOI: 10.1109/52.469759.
39. **Lemoine, F., Aubonnet, T., Simoni, N. (2020).** Self-assemble-featured internet of things. Future Generation Computer Systems, Vol. 112, pp. 41–57. DOI: https://doi.org/10.1016/j.future.2020.05.012.
40. **Lin, P., MacArthur, A., Leaney, J. (2005).** Defining autonomic computing: a software engineering perspective. 2005 Australian Software Engineering Conference, IEEE, pp. 88–97.
41. **Marques, P., Manfroi, D., Deitos, E., Cegoni, J., Castilhos, R., Rochol, J., Pignaton, E., Kunst, R. (2019).** An iot-based smart cities infrastructure architecture applied to a waste management scenario. Ad Hoc Networks, Vol. 87, pp. 200–208.
42. **Matsumoto, M., Hashimoto, S. (2007).** A design of self reproducing hardware. 2007 IEEE International Symposium on Assembly and Manufacturing, IEEE, pp. 259–263.
43. **Matsumoto, M., Hashimoto, S. (2009).** Passive self-replication of millimeter-scale parts. IEEE transactions on automation science and engineering, Vol. 6, No. 2, pp. 385–391.
44. **McCann, J. A., Huebscher, M. C. (2004).** Evaluation issues in autonomic computing. Grid and Cooperative Computing-GCC 2004 Workshops: GCC 2004 International Workshops, IGKG, SGT, GISS, AAC-GEVO, and VVS, Wuhan, China, October 21-24, 2004. Proceedings 3, Springer, pp. 597–608.
45. **Mendoncca, N. C., Garlan, D., Schmerl, B., Camara, J. (2018).** Generality vs. reusability in architecture-based self-adaptation: The case for self-adaptive microservices. Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, Association for Computing Machinery, New York, NY, USA, pp. 6. DOI: 10.1145/3241403.3241423.
46. **Meselson, M., Stahl, F. W. (1958).** The replication of dna in escherichia coli. Proceedings of the national academy of sciences, Vol. 44, No. 7, pp. 671–682.
47. **Mohalik, S. K., Narendra, N. C., Badrinath, R., Le, D.-H. (2017).** Adaptive service-oriented architectures for cyber physical systems. 2017 IEEE Symposium on Service-Oriented System Engineering (SOSE), pp. 57–62. DOI: 10.1109/SOSE.2017.10.
48. **Movahedi, Z., Ayari, M., Langar, R., Pujolle, G. (2011).** A survey of autonomic network architectures and evaluation criteria. IEEE Communications Surveys & Tutorials, Vol. 14, No. 2, pp. 464–490.
49. **Nami, M. R., Bertels, K. (2007).** A survey of autonomic computing systems. Third international conference on autonomic and autonomous systems (ICAS/'07), IEEE, pp. 26–26.
50. **Neumann, J. v., Burks, A. W., et al. (1966).** Theory of self-reproducing automata, Vol. 1102024. University of Illinois press.
51. **Ochoa-Aday, L., Cervello-Pastor, C., Fernandez-Fernandez, A. (2020).** Self-healing and sdn: bridging the gap. Digital Communications and Networks, Vol. 6, No. 3, pp. 354–368.
52. **of ITU, T. S. S. (2014).** Recommendation itu-t y.3300. ITU-T Y.3300.
53. **Omarov, B., Altayeva, A., Turganbayeva, A., Abdulkarimova, G., Gusmanova, F., Sarbasova, A., Omarov, B., Dauletbek, Y., Altayeva, A., Omarov,**

N. (2019). Agent based modeling of smart grids in smart cities. Electronic Governance and Open Society: Challenges in Eurasia: 5th International Conference, EGOSE 2018, St. Petersburg, Russia, November 14-16, 2018, Revised Selected Papers 5, Springer, pp. 3–13.

54. **Patra, M. K. (2017).** An architecture model for smart city using cognitive internet of things (ciot). 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT), IEEE, pp. 1–6.
55. **Plasson, R. (2011).** Self Replication, chapter 1. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1498–1500.
56. **PPP, G. (2019).** 5G PPP 5G Architecture White Paper Revision 3.0. 5G Infrastructure Public Private Partnership.
57. **Ramirez-Tovar, D.-E. (2024).** An iot autonomic architecture for self-replicating and self-assembly systems. `https://github.com/CinvestavGDL-NS/Self-StarPlusPlus`. Python Script, version 1.008.
58. **Rolim, C. O., Rossetto, A. G., Leithardt, V. R., Borges, G. A., Dos Santos, T. F., Souza, A. M., Geyer, C. F. (2015).** An ubiquitous service-oriented architecture for urban sensing. Agent Technology for Intelligent Mobile Services and Smart Societies: Workshop on Collaborative Agents, Research and Development, CARE 2014, and Workshop on Agents, Virtual Societies and Analytics, AVSA 2014, Held as Part of AAMAS 2014, Paris, France, May 5-9, 2014. Revised Selected Papers, Springer, pp. 1–10.
59. **Roscia, M., Longo, M., Lazaroiu, G. C. (2013).** Smart city by multi-agent systems. 2013 International Conference on Renewable Energy Research and Applications (ICRERA), IEEE, pp. 371–376.
60. **Rubenstein, M., Krivokon, M., Shen, W.-M. (2004).** Robotic enzyme-based autonomous self-replication. 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566), IEEE, Vol. 3, pp. 2661–2666.
61. **Santana, E. F. Z., Chaves, A. P., Gerosa, M. A., Kon, F., Milojicic, D. S. (2017).** Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture. ACM Computing Surveys (Csur), Vol. 50, No. 6, pp. 1–37.
62. **Savaglio, C., Ganzha, M., Paprzycki, M., Bădică, C., Ivanović, M., Fortino, G. (2020).** Agent-based internet of things: State-of-the-art and research challenges. Future Generation Computer Systems, Vol. 102, pp. 1038–1053. DOI: `https://doi.org/10.1016/j.future.2019.09.016`.
63. **Sharma, D. P., Singh, B. K., Gure, A. T., Choudhury, T. (2021).** Autonomic computing: models, applications, and brokerage. Autonomic Computing in Cloud Resource Management in Industry 4.0, pp. 59–90.
64. **Shoham, Y., Leyton-Brown, K., et al. (2009).** Multiagent systems. Algorithmic, Game-Theoretic, and Logical Foundations.
65. **Sterritt, R., Hinchey, M. (2010).** Spaace iv: Self-properties for an autonomous & autonomic computing environment–part iv a newish hope. 2010 Seventh IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems, IEEE, pp. 119–125.
66. **Sun, Y., Song, H., Jara, A. J., Bie, R. (2016).** Internet of things and big data analytics for smart and connected communities. IEEE Access, Vol. 4, pp. 766–773. DOI: `10.1109/access.2016.2529723`.
67. **Tekinerdogan, B., Köksal, Ö., Çelik, T. (2023).** System architecture design of iot-based smart cities. Applied Sciences, Vol. 13, No. 7, pp. 4173.
68. **Villela Zavala, L. E., Ordoñez García, A., Siller, M. (2019).** Architecture and algorithm for iot autonomic network management. 2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 861–867. DOI: `10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00155`.
69. **W. Collier, R., O'Neill, E., Lillis, D., O'Hare, G. (2019).** Mams: Multi-agent microservices. Companion Proceedings of The 2019 World Wide Web Conference, pp. 655–662.
70. **Wang, S., Zhou, A., Yang, M., Sun, L., Hsu, C.-H., Yang, F. (2017).** Service composition in cyber-physical-social systems. IEEE Transactions on Emerging Topics in Computing, Vol. 8, No. 1, pp. 82–91.
71. **Weyns, D. (2018).** Engineering self-adaptive software systems – an organized tour. 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W), pp. 1–2. DOI: `10.1109/FAS-W.2018.00012`.

72. **Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., Wuttke, J., Andersson, J., Giese, H., Goschka, K. M. (2013).** On patterns for decentralized control in self-adaptive systems. In **de Lemos, R., Giese, H., Müller, H. A., Shaw, M.**, editors, Software Engineering for Self-Adaptive Systems II. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 76–107. DOI: 10.1007/978-3-642-35813-5_4.

73. **Wooldridge, M. (2009).** An introduction to multiagent systems. John Wiley y Sons.

74. **Zavala, L. E. V. (2018).** Arquitectura y Algoritmo de Gestión para Redes IoT Autonómicas. Master's thesis, CINVESTAV Guadalajara.

Article received on 07/04/2026; accepted on 14/03/2026.
**Corresponding author is Mario Siller.*