applied sciences

MDPI

Article

SCM-IoT: An Aproach for Internet of Things Services Integration and Coordination

Isaac Machorro-Cano [1], José Oscar Olmedo-Aguirre [2], Giner Alor-Hernández [3,*], Lisbeth Rodríguez-Mazahua [3], José Luis Sánchez-Cervantes [4] and Asdrúbal López-Chau [5]

1 Universidad del Papaloapan, Calle Circuito Central #200, Col. Parque Industrial, Tuxtepec C.P. 68301, Oaxaca, Mexico; imachorro@unpa.edu.mx
2 Department of Electrical Engineering, CINVESTAV-IPN, Av. Instituto Politécnico Nacional 2 508, Col. San Pedro Zacatenco, Delegación Gustavo A. Madero C.P 07360, Mexico City, Mexico; oolmedo@cinvestav.mx
3 Tecnológico Nacional de México/I. T. Orizaba, Av. Oriente 9 852, Col. Emiliano Zapata, Orizaba C.P. 94320, Veracruz, Mexico; lrodriguezm@ito-depi.edu.mx
4 CONACYT-Tecnológico Nacional de México/I. T. Orizaba, Av. Oriente 9 852, Col. Emiliano Zapata, Orizaba C.P. 94320, Veracruz, Mexico; jlsanchez@conacyt.mx
5 Universidad Autónoma del Estado de México, Centro Universitario UAEM Zumpango, Camino Viejo a Jilotzingo Continuación Calle Rayón, Valle Hermoso, Zumpango C.P. 55600, Estado de México, Mexico; alchau@uaemex.mx
* Correspondence: giner.ah@orizaba.tecnm.mx; Tel./Fax: +52-272-725-7056

Abstract: Today, new applications demand an internet of things (IoT) infrastructure with greater intelligence in our daily use devices. Among the salient features that characterize intelligent IoT systems are interoperability and dynamism. While service-oriented architectures (SOA) offer a well-developed and standardized architecture and protocols for interoperability, answering whether SOA offers enough dynamism to merge IoT with artificial intelligence (AI) is still in its beginnings. This paper proposes an SOA model, called SCM-IoT (service composition model for IoT), for incorporating AI into IoT systems, addressing their coordination by a mediator offering services for storage, production, discovery, and notification of relevant data for client applications. The model allows IoT systems to be incrementally developed from three perspectives: a conceptual model, platform-independent computational model, and platform-dependent computational model. Finally, as a case of study, a domotic IoT system application is developed in SCM-IoT to analyze the characteristics and benefits of the proposed approach.

Keywords: colored Petri nets; intelligent systems; internet of things; services interoperability; services composition

check for updates

Citation: Machorro-Cano, I.; Olmedo-Aguirre, J.O.; Alor-Hernández, G.; Rodríguez-Mazahua, L.; Sánchez-Cervantes, J.L.; López-Chau, A. SCM-IoT: An Aproach for Internet of Things Services Integration and Coordination. *Appl. Sci.* **2022**, *12*, 3133. https://doi.org/10.3390/app12063133

Academic Editors: Agostino Forestiero and Stefan Fischer

Received: 20 December 2021
Accepted: 14 March 2022
Published: 18 March 2022

1. Introduction

Technological advances have allowed complex closed systems to be redesigned on a more open basis of composing interoperable components. IoT exemplifies this trend motivated by the progressive decomposition of components, seeking to minimize their functionality while maximizing the system's dynamism, flexibility, and maintainability through service composition (SC) [1]. The concept of IoT originated after the rise of wireless gadgets and devices, and as a result, multiple IoT technologies are employed nowadays. Some of these technologies include radio frequency identification devices (RFIDs), wireless communication, cloud applications for wireless sensor networks (WSNs), near-field communication (NFC), global positioning system (GPS), WiFi, and Bluetooth [2–4]. Indeed, the IoT is a captivating trend when it comes to integrating mobile technology to things or real-world devices, which now create new forms of communication, including machine to machine (M2M), peer to machine (P2M), and peer to peer (P2P) communication. These

new forms of communication have managed to close the gap between communication and information [5].

Similarly, IoT has real-world characteristics, yet it also experiences processing and storage limitations that imply new trends and challenges for improving privacy, reliability, security, and performance [6]. In addition, the IoT is a clear representation of the growth of the Internet, where different devices are interconnected with people. For instance, several microcontrollers (e.g., health monitors, wearables, sensors, and actuators) are developed each year to interconnect with other things or smart devices through a specific network. In this sense, smart devices have remarkable detection, processing, and networking capabilities, and they are used in various IoT application domains, such as healthcare and domotic [7,8].

Likewise, web protocols guarantee the communication of smart devices, which are compatible with constraint application protocols (CoAPs) and with other smart devices. In addition, there is interaction with various web services, and the services offered by smart devices are consumed through ubiquitous applications [9]; however, performing the SC of the smart devices is complicated. SC is considered a fundamental aspect of service-oriented computing (SOC), where various services collaborate to achieve a common goal [10]. Services orchestration and choreography are two relevant processes interaction patterns for the SC [11]. The centralized process for coordinating interactions between the services of a business process or activity is known as service orchestration. The participants in a service orchestration process are not regularly known, similar to what happens in different scenarios in the IoT [12].

Services choreography describes the collaboration among the participant process, particularly in their order of participation. Participants in a service choreography process collaborate knowing the specific form of participation [13]. Besides, BPEL and WS-CDL are two compositional languages used to orchestrate and choreograph services. Additionally, in the context of business processes and web services, the application of SC in real-life cases is usually investigated more frequently. However, it is not possible to apply the procedures applied on the web services in the context of the IoT due to the limitations of the devices and scenarios particularities presented in the different IoT application domains. For this reason, advanced mechanisms are required to facilitate the integration and collaboration of heterogeneous smart devices, the workflows representation, and the IoT based services coordination, where the services are used through the services orchestration and choreography in different real-life scenarios to meet the diverse needs of users, to take advantage of and exploit the advantages that the IoT offers. Therefore, IoT provides new opportunities to create a more innovative world, which is evident when the number of data and events from various smart objects are dynamically composed in a simple way for new applications.

Applications in IoT arrival are because the protocols based on the Web facilitated the support in real-time. Likewise, to meet the diverse requirements of users in the IoT, the SC combines services of various intelligent devices, which allows the development of new applications [14]. In addition, healthcare and domotic in the context of the IoT are two relevant application areas in which there is an opportunity to improve services interoperability and reduce some costs. In this sense, the services integration and coordination in the healthcare and domotic are required to cover the different needs of patients who need to continue their recovery or rehabilitation at home [15]. Healthcare and domotic share various integration and coordination aspects in their processes such as information flow, organizational processes, market development, integration of inter and intra-organizational processes, and market approaches. For health and domotic services management, coordinated integration of processes is necessary due to the relationship between patients and providers of the smart wearables used to monitor their health, increase comfort, and save energy at home [16,17].

Current SC challenges on intelligent systems are still largely unexplored for IoT services interoperability. This work proposes an IoT services composition mechanism

called SCM-IoT, which performs the IoT services integration and coordination. BPEL is important because it addresses IoT services orchestration and choreography, though the latter to a lesser extent. SCM-IoT offers three services in BPEL (selection, update, and notification) to provide an infrastructure for systems exhibiting greater intelligent behavior. Finally, to validate SCM-IoT, a case study is developed to automate domotic appliances for a physically disabled and overweight patient who needs to continue his recovery and rehabilitation at home. This work extends [18] that PISIoT presents, a platform that contributes to the control of overweight or obesity, and [19] that HEMS-IoT presents, a platform that provides recommendations for energy-saving and comfort for residents at home.

SCM-IoT is proposed to develop IoT applications in a broker-oriented architecture providing storage, edition, and notification services upon the message-passing distributed architecture of BPEL. Consequently, SCM-IoT makes it possible to extend and simplify the development of applications often found in artificial intelligence. These applications involve the collaboration of intelligent agents exchanging information through shared memory. Among the various applications of SCM-IoT, the following have been already addressed:

- Home automation, for the assistance of people with disabilities through automated control of home appliances;
- Healthcare, for reducing overweight and obesity;
- Systems integration, for monitoring the programmed activities needed in treatments and therapies of patients with physical disabilities.

SCM-IoT also extends the BPEL markup language with new elements further translated into BPEL elements only. This approach allows for orchestrating the activities of agents willing to be notified on any relevant application data. Besides, SCM-IoT closely resembles a data-centric model, similar to the distributed database model. Data may be located in physically distributed storage media, though interconnected by a network infrastructure that ensures the retrieval and exchange of information between applications.

Among the original contributions brought by the SCM-IoT, we can outline the following: (1) The SCM-IoT model introduces a coordination mechanism where storage, production, and notification of findings of data of interest by a mediator, called the editor, are performed on the original data provided by content producers. (2) SCM-IoT is a table-oriented coordination language instead of other tuple-oriented models such as Linda coordination model [20,21]. (3) Unlike the Linda model, the primitive operations of SCM-IoT are not based on the availability of tuples but based on the satisfaction of logical conditions. (4) In SCM-IoT, the small number of changes in the number of sensors and actuators in many IoT applications, such as home automation and healthcare, lead to a fixed table structure. However, the static nature of the table is not a conceptual limitation of the model but a non-functional constraint that ensures greater efficiency during coordination. A design based on a fixed-structure table may additionally give applications greater clarity by allowing all logically related entries (i.e., those that follow the same editing rules) to be grouped under the same table.

This paper is structured as follows: Section 2 discusses works on SC, services coordination, services orchestration, and services choreography in the IoT context. In addition, the abstract models of distributed systems, colored Petri nets, and event-condition-action (ECA) rules are introduced. Next, in Section 3, we describe the methodology used. In Section 4, a case study is presented to describe the results obtained. In addition, in Section 5, we discuss the lack of dynamism and limitations. Finally, in Section 6, some conclusions and suggestions for future research are given.

2. Related Work

The interconnection in the IoT is due to the wide adoption and application development in application domains. These devices collaborate and cooperate with other smart devices to integrate and combine different functionalities to achieve a common goal [22]. In this section, we review state-of-the-art IoT initiatives for SC.

2.1. IoT Services Composition and Coordination

In artificial potential fields (APFs), Rapti et al. [23] provided a decentralized SC model for IoT environments. Pang et al. [24] described an enterprise technology co-design methodology for an in-home healthcare station (IHHS). In addition, they used design principles of an IHHS solution, including 3C platform reuse and efficient SC, among other aspects. Swiatek [25] introduced the ComSS middleware to operate and manage IoT compositional service flow. The performance of ComSS depends on several factors: the number of available services, inputs, data, and flow formats used. In contrast, Dijkman et al. [26] proposed a scheme for developing business models in IoT applications based on literature, interviews, and a survey among IoT practitioners. Furthermore, Pisching et al. [10] presented a study on SC 4.0 cloud-centric manufacturing, as, in this environment, all objects, features, and resources representing their states, information, and mode of operation are considered services. Services, such as description, location, publication, and invocation in a network, respond to requests between consumers and service providers. On the other hand, Shehu et al. [27] provided two evolutionary algorithms (VPSO and NGA) for QoS in IoT, taking into account the network. For QoS, the algorithms search for the composition of services with optimal response time, network latency, cost, and reputation. Vidyasankar [28] proposed a correctness rule for QoS activities and a transaction model in the context of IoT. The atomicity and isolation of information exchange are flexibly defined with his proposal to cope with the diversity of applications. From the perspective of the energy optimization mechanisms in IoT, Sun et al. [22] optimized simultaneous requests in SC. Their proposal optimizes energy consumption and facilitates the services exchange between simultaneous requests in the context of the IoT.

Furthermore, Gierej et al. [29] developed a business model devoted to companies implementing industrial internet of things technologies. The proposed concept was developed to support traditional companies transitioning to the digital market. Moreover, Ju et al. [30] provided a generic business model for IoT services based on a literature review and interviews with eight IoT experts. Urbieta et al. [31] reported an adaptive service composition framework that supports dynamic reasoning based on wEASEL, and an abstracted service model representing services and user tasks in terms of their signature, specification, and conversation. Salle et al. [32] introduced a preliminary inspired biological modeling approach to exploit the peculiarities of the immune system in a way that enables dynamic and reliable software and service composition in IoT.

In contrast, Baker et al. [33] elaborated E2C2, a multi-cloud IoT SC algorithm that enables the deployment of an energy-aware composition scheme through the integration and retrieval of constrained services in the IoT. Yamaoka et al. [34] introduced Dracena, a data processing platform that supports easy integration of real-time IoT services. The performance of Dracena was assessed in handling many moving vehicles, where concurrently generated data was processed in real-time, while multiple services used the yielded data. Krishna et al. [35] developed IoT composer, a tool to support the development of IoT applications by providing a model of object behavior and composition. They also developed an implementation to effectively link and instantiate all the objects involved in the SC. In turn, Ridhawi et al. [36] presented a solution for decentralized SC based on blockchain to provide multimedia services for user subscription. For this purpose, they dynamically created user-defined services without requiring intermediary services.

Considering spatial and temporal aspects, Lakhdari et al. [37] proposed a new energy SC model to use compositional patterns of devices in IoT. They developed a heuristic-based composition approach with multiple local knapsacks to select an optimal set of energy-based services that devices in the IoT can deliver. Arellanes and Lau [38] analyzed the fundamentals of service composition mechanisms in the IoT to determine scalability requirements. Likewise, they identified that data flows, orchestration, and choreography do not fully satisfy such requirements, unlike the novel composition mechanism DX-MAN. Likewise, Abusafia et al. [39] presented an incentive-based framework for energy service request composition. In addition, they designed an incentive model that considers

the context of providers and consumers to determine the rewards generated by wireless energy sharing.

Concerning service coordination, in their work, Cano et al. [40] presented a case study for the design of secure IoT applications to relate the semantics of event-condition-action (ECA) rules based on automata used for verification of rule compilation at run-time. Furthermore, Giang et al. [41] presented IoT applications in smart cities from a distributed coordination model supervised by components. Cheng et al. [42] proposed a service platform for the IoT based on SOA to design services oriented to events. Besides, an event definition language (SEDL) was presented, based on automata, event detection algorithms, and a situational event driven by the service coordination behavior model. Belkeziz and Jarir [43] presented a study describing the behavior of the layers of flexible and multiple architectures for efficient IoT services coordination. Instead, they used the orchestration or choreography of services to achieve their coordination. However, this design decision posed considerable complexity since the coordination of services represented several challenges such as heterogeneity, accessibility, context awareness, and discovery.

Moreover, García–Magariño et al. [44] proposed an agent-based approach to support large-scale IoT by providing complex integrated services. They avoided the common bottlenecks of this approach and the excessive bandwidth use due to direct communications in peer-to-peer (P2P) networks. Likewise, Belkeziz and Jarir [45] described a recent review of existing service coordination approaches in the IoT where they identified a tendency to use orchestration or choreography to meet this challenge. They presented a classification and overview of the leading service coordination models in the IoT. They designed an architecture in the context of the IoT that can achieve service coordination through the concept of meta-workflows, combining both coordination approaches: orchestration and choreography.

2.2. IoT Services Orchestration and Choreography

Regarding service orchestration, Cubo et al. [46] introduced the DEEP platform to support integrating and orchestrating heterogeneous devices that deliver services, storage, and access through the cloud. Alternatively, Qu et al. [47] reported a model for the specification of dynamic services for entities in the IoT. In this model, the entity state information was broadcast in real-time by the extended structure and released to requesters as dynamic services. Furthermore, Yu et al. [48] provided a platform adapted for the convergence of IoT and WoT, which was crucial for implementing smart grids by fusing dynamic elements with no user intervention. Bergesio et al. [49] suggested an object-oriented model capable of orchestrating services and using an autonomous system to help users personalize smart spaces. Likewise, Wen et al. [50] described a fog orchestrator to centralize a collection of resources, map applications to specific requests, deliver automated workflow to physical resources, build workload execution with quality of service control at runtime, and create time-efficient policies to handle objects. In addition, Macker and Taylor [12] further suggested the network edge workflow tool (Newt) for two use cases. Additionally, they provided a set of workflow requirements for MANET group applications to support decentralized decision making, group-based communication, and multiple transport media widely.

In addition, Ren et al. [51] introduced a service selection model that emphasizes the global synergy effect based on collaboration requirements. Pahl et al. [52] reported an architectural pattern with its underlying principles. The pattern combines IoT edge orchestration with a provenance mechanism, which relies on the blockchain for trusted orchestration management (TOM) in the cloud. A novel IoT service orchestration approach was proposed by Malik et al. [53] on multiple sensor and actuator platforms using virtual objects in an IoT application store. Likewise, Ren et al. [54] presented a new orchestration scheme to solve the optimization problem when implementing the service function chains concept in the IoT. In addition, they modified the traditional SFC definition language to describe the composite definitions in the IoT. In addition, Rafique et al. [55] proposed an IoT

application development framework called IADev to address the challenges arising from the lack of tools and techniques for IoT systems using attribute-based design and model-driven development (MDD). It was identified that IADev achieved higher participant satisfaction in service orchestration and application development in the IoT compared to conventional approaches. Likewise, Serhani et al. [56] proposed a comprehensive architecture for end-to-end workflow management processes, in which they included declarative specification and composition, as well as orchestration, adaptation, among other aspects. In addition, through supervision and automated analysis of cloud resource orchestration at run-time, they achieved orchestration of a workflow in the context of IoT.

In their work on service choreography, Rodriguez–Valenzuela et al. [57] described a novel method to implement a merged acquisition of distributed data using a lightweight SC model, guaranteeing the accuracy of collaborations with no cyclic behavior with the data in a distributed and decentralized way. Dar et al. [58] analyzed the design and implementation of ROA, a generic architecture model with tools for integrating end-to-end systems and IoT-based business processes. On the other hand, Duhart et al. [59] introduced the environment monitoring and management agent (EMMA) framework, based on a suite of elements to design distributed architectures for distributed environments. Cherrier et al. [60] showed a complete architecture for designing applications in the IoT and proposed D-LIT, providing universal access to the internal processing of objects and computing power. Blanc et al. [61] suggested process choreography, a paradigm based on SOA combined with wireless sensor and actuator networks (WSANs) to virtualize WSANs and use them in IoT. Besides, Chen and Englund [13] argued a choreography platform for internet-oriented services, performing the choreography of heterogeneous services through an automatic synthesis of choreography diagrams.

Montali and Plebani [62] approached IoT to foster business processes in various application domains (logistics, manufacturing, and healthcare). Seeger et al. [63] provided a design for managing dynamic choreographies in IoT to expand the evaluation and implementation facets to maintain the functionality of building automation systems so that new devices involved in the choreography either actively participate or become inactive again. Singhal et al. [64] studied the selection mechanism in BPEL, implementing a dynamic SC approach through orchestration and choreography in health applications. Their analysis based on time, memory, and energy consumption identified that choreography is faster than orchestration. In the context of the IoT, Arreaga et al. [65] presented a monitoring control and management system to achieve the interoperability of multiple smart gateways in a greenhouse. They also measured soil and environmental parameters to help in the farm supervision and management of the greenhouse on the Chlorophyll_X Web platform. Automatic decision-making was promoted in real-time on the water supply for crops through a network of wireless sensors with several intelligent gateways. The following section describes the expressiveness, design, formalization, and validation of SCM-IoT.

2.3. Abstract Models of Distributed Systems

From the perspective of formal modeling of IoT systems, at least three inherent properties that characterize IoT systems can be recognized: concurrency, interaction, and reconfigurability [66,67].

Concurrency is a fundamental property of any distributed system [68,69]. A concurrent system is organized in computational subsystems or components that exhibit a relatively independent behavior from each other while developing their activities. Components possess a configuration determined by their internal state and their knowledge of the context they operate. From an engineering perspective, concurrency promotes efficiency in achieving several activities by reducing the time to complete all of them. However, this improvement introduces the need to coordinate precedence relationships between component activities by establishing those that must finish before starting others [70]. A formal visual representation of the concurrency of a system can be seen by a directed graph where component activities are represented by nodes and precedence relationships between

activities by the directed arcs that link them [67,71]. The interaction between components provides a flexible mechanism to coordinate these dependencies properly.

The interaction between the components of a distributed system lies in the elementary ability to perceive changes in their environment and perform actions accordingly. The environment establishes a computational context that may consist solely of components or include a communication medium. Direct interactions [72,73] between components are conducted through the exchange of messages, where the message sending component reveals its intention to deliver information using a signal that is perceived only by the receiving component, which then prepares to receive it. In representing a system as a directed network between activities, the direct interaction between components can be represented as a precedence relationship from the sending component to one or more receiving components. Indirect interactions between components are conducted through a communication medium that can be realized over a possibly shared, persistent, or transactional storage space as in the Linda coordination model [20,21]. Through this storage space, decoupled interactions between components are developed, i.e., where it is not necessary to know the identity between the participating components but to know the identity of the subspace where the information will be stored by the sender and eventually retrieved by the receivers. In the activity dependency graph, indirect interaction can be modeled by additional fictitious activities, although this approach makes this representation complex. A more suitable type of representation is Petri nets [16,74,75], where there is a clear distinction between two types of nodes, in activities (transitions) and partitioned storage space (places), as well as their dependency relationships that indicate the flow of information to or from the medium.

Reconfigurability refers to the ability of a highly dynamic distributed system to modify its configuration, including that of its components [67,76]. In modern internet-based distributed systems, it is increasingly common to change the computational locality of an agent, human or computational, represented by a component. Changing locality involves following connection and disconnection protocols between components to preserve communication dynamically [66]. In its representation as directed graphs, the reconfiguration of a distributed system can be seen as a change in the graph by deletion of an incident arc at one node followed by the creation of a new incident arc at another node but with the same origin node [77]. In IoT, the ability to reconfigure a system is crucial in the presence of frequent disconnections between components due to their unstable communication medium, intermittent operation, or physical substitution. Besides, reconfigurability enables extending the system's functionality by providing new services by incorporating new components.

2.4. Coloured Petri Nets (CPN)

A CPN is a directed bipartite graph with annotations on its elements [78]. In a CPN, the nodes are of two distinct types, known as *places* and *transitions*, and visualized by ellipses and rectangles. CPN tools [79], a widely used modeling environment, adopts this form of visualization. At any moment, a place contains a possibly empty multiset of structured data called *tokens*. The distribution of the tokens among the places is known as *marking*. Place marking is visualized as a collection of tokens inside that place. The *arcs* between the nodes must satisfy the structural constraint that they only can connect nodes of different types, either from places to transitions or from transitions to places. The places and their respective arcs incident to a transition are known as the *input places* and *input arcs* of the transition, respectively. Likewise, the places and their respective arcs incident on them, starting from a transition, are known as the *output places* and *output arcs* of the transition. *Annotations* are written on the elements of a CPN using the functional programming language SML as in CPN Tools, according to well-defined syntactic and semantic rules. Annotations on the input arcs impose constraints on the selection of the resources, whereas the annotations on the output arcs establish their transformation rules. The scope of the variables used in the annotations is centered around each transition. An input arc annotation is an expression

that uses variables to designate either a token or one of its components within the token structure. In concurrent systems, places correspond to passive resource storage sites, while transitions are actions intended to move resources from input places to output places, possibly transforming them according to a rule of transformation.

In CPN, the basic operations of movement and transformation of resources, grounded on variable substitution and annotation instantiation under a substitution, are centered on transitions. Substitution is an assignment of values to the variables occurring in at least one annotation of the input arcs for any transition. A *consistent substitution* produces instances of the input arc annotations that can be identified as tokens in the marking of the corresponding input places. A transition is *enabled* if a consistent substitution satisfies the transition condition (guarding). An enabled transition is eventually fired as long as it remains enabled. *Firing* an enabled transition changes the distribution in the marking of the input and output places of the transition.

2.5. ECA Rules

An event–condition–action (ECA) rule is a computational model for event-driven architectures [80]. An ECA rule is structured in three parts:

- The event part specifies an event expression that binds the values observed in the change of the computational context to the variables occurring in the expression;
- The condition part specifies a logical expression that restricts the values passed from the event part;
- The action part specifies the changes exerted on the computational context but only for those values that satisfy the condition part.

The computational context comprises all the objects that respond to actions of the action part by exhibiting changes in their structure or data content. ECA rules use varied forms of shared data storage. Active databases [81] are extended database systems that react to the CRUD (create, read, update, delete) operations performed on tables. Production systems (PS) [82] are collaborative systems for agents that react to the rising data patterns observed in working memory after modifying its content. Besides, the abstract computational model of CPN extends the single working memory unit of PS to a set of partitioned possible shared working memory units corresponding to the set of places in CPN. Under this interpretation of CPN, ECA rules provide an operational reading for them, where transitions are instances of rules, enabled transitions are selectable rules for execution, firing transitions correspond to rule execution, and marking to event detection. Depending on the level of abstraction of the ECA rule language used, CPNs can be understood as the operational interpretation of ECA rules, whereas ECA rules correspond to their declarative counterpart.

The IoT system development methodology uses this declarative-operational interpretation of ECA rules and CPNs. This approach paved the way in system development for transiting from more abstract description models to more detailed ones.

3. Methodology

SCT-IoT proposes a reduced set of interaction patterns to facilitate the development and interoperability of IoT systems. In order to provide the editing services of the SCM-IoT model in the BPEL service composition model, the producer-editor-subscriber pattern of interaction is designed, modeled, and implemented. The new constructs provided by the SCM-IoT model correspond to the new XML mark-up elements added.

The representation of the extensions in CPN provides a theoretical framework to formulate and verify that the system has desirable properties such as progress, boundedness, and invariance. For the presentation of the proposed extensions, they are described with different levels of abstraction, from the conceptual level of the problem domain to the detailed level of its implementation as follows:

- Conceptual model;
- Platform independent computational model;

- Platform dependent computational model.

The conceptual model defines the data model of the IoT system described by means of tables, including entries for data produced by sensors, for data monitored and modified by the editor, and for data to be notified to the actuators. In the platform-independent model, an abstract computational model such as CPN is used to formally describe both the tables designed in the conceptual model and the instances of the CPNs corresponding to the design patterns of SCM-IoT operations. Finally, in the platform-dependent model, an orchestration language such as BPEL is used to code the CPNs obtained from the platform-independent model. These models are described in more detail in the following subsections.

3.1. Conceptual Model

The SCM-IoT model consists of a centralized, shared, stable storage medium for uncoupled interaction among agents, where a privileged agent, called the (*content*) *editor*, controls the storage access through storage services provided to the other agents. The editor can analyze, modify, produce and store new data into the storage medium based on predefined rules. The editor can consume, produce, and publish data by receiving service requests and perceiving changes in the storage contents, making the storage change. Publishing data generates and propagates notifications about the rising of expected data from the storage. Clients of the editing services are differentiated between (*content*) *producers* and (*content*) *subscribers* according to the services they request.

SCM-IoT proposes an architectural pattern for the design and implementation of IoT systems with loosely coupled components in a data-oriented approach as opposed to the entirely based message-oriented approach. Figure 1 shows the SCM-IoT deployment scenario.

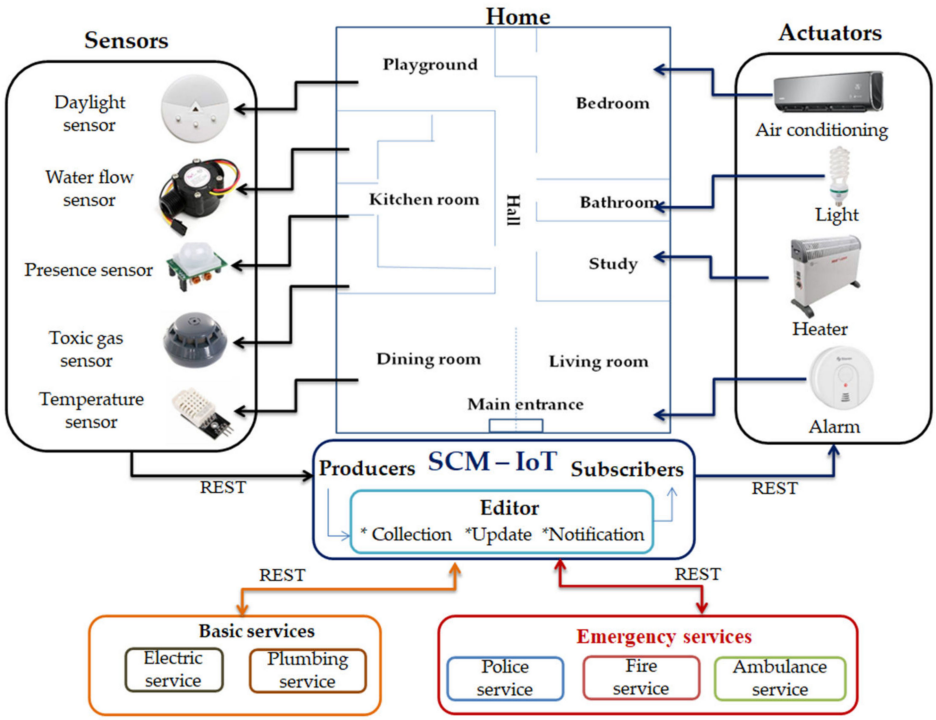


Figure 1. SCM-IoT deployment scenario.

The storage medium is organized in tables with fixed structures. Tables are fixed collections of data where no rows can be deleted, nor new rows can be added. Column names are called *topics*. Rows are organized in fields of some type (string, enumeration, boolean, integer, or real) that store pieces of information. A row always has a *key* field that uniquely identifies it. In SCM-IoT, rows and tables are called *pages* and *notebooks*, respectively, and the message containing a request service is a *notification*, or simply, *note*.

Table 1 shows the structure and content of a notebook for the home automation application presented in the case study. This table has seven pages (rows) and nine fields (columns). Each page is identified by the unique value of the "id" field.

Table 1. Room state.

ID	Name	Presence	Temperature	Smoke	Toxic Gas	Heater	Cooler	Alarm
PAS	HALL	NO	NOR	NO	NO	NO	NO	NO
SAL	LIVINGROOM	NO	BAJ	NO	NO	NO	NO	NO
COM	DININGROOM	SI	BAJ	NO	NO	NO	NO	NO
COC	KITCHEN	SI	NOR	NO	NO	NO	NO	NO
REC	BEDROOM	NO	NOR	NO	NO	NO	NO	NO
EST	DEN	NO	NOR	NO	NO	NO	NO	NO
BAN	BATHROOM	NO	NOR	NO	NO	NO	NO	NO

The notebook content shown in Table 1 shows the last state of the house as it is perceived by the sensors (presence, temperature, smoke, and toxic gas) installed in each room of the house. In this description, the state of the actuators (heater, cooler, alarm) is also included. The page representing the state of the dining room (with "COM" as "id") shows that this room is occupied, has low temperatures, and has no smoke and no toxic gas detected, with all the actuators not activated.

In general, the information contained in the notebook has two properties:

- At all times, the book contains the latest known information until a new update service is successfully consumed;
- Participants can serially access the notebook contents through the editor services, eliminating many concurrent access problems to the shared memory, such as those observed in distributed databases.

Besides the primary editing notebook, there are also secondary notebooks that the editor uses to perform his services. The subscription notebook is used to register clients who wish to receive notifications of relevant data arising in its content due to the editor activity. The notification notebook is also used to register subscription data before sending it to the subscriber.

The three basic storage services that the editor provides to their clients are:

- *Update*, for modifying the data contained in the storage with new incoming or calculated data;
- *Subscribe*, for resuming clients' activity awaiting pages whose content satisfies the so-called notification conditions;
- *Collect*, for retrieving all pages from a notebook that satisfy a collection condition;
- There is no limit in the number of notebooks for edition. Each notebook may differ in its structure according to its intended purpose.
- The services characterize the role of the actors that request them to the editor;
- Content producers request update services to introduce new data into a field or modify existing page data. Production notes contain all the necessary information to determine the page and the field addressed for the update;
- Content subscribers request notification services on specific data contained in the storage as soon as available. Subscription notes contain the unique identifier of the corresponding subscriber and the notification condition upon the storage contents the subscriber is interested in being notified of.

In IoT, particularly home automation, sensors and actuators are identified respectively with producers and subscribers. Producers originate the data to update the book by intermediation of the editor. Subscribers receive notifications from the editor about the logical conditions that are satisfied with the notebook's contents. Thus, an application in IoT develops an observable behavior for the data flow, as a data flow that transits from sensors to actuators. The editor act as a broker that uses the notebooks for data reception,

transformation, and further notification. SCM-IoT simplifies the scheduling of activities since the interactions between participants are generally very complex. In SCM-IoT, sensors and actuators are grouped by the topic of environmental variable they are intended to deal with. Among the topics often used in domotic applications are, for example, human presence detection, temperature, humidity, or light intensity.

As shown in Figure 1, the architecture has a layered design, organized in production layer, edition layer, and subscription layer:

- The production layer consists of relatively independent client producers responsible for generating the primary content of the storage medium. The production layer emits update note requests to the edition layer for this aim;
- The edition layer receives all the update request notes addressed by the production layer. If relevant, the notes are accepted to be posted in a field of a notebook page. The editor judges the relevance of an update request, deciding on its acceptance according to a set of rules. A simple relevance rule is that the datum in the note must differ from the one already contained in the corresponding page's field, avoiding unnecessary editor actions. The editor monitors each update, filling other fields on the same page or other pages, following system application rules. The end of the editor's activity triggers the test of the notification conditions of all subscribers and to carry out the notification to all the subscribers whose notification conditions hold;
- The subscriber layer consists of a collection of independent clients who are ultimately responsible for undertaking the necessary actions to ensure compliance with the system's overall goals. The subscribers' activities are triggered after their notification or collection conditions have been met upon the current state of the storage contents.

The editor aims to preserve the consistency and integrity of the data stored in the notebook, acting as a provider of services for querying, updating, and notifying the notebook contents. The following elements characterize the evolution of the behavior of a system using the SCM-IoT model:

- Clients carry out their activities independently of each other but indirectly interacting through the editor's storage services;
- The data flow direction differentiates the editing services in the notebook. The data coming from a note are entered into the notebook via an update service, whereas data coming from the notebook are copied out into a note via a notification service;
- Once the satisfied notification conditions have been identified, the content editor selects one notification non-deterministically and sends it to the subscriber that requested it;
- The activities performed by any participant should always terminate, leaving the notebook in a consistent state;
- Verification of editor actions ensures the logical consistency of the notebook contents. Termination and verification ensure that the contents are logically consistent upon completing a sequence of interactions among the participants.

3.1.1. Update

The update storage service modifies a field in a page of a notebook based on a request from the producer. Thus, the note with the request must provide the notebook's name, the page's identifier, the topic of the field, and the new data to replace the previous one. However, the editor accepts not all update requests but only those that fulfill the condition given in the request. The update is applied on the indicated page when the condition is missing. In any case, only updates with different data from the current one are accepted. The events of receiving update requests trigger editing actions into the notebook contents. Such modifications must always end eventually, leaving the notebook in a stable state.

3.1.2. Subscription for Notification (Notify)

A subscription for notification is a page-oriented service whose purpose is to notify the subscribers that a page content has reached a state where the notification conditions are

met. After an update, the editor checks if the notebook content satisfies some notification conditions. Once all the changes to the storage medium are completed, the editor looks for the table relating fields modified by topic with the corresponding subscriber. Under this approach, the update operations are the activity triggers looking for applying pertinent application rules.

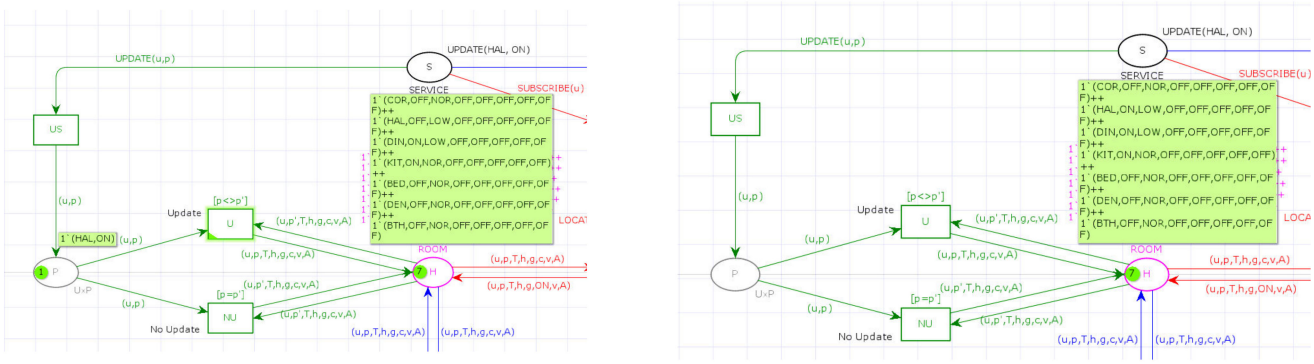
3.1.3. Subscription for Collection (Collect)

A subscription for collection extends the subscription for notification service from one to several content subscribers, all interested in recognizing the rise of data that meet a collection condition. The collection is an indivisible operation across all pages of the notebook. Once the pages of interest have been identified for collection, they cannot be used in other notifications or collections until completing the service. This restriction guarantees the consistency and integrity of the notebook concerning the invariance conditions established on its content. Uninterrupted notification delivery to all the corresponding subscribers avoids interleaved access to other clients. The indivisibility of the collection service can be ensured by giving this service the highest priority of attention. If more than one collection conditions hold, one is chosen non-deterministically for its application.

In the following subsections, we present and describe the formalization and validation of the editor services.

3.2. Platform Independent Computational Model

The SCM-IoT model was formally described through CPNs, allowing for the possibility of rigorous verification of some of its properties to be visualized, analyzed, validated, and enabled. In the CPNs shown in Figure 2 of Section 3.2.1, the notebook is represented by location H, accessible to three transitions U, N, and C that represent the update, subscription for notification, and subscription for collection services, respectively, as described in the conceptual model.



(**a**) (**b**)

Figure 2. Update service behavior. In (**a**), transition U is enabled by request on place *P*. In (**b**), the effect of changing the notebook contents of place H is shown.

The content of the notebook corresponds to the marking of place H, given by its current marking with the multiset expression "1′ (PAS, NO, BAJ, NO, NO, NO, NO, NO) ++ 1′ (SAL, YES, BAJ, NO, NO, NO, NO, NO) ++ . . . ". This expression is taken from the case study presented below. The expression consists of the sum of several tuples representing tokens, each one multiplied by their corresponding multiplicities, i.e., the number of times it appears repeated in the multi-set. Thus, the token "(PAS, NO, BAJ, NO, NO, NO, NO, NO)" appears only once in the book. The expression "(u, *p*, T, h, g, c, v, A)", annotated on the input arc of transition U to place H, declaratively indicates how the values are assigned to the variables that appear in this expression. For the token given before, the values are assigned sequentially in the order in which they appear from left to right, so the list of assignments is "u = PAS, *p* = NO, T = LOW, h = NO, g = NO, c = NO, v = NO, A = NO". The

variable names that appear to the left of each assignment can be used in logical conditions such as "p = SI". According to the SCM-IoT model, the tokens represent the notebook pages, and the data separated by commas and grouped by parentheses represent the fields. The number of elements in the multiset, including duplicates, is circled within the marked place, as shown in Figure 2a.

The editor services associated with the three transitions U, N, and C, mentioned above, are described in detail below.

3.2.1. Update

The update service modifies the notebook's contents, but only on one page and one field at a time. This service is represented by the structural pattern given by the CPN marked in green as shown in Figure 2a,b. The request originates from the service request place S with the token "UPDATE (COM, SI)" indicating to make the update in the page identified by "COM" for the field "PRESENCE" with the value "YES", meaning that someone is in the dining room as shown in Figure 2a. This request is left in place "P" where the presence update requests are attended.

The structural pattern of the CPN that represents this operation consists of two transitions U and NU, both having as input places the notebook H and the request container P for receiving update notes. In Figure 2a, note "(COM, SI)" can be seen in P and the corresponding page "(COM, NO, BAJ, NO, NO, NO, NO, NO)" to be modified in H. The "u" field corresponds to the first field in the last expression, while the "p" field corresponds to the second.

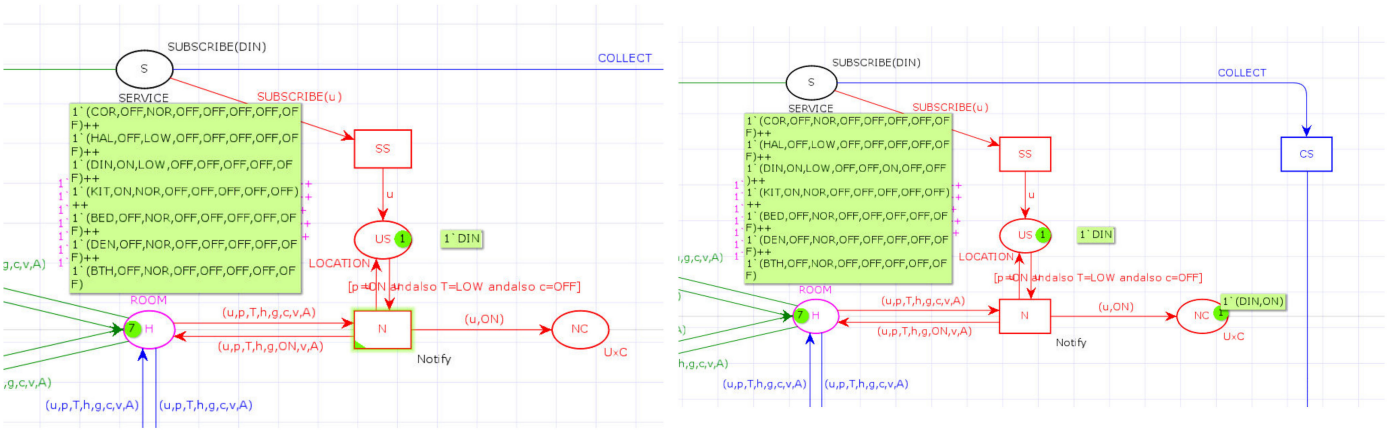
As described in the informal model, the update is carried out only when the note contains a value that is not already registered in H. The above is expressed with the restriction "[p <> p']" noted on the transition U. The conditions of qualification annotated on the transitions U and NU are mutually exclusive among themselves since they indicate the criteria to carry out the update, depending on the values of "p", taken from the note in P, and of "p'", taken from the sheet identified by "COM" in H, as shown in Figure 2b.

In the case of the update note "(COM, NO)", the NU transition is enabled while the U does not, so the actions noted for the NU transition do not cause any change in H. This representation preserves the number of tokens in H and the correspondence and uniqueness between tokens and rooms.

3.2.2. Notify

The subscription service for receiving notifications allows resuming the activities of those subscribers waiting for the emergence of data of interest generated by the editor. Subscription is a service that corresponds to the pattern given by the CPN in red, as shown in Figure 3a,b. The subscription request "SUBSCRIBE(u)" is marked in place S, which enables the SS transition that, when triggered, marks the NS place with the subscription note that contains only the subscriber identifier "u". The subscription ends with this marking in the US place, allowing all subscribers listed here to receive notifications eventually.

The subscription for notifications service has the behavior pattern described by transition N. The notebook for edition is represented by place H, where relevant data is expected to arise. The input arc with annotation "(u, p, T, h, g, c, v, A)", retrieves all tokens from H and binds the variables appearing in the annotation with values.



(**a**) (**b**)

Figure 3. Subscription for notification service behavior. In (**a**), transition N is enabled by the marking, showing only one subscriber in place US. In (**b**), a notification for the subscriber has been deposited in place NC.

As shown in Figure 3a, the enabling and triggering of transition N can be described by the ECA rule:

1. ON detecting (u,*p*,T,h,g,c,v,A) at H AND detecting u at US;
2. IF *p* = ON andalso T = LOW andalso c = OFF;
3. DO remove (u,*p*,T,h,g,c,v,A) from H; insert (u,*p*,T,h,g,ON,v,A) to H.

Notifications are performed by firing transition N, where H is an input and an output place for N, as shown in Figure 3b.

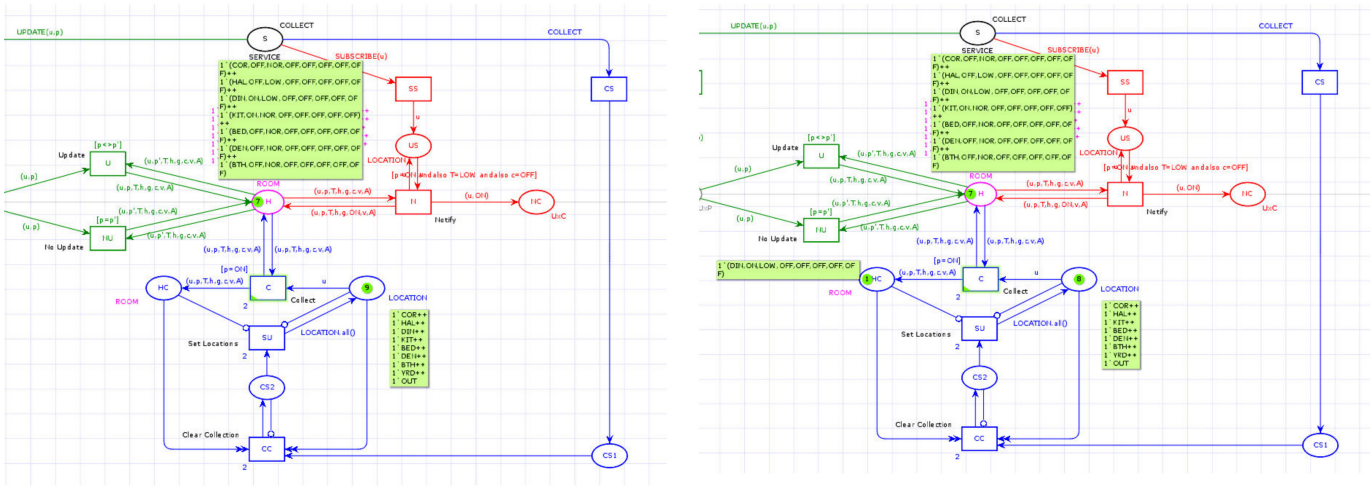
The same tuple expression "(u, *p*, T, h, g, c, v, A)" is annotated in the input and output input arcs, indicating that no field is modified during notification. Though the values assigned to the variables are used to evaluate the logical condition on N. The condition "[*p* = SI, T = BAJ, and c = NO]" establishes the values that the variables "*p*", "T", and "c" must take to verify it. In this way, transition N is enabled when there is a token such as "(COC, YES, LOW, NO, NO, NO, NO, NO)" in H because it assigns the values "YES", "LOW", and "NO" to the variables "*p*", "T", and "c", respectively. Once this transition is enabled, it can be triggered as long as no collection request applies to this token since it has a higher priority of attention. When this enabled transition is eventually triggered, a notification is sent to the subscribed client to receive it. The output arc from N to NC with the notation "(u, SI)" indicates this action.

The notifications do not alter the notebook's content or the list of subscribers, so each subscriber may continue to receive more notifications from time to time. Likewise, since there is a notification when allowing the resumption of the actions of a subscriber who can potentially modify the contents of the notebook, its interleaved execution during the execution of the collection service is not allowed.

3.2.3. Collect

As indicated in the conceptual model, the collection is a service that allows all the notebook sheets that meet a pre-established condition to be retrieved. The representation of this service corresponds to the CPN pattern outlined in blue, as shown in Figure 4a,b. The service request begins with the insertion of the "COLLECT" token in place S. This initial marking causes the transition to be enabled. CS that, when triggered, inserts a unitary token in place CE1. The CC transition is enabled with this marking, thereby initiating a sequence of actions to carry out the collection. The first action is to remove the entire content of the HC place, which is where the collected data reside. The second action is to initialize the place U with all the identifiers corresponding to each notebook page. The third and last action consists of copying in the destination place HC in each notebook page

that satisfies the collection condition "[p = SI]" of transition C as shown in Figure 4a. The three previous actions are carried out sequentially in an indivisible way to guarantee that all the data in the collection comply with the established condition. Figure 4b shows the marking after repeatedly firing transition C until it is no longer enabled when no more pages are found that satisfy the collection condition.



(**a**) (**b**)

Figure 4. Subscription for collection service behavior. In (**a**), transition C is enabled by the marking. In (**b**), a notification for each subscriber found in place U has been deposited in place HC.

The abstract behavior of the editor services described in the CPN formalism allows representing the main features of the system application. The platform-independent computational model is a prototype that is further detailed into a platform-dependent computational model, according to the technological aspects of the infrastructure available.

3.3. Platform Dependent Computacional Model

The data-driven approach of SCM-IoT is based on a reduced set of services that enable IoT systems to exhibit relatively complex intelligent behavior. The services update, subscription, and collection, already described in the previous models, are now described as a platform-dependent computational model built upon the XML, XSL, and BPEL technologies.

The extensible markup language (XML) is a markup language that consists of a set of syntactic rules for describing information in both human-readable and machine-readable formats. The XML schema language uses XML as a metalanguage to rigorously describe the formation rules of XML structures. The extensible stylesheet language (XSL) is a family of XML languages designed to transform XML structures. The path language (XPath), belonging to the XSL family, is heavily used within XSL transformations to access the XML elements and attributes of the structure.

The business process execution language (BPEL) is an XML language for describing business process behavior and interaction protocols. BPEL provides service composition and orchestration through an interoperable model of web services integration to facilitate business process automation. BPEL can approach a business process description from two perspectives. The executable processes model is the actual behavior in business interaction, and the abstract process for business protocols specifies the externally observable behavior through the exchange of messages without revealing internal behavior.

3.3.1. Storage Medium

The storage medium, represented by a collection of tables in the conceptual model and as multisets of tuples in the platform-independent computational model, is represented

as a collection of XML documents in the platform-dependent computational model. The XML elements representing pages, notification, subscription, and editing notebooks are described in Listing 1 using the XML schema language.

Listing 1. Definition of element types for page, notebook, notification, and subscription.

Line	Code
1	<xs:element name = “PageType” minOccurs = “1” maxOccurs = “unbounded”>
2	<xs:complexType>
3	<xs:sequence>
4	<xs:element name = “id” type = “xs:string” use = “required”/>
5	<xs:element name = “topic-1” type = “Topic-1Type” use = “required”>
6	<xs:ComplexType>
7	<xs:attribute name = “time” type = “xs:int” use = “optional”/>
8	<xs:ComplexType>
9	</xs:element>
10	<xs:element name = “topic-2” type = “Topic-2Type” use = “required”> . . . . .
11	</xs:sequence>
12	</xs:complexType>
13	</xs:element>
14	<xs:element name = “NotebookType”>
15	<xs:complexType>
16	<xs:sequence>
17	<xs:element name = “page” type = “PageType”/>
18	</xs:sequence>
19	</xs:complexType>
20	</xs:element>
21	<xs:element name = “UpdateType” minOccurs = “1” maxOccurs = “unbounded”>
22	<xs:complexType>
23	<xs:sequence>
24	<xs:element name = “page-id” type = “xs:string” use = “required”/>
25	<xs:element name = “topic” type = “xs:string” use = “required”/>
26	<xs:element name = “value” type = “TopicType” use = “required”/>
27	</xs:sequence>
28	</xs:complexType>
29	</xs:element>
30	<xs:element name = “SubscriptionType” minOccurs = “1” maxOccurs = “unbounded”>
31	<xs:complexType>
32	<xs:sequence>
33	<xs:element name = “subscriber-id” type = “xs:string” use = “required”/>
34	<xs:element name = “page-id” type = “xs:string” use = “required”/>
35	<xs:element name = “topic” type = “xs:string” use = “required”/>
36	<xs:element name = “notified” type = “xs:boolean” use = “required”/>
37	</xs:sequence>
38	</xs:complexType>
39	</xs:element>

The notebook’s page structure, described by the “PageType” type, contains the field structure. In the “PageType”, the “page” field has a unique value that identifies the page. All other fields are required, have a simple type and an optional “time” attribute. The “time” attribute serves to record the most recent updates of the field, as described later. The “NotebookType” type is defined as a non-empty sequence of “PageType” elements. The service request types, “UpdateType”, “SubscriptionType”, and “NotificationType”, define the structure of updates, subscriptions, and notifications, respectively.

The editor offers access, storage, and generation of new data through the services announced in the description “EditorServicesPT” of its port type, as indicated in Listing 2.

Listing 2. Description "EditorServicesPT" of a port typen.

Line	Code
1	<wsdl:message name = "UpdateMT">
2	<wsdl:part name = "notebook" type = "NotebookType">
3	<wsdl:part name = "page" type = "PageType">
4	<wsdl:part name = "topic" type = "xs:string">
5	<wsdl:part name = "note" type = "UpdateType">
6	</wsdl:message>
7	<wsdl:message name = "SubscriptionMT">
8	<wsdl:part name = "subscriber-id" type = "xs:string">
9	<wsdl:part name = "page-id" type = "xs:string">
10	<wsdl:part name = "topic" type = "xs:string">
11	</wsdl:message>
12	<portType name = "EditorServicesPT">
13	<operation name = "update">
14	<inputMessage name = "UpdateMT"/>
15	</operation>
16	<operation name = "subscribe">
17	<inputMessage name = "SubscriptionMT"/>
18	</operation>
19	</portType>

In order to shorten and simplify the presentation, in Listing 3 all the pieces of information needed to describe the behavior of the editor attending the requested services are available through BPEL variables with global scope.

Listing 3. Description of the behavior of the editor in BPEL.

Line	Code
1	<bpel:variables>
2	<bpel:variable name = "editing" type = "xsd:boolean"/>
3	<bpel:variable name = "notebook" element = "NotebookType"/>
4	<bpel:variable name = "note" element = "NotificationType"/>
5	<bpel:variable name = "subscription" element = "SubscriptionType"/>
6	</bpel:variables>

The global scope variable "editing" synchronizes exclusive access to the notebook when relevant data has emerged for a group of subscribers. When the editor checks the notebook's content, no client can be considered for service.

To attend to the editing services, the procedure followed by the editor consists of repeatedly performing two activities sequentially: first, receiving service requests, then making changes to the content of the notebook according to the requests received. Listing 4 shows a fragment of this procedure that shows in detail. The attention to the update request has a very similar structure. The relevance of this procedure lies in showing in detail how the integration of SOA has been achieved through a message-oriented language and the representation and administration of a shared storage medium where decisions with various degrees of complexity can be analyzed and made.

Listing 4. Main control loop for receiving services requests before notebook edition.

Line	Code
1	<bpel:if>
2	<bpel:condition>not($editing)</bpel:condition>
3	<bpel:sequence>
4	<bpel:pick>
5	<bpel:onMessage operation = “update” variable = “note” portType = “NotificationPT” partnetLink = “Producer”>
6	<bpel:scope>
7	<bpel:variables>
8	<bpel:varaible name = “samepage” type = “xsd:boolean”/>
9	<bpel:varaible name = “valueinnotebook” type = “xsd:string”/>
10	<bpel:variable name = “valueinnote” type = “xsd:string”/>
11	</bpel:variables>
12	<bpel:sequence>
13	<bpel:assign><bpel:copy>
14	<bpel:from>
15	<bpel:query>($notebook/$page/@id = $note/@page-id)</bpel:query>
16	</bpel:from>
17	<bpel:to variable = “samepage”/>
18	</bpel:copy></bpel:assign>
19	<bpel:assign><bpel:copy>
20	<bpel:from><bpel:query>
21	($notebook/$page[@id = $note/@page-id]/@topic)
22	</bpel:query></bpel:from>
23	<bpel:to variable = “valueinnote”/>
24	</bpel:copy></bpel:assign>
25	<bpel:assign><bpel:copy>
26	<bpel:from><bpel:query>($note/@topic)</bpel:query></bpel:from>
27	<bpel:to variable = “valueinnotebook”/>
28	</bpel:copy></bpel:assign>
29	<bpel:if>
30	<bpel:condition>
31	$samepage and $valueinnotebook != $valueinnote</bpel:condition>
32	<bpel:assign><bpel:copy>
33	<bpel:from>bpel:doXslTransform(“update.xsl”, “$notebook”, “$page”, “topic”, $note)</bpel:from>
34	<bpel:to variable = “notebook”/>
35	</bpel:copy></bpel:assign>
36	</bpel:if>
37	</bpel:sequence>
38	</bpel:scope>
39	</bpel:onMessage>
40	<bpel:onMessage operation = “subscribe”
41	variable = “subscription” port-Type = “SubscriptionType” partnetLink = “Subscriber”>
42	. . .
43	</bpel:onMessage>
44	</bpel:pick>
45	<!- - Edition and Notification activities shown in next Figure - ->
46	. . .
47	</bpel:sequence>
48	</bpel:if>

Lines 1 and 2 establish the exclusion condition to accept new requests. The condition establishes that the editor has exclusive access until the completion of his editing activities as indicated by the variable “editing”. Line 4 establishes the selection dictated by the BPEL implementation to determine which message will be served next. If two requests are

received simultaneously, one is selected non-deterministically while the other is discarded. In a relatively slow-changing IoT environment, losing some messages do not negatively affect the system performance. In line 5, the received message is declared and accepted in the <pick/> branch by <onMessage/>. The port type of the update service, the link to the business partner, and the variable where the request information will be stored are also indicated. Lines 6 and 38 delimit the scope of the local variables declared in lines 8 to 10. Lines 13 to 18 show the assignment of the logical expression value given in line 15 to the variable "samepage" that establishes the first condition to update the notebook. The second condition is established from lines 19 to 28. In lines 19 to 24, the requested data are received in "valueinnote" variable. The value registered in the notebook is assigned into the variable "Valueinnotebook", as indicated in lines 25 to 28. Both assignments display the Xpath expressions to obtain the indicated values. In lines 29 to 36, it is decided whether or not to carry out the update. The updating criterion is given by the conjunction of the two previous conditions. First, the values to be compared correspond to the same notebook, page, and topic, and second, the value received is different from the one previously recorded. When both conditions are met, the update is performed through an XSLT transformation to the notebook's contents. Since the notebook structure remains unchanged as long as the number and type of publisher clients remain unchanged, the corresponding "update.xsl" operation is straightforward. In line 33, the invocation of the transformation procedure appears together with its current parameters. In line 34, once the transformation of the notebook is completed, the updated structure is assigned to the same variable. Finally, between lines 40 and 43, the second branch of <pick> appears schematically to receive subscription requests.

The invocation to doXslTransform is what transforms the edit notebook. Notice the update of the attribute "time" defined for all fields. This attribute establishes how recent the update is, so newly updated fields will include this attribute with the value of "0". The value of this attribute is decremented by one with each update (by invocation of doXslTransform). Accordingly, in the next update of the notebook, those fields that had the value of "0" will now have the value "−1", which makes it possible to distinguish them from the newly updated ones because the latter will have the value of "0" as said. At the end of the next update of the notebook, the fields whose attribute "time" had the value "−1" will now have the value "−2" and so on. Although this counting can be continued, it is enough to limit the count up to "−2". Fields that have reached their "time" count limit are considered not changed recently. As unchanged fields can be easily located, it is preferable to eliminate their "time" attribute to simplify matters. In summary, the "time" attribute of each field of a page can:

- Have the value "0", indicating that it is a recently updated field;
- Have the value "−1", indicating that it is a previously updated field;
- Have no value, indicating that it is either older or has never changed.

The "time" attribute is important because it allows one to quickly identify those fields that have been recently updated so that the editor can more efficiently evaluate the sheets in which they appear. The newly updated pages allow defining the editing actions to be applied next. These actions can modify other fields on the sheet, triggering notifications to the corresponding subscribers. The editor is in charge of carrying out all these actions during the edition of the notebook. The editing procedure is shown in the Listing 5.

Listing 5. Edition and notification to interested subscribers.

Line	Code
1	<bpel:assign><bpel:copy>
2	<bpel:from><bpel:query>true()</bpel:query></bpel:from>
3	<bpel:to variable = “editing”/>
4	</bpel:copy></bpel:assign>
5	<bpel:assign><bpel:copy>
6	<bpel:from>bpel:doXslTransform(“edition.xsl”, $notebook)</bpel:from>
7	<bpel:to variable = “notebook”/>
8	</bpel:copy></bpel:assign>
9	<bpel:scope>
10	<bpel:variables>
11	<bpel:variable name = “notification” element = “NotificationType”/>
12	</bpel:variables>
13	<bpel:while>
14	<bpel:condition>
15	<bpel:query>exists($notebook/$page/$topic[@time = “0”])</bpel:query>
16	</bpel:condition>
17	<bpel:sequence>
18	<bpel:assign><bpel:copy>
19	<bpel:from>bpel:doXslTransform(“doEdition.xsl”, $notebook)</bpel:from>
20	<bpel:to variable = “notebook”/>
21	</bpel:copy></bpel:assign>
22	<bpel:assign><bpel:copy>
23	<bpel:from>bpel:doXslTransform(“doNotification.xsl”, $notebook, $subscription, $notification)</bpel:from>
24	<bpel:to variable = “notification”/>
25	</bpel:copy></bpel:assign>
26	<bpel:invoke operation = “notify” inputvariable = “notification” portType = “EditionServicesType” partnerLink = “Subscriber”/>
27	</bpel:sequence>
28	</bpel:while>
29	</bpel:scope>
30	<bpel:assign><bpel:copy>
31	<bpel:from>
32	<bpel:query>false()</bpel:query>
33	</bpel:from>
34	bpel:to variable = “editing”/>
35	</bpel:copy></bpel:assign>

Lines 1 to 4 show the assignment of the logical value true to the exclusion variable “editing”, which prevents the acceptance and attention of new service requests. Lines 5 to 8 show the transformation of the edition and notification table, applying the transformation rules given in “edition.xsl” as shown in line 6. The updated notebook is saved again in the same variable to accumulate the changes indicated on line 8. In lines 9 to 29, the scope is defined for the variable entered in the declaration that appears on line 11. The notification cycle to all subscribers for relevant data is shown in lines 13 to 28. In line 15, the editing condition indicates a field with the attribute “time” set to “0”. When this condition is met, the transformation of the notebook is carried out, as indicated in line 19. The transformed notebook (lines 18 to 21) is assigned to the same variable (line 20) to accumulate the updates. Once the notebook has been updated, the editor searches for relevant data and prepares the notification to the corresponding subscribers. The preparation consists of updating the notification notebook, noting the data of interest for the topic and the corresponding page. Then, the editor sends the notification to the corresponding subscriber, as shown

in line 26. The edition and notification cycle ends when there are no more changes in the notebook, as indicated by the absence of any field with "time" attribute throughout all the notebook' pages.

Once the edition is finished, the notebook's content remains stable until a new update occurs, starting a new edition cycle on the notebook. This behavior is consistent with the behavior analyzed in Section 3.2.1.

3.3.2. Notify

The interface of the subscription service appears in the portType. The editor's behavior follows a similar structure to the update service, except that the updates are made on the subscription notebook and not on the edition notebook itself. Thus, instead of activating the editor with each update request, the subscribers are activated with each notification condition satisfied, as discussed in Section 3.2.2.

3.3.3. Collect

The interface of the subscription service appears in the portType. The behavior is similar to the subscription for notification service. The essential difference between both services refers to the sheets involved and the destination of the update. While the subscription service refers to only one sheet in a notebook, the collection service refers to only one field throughout all the sheets of the notebook. While the subscription service affects the same notebook page, the collection service affects other notebooks, pages, and fields.

In the next section, a case study shows some of the benefits of the approach posed by the SCM-IoT model. This model allows the integration of the interoperable infrastructure that the intelligent behavior the application needs.

4. Case Study

An application to home automation of the SCM-IoT model is described below to demonstrate its flexibility in describing interactions in IoT systems. The shared storage model allows describing complex events that require the combination of data generated by various sensors to make decisions, such as, for example, to notify emergency systems when very high temperatures are observed with the presence of smoke and toxic gases, as well as to determine the room where the residents of a house are.

4.1. Conceptual Model

The requirements that computational models must meet are listed below:

- Determining at all times the ubication (in one of the rooms) of any inhabitant of the house;
- Turn on the heater whenever an occupied room has a low temperature. Turn it off when either the room is no longer occupied or the temperature rises over a threshold;
- Turn on the AC whenever an occupied room has a high temperature. Turn it off when either the room is no longer occupied, or the temperature falls below a threshold;
- Assess possible emergencies such as fire by analyzing the data reported from temperature, smog, and toxic gas sensors installed in each room of the house;
- Determine if the house is on fire if three conditions are met in any room: very high temperature, presence of smog, and presence of toxic gas;
- Turn off the heater, the AC, or both for all rooms if the house is on fire;
- Make urgent calls to the emergency call center, reporting the room in which the incident started;
- Also, determine if the house is inhabited. Report to the rescuers about those rooms where residents were detected since the last received sensor data.

The development of the SCM-IoT platform-independent computational model consisted of defining the data model and the three storage and editing services.

The fields describe the data associated with each sensor or actuator installed in a room. The most relevant fields are the following:

- ID: room identifier;
- Name: usual name of the room;
- Presence: indicates whether a resident is present in the room (YES or NO);
- Temperature: indicates the temperature range in which the temperature detected by the sensor is in the room (LOW, NORMAL, HIGH, VERY HIGH);
- Smoke: indicates whether the corresponding sensor has detected smoke in the room (YES or NO);
- Toxic gas: indicates whether the corresponding sensor has detected toxic gas such as carbon dioxide in the room (YES or NO);
- Heater: indicates whether the room heater is in operation (YES or NO);
- Cooler: indicates whether the air conditioner in the room is operating (YES or NO);
- Alarm: indicates whether the room's audible (and visual) alarm is activated (YES or NO).

Having described the requirements for the IoT system, a more detailed presentation of the comfort and energy save conditions are given next.

4.1.1. Comfort Conditions and Energy Saving

Among the comfort conditions that the system must guarantee is that the temperature of a room is pleasant for those who occupy it. In other words, the temperature is between 15 and 25 °C, being these limits are subject to configuration adjustments according to personal preferences. However, energy-saving conditions are also considered, stating that energy must be only consumed when the room is occupied. Consequently, the conditions to be jointly met are that:

- The most recently updated temperature is low (comfort condition);
- The room is occupied by at least one resident (energy saving condition);
- The heater or fan (air conditioning) in the room is turned off (condition of reduced network traffic and consequently emergency saving).

The comfort conditions are the conditions noted in the corresponding transitions in the instances of the service patterns, as analyzed in the computational models as shown in Sections 4.2 and 4.3.

4.1.2. Safety Conditions

Safety conditions refer to providing the necessary actions to face contingencies such as fires. To achieve this end, the conditions that determine a fire are precisely described, and the rooms are equipped with the necessary detection equipment that provides the required information on time. The conditions that are jointly fulfilled to determine the existence of fire are:

- The most recent temperatures obtained by the sensor are VERY HIGH, that is, well above the usual limits of high temperatures in hot or desert climates;
- The presence of high concentrations of smoke, which is a widely known sign of fire;
- The presence of toxic gases, such as carbon monoxide, is additionally a venous gas.

The safety conditions are established as conditions in the corresponding transitions in the service patterns as analyzed in the computational models exposed next.

4.2. Platform Independent Computational Model

Figure 5 shows a partial view of the CPN built systematically using the structural design patterns of the editor services.

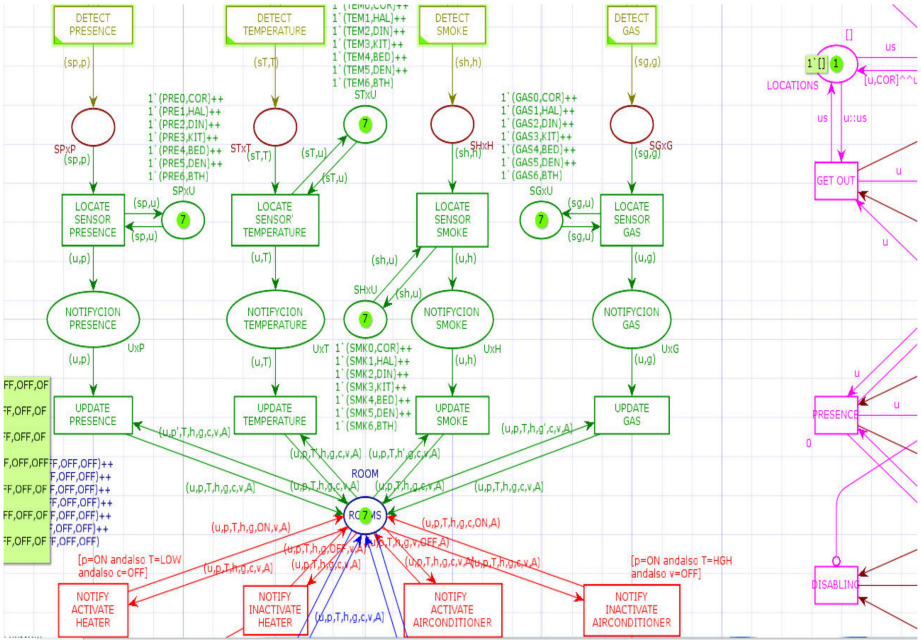


Figure 5. CPN systematically constructed using the SCM-IoT patterns.

In Figure 6, the ellipse representing the notebook appears in the center in pink, with seven pages, one for each room. In Figure 6, the CPNs representing the instances of structural patterns that serve the publisher update services appear in the upper part in green. At the bottom, in red, the CPNs that represent the instances of the subscription–notification service patterns appear. It should be noted that the fragment corresponding to the subscription has been omitted to reduce the excess of visual elements that impede its understanding. Likewise, in the lower part, in blue, appear the CPNs that represent the pattern of the collection service.

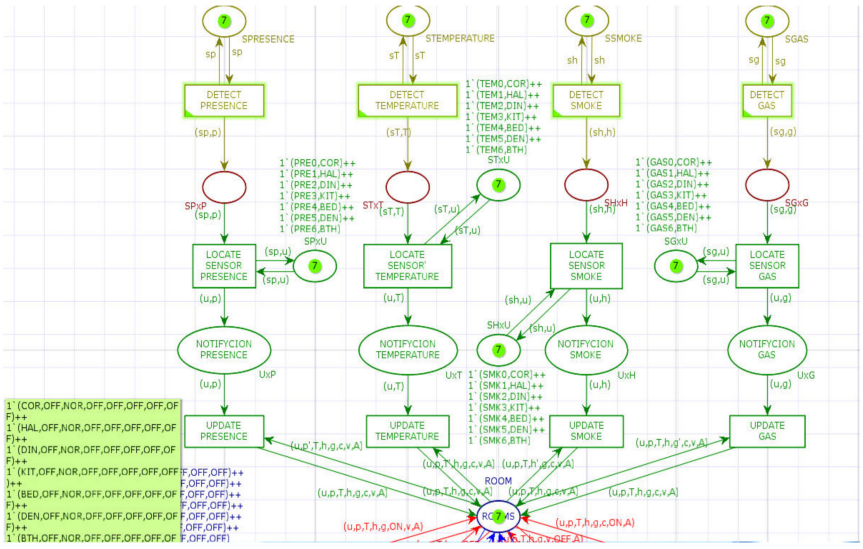


Figure 6. CPN of the update service for the case study.

Figure 6 shows the representation of the update service in greater detail. Compared to the update pattern shown earlier in Figure 2, only the branch with the transition with condition "$p <> p'$" has been included here.

Figure 7 shows the detail of the subscription notification and collection services. The notification conditions for the activation or deactivation of the heater are shown. The activation condition "[*p* = SI andalso T = LOW andalso c = NO]" in the transition, determines the updating of the field "c" that refers to the state of the heater, changing the entry arc annotation from "(u, *p*, T, h, g, c, v, A)" to "(u, *p*, T, h, g, SI, v, A)" in the starting arc annotation. Similarly, the inactivation condition "[(*p* = SI andalso T = NOR andalso c = YES) or else (*p* = NO andalso c = YES) or else (A = SI andalso c = YES)]" specifies the cases in which that the heater should be turned off corresponding to the cases: (i) when the room is occupied with normal temperature, and the heater is on, (ii) when the room is not occupied, but the heater is on, and (iii) when the fire alarm has been activated, and the heater is on. The activated alarm condition is assumed to be equivalent to the house on fire.

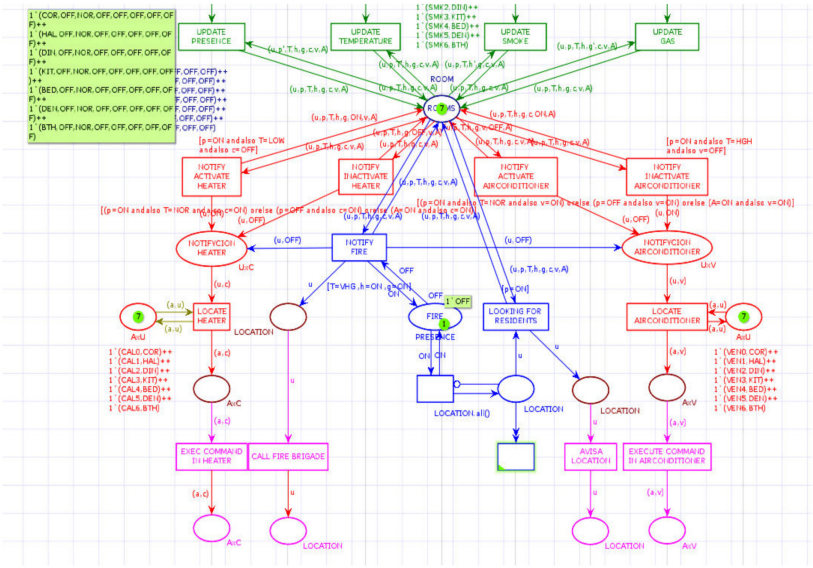


Figure 7. CPN subscription for notification and collection services.

Finally, in Figure 7, the CPN shows the collection and notification service. The three criteria give the condition to start the collection to determine a fire situation in at least one house room: very high temperature, smoke, and toxic gas present. The transition "NOTIFES FIRE" activates the search for residents in each room of the house under the condition "[*p* = YES]". The information with all the resident locations is sent to the emergency services to facilitate their rescue.

4.3. Platform Dependent Computational Model

A note describing the most relevant aspects of the room status information is represented as an XML element, which contains a fixed set of sub-elements, each representing a field of the note. The note and field names are as indicated above. Listing 6 shows an example of a note in XML.

The page type of a notebook contains minimal domotic information about the room of a house. Listing 6 shows the room "LIVINGROOM", identified as "LIV", occupied by a resident. The room has a "LOW" temperature for what the system has activated the operation of the heater ("YES"), deactivated the air conditioning ("NO"), and no detection for smoke nor toxic gas in the room. Thus, the emergency alarm is deactivated.

Listing 6. Note in XML.

Line	Code
1	<room id = "ROOM">
2	<name>LIVINGROOM</name>
3	<presence>YES</presence>
4	<Temperature>LOW</Temperature>
5	<smoke>NO</smoke>
6	<toxicgas>NO</toxicgas>
7	<heater>YES</heater>
8	<ventilator>NO</ventilator>
9	<Alarm>NO</Alarm>
10	</room>

4.3.1. Definition of the Notebook Data Model

The edition notebook contains home automation information for each room as defined above. The book records the known information on the status of each room obtained from the last update reported by the sensors and from the decisions taken to put the actuators into operation or not. However, the book has to be placed alongside other notebooks, such as sensors and actuators notebooks, thus forming a collection and defining the appropriate context for treating all relevant information for the IoT application. As illustrated in Listing 7, the element "house" is the name of the book collection and establishes the context in which the room book is located.

Listing 7. Notebook "rooms".

Line	Code
1	<home>
2	<rooms>
3	<room id = "HAL">
4	<name>HALL</name>
5	...
6	</room>
7	<room id = "LIV">
8	<name>LIVINGROOM</name>
9	...
10	</room>
11	...
12	...
13	</rooms>
14	<sensors> ... </sensors>
15	<actuators> ... </actuators>
16	</home>

The notes found in these sub-elements contain the minimum essential information on the installation location of home automation devices. Consequently, the location determines the room where the sensors detect environmental information or how the actuators modify the local ambient conditions. A note from a sensor that essentially includes the identifier and reading fields. The reading field contains the information that originates in the environment, corresponding to the most recently obtained by the sensor. Listing 8 presents an excerpt from the notebook for the sensors.

Listing 8. Notebook "sensors".

Line	Code
1	<home>
2	<rooms> . . . </rooms>
3	<sensors>
4	<presence>
5	<sensor id = "PRE0">
6	<reading>NO</reading>
7	</sensor>
8	. . .
9	</presence>
10	<Temperature>
11	<sensor id = "TEM0">
12	<reading>LOW</reading>
13	</sensor>
14	. . .
15	</Temperature>
16	<smoke>
17	<sensor id = "SMO0">
18	<reading>NO</reading>
19	</sensor>
20	. . .
21	</smoke>
22	<toxicgas>
23	<sensor id = "TOX0">
24	<reading>NO</reading>
25	</sensor>
26	. . .
27	</toxicgas>
28	</sensors>
29	<actuators> . . . </actuators>
30	</home>

The sensors were grouped by type into sensors for presence, temperature, smoke, and toxic gas under corresponding XML elements of the same name. The other types of sensors have a similar note structure except for the temperature sensor, whose values reported in the reading field take the values indicated above ('LOW', 'NORMAL', 'HIGH', 'VERY HIGH'). The notebook for actuators has a similar structure for sensors, grouping under the XML element actuators, grouped according to their type into heaters, fans (air conditioners), and alarms. The essential information that each note in the actuator book contains is its identifier and its operating status. An extract from the notebook for the actuators is presented in Listing 9.

Unlike the sensors, the status field is writable because it indicates the order given to the actuator to change its operation immediately, assuming that it is on and in good operating conditions.

4.3.2. Definition of Notification Conditions for Actuators

Notification conditions for actuators are classified into two parts, comfort and safety.

Comfort and energy-saving conditions. Among the comfort conditions that the system allows to guarantee is to ensure that the temperature of a room is pleasant for those who occupy it.

Listing 9. Notebook "actuators".

Line	Code
1	<home>
2	<rooms> . . . </rooms>
3	<sensors> . . . </sensors>
4	<actuators>
5	<heaters>
6	<actuator id = "HEA0">
7	<state>NO</state>
8	</actuator>
9	. . .
10	</heaters>
11	<ventilators>
12	<actuator id = "VEN0">
13	<state>NO</state>
14	</actuator>
15	. . .
16	</ventilators>
17	<Alarms>
18	<actuator id = "ALA0">
19	<state>NO</state>
20	</actuator>
21	. . .
22	</Alarms>
23	</actuators>
24	</home>

The above conditions are described as logical expressions in the SCM-IoT model as follows:

$ update/Temperature = 'LOW' and
$ update/presence = 'YES' and
$ update/heater = 'NO'
$ update/fan = 'NO'

In previous expressions, "$ update" designates the variable that represents the last update of the notebook and that, therefore, contains the most recent information available on the temperature of the room, the presence of a resident in it, and the state of the heater or fan (air conditioning) installed in a room.

The above conditions are established as follows:

$ room/Temperature = 'VERY HIGH' and
$ room/smoke = 'YES' and
$ room/mexico expense = 'YES' and
$ Fire = 'NO'

The variable "$ room" contains a copy of the records recorded in the main book and consequently corresponds to the most recent readings reported by the sensors.

The presented case study describes the scope of SCM-IoT as established in the contributions provided by this model. The following is a more detailed description of the contributions:

1. The SCM-IoT model introduces a coordination mechanism where storage, production, and notification services are performed on the original data provided by the content producers. By providing these services, the editor may generate new data following application-defined rules. The editor coordinates participant activities through requests for direct modification of the table content (by update services) and subscriptions for relevant data notifications (by notify and collect services). The editor offers clients a mediated interaction driven by their interest in relevant data detection and notification. The centralization of data processing and pattern recognition may

simplify the design of IoT applications characterized by sensors and actuators with insufficient computational resources.

2. SCM-IoT is a table-oriented coordination language instead of tuple-oriented as in the Linda coordination model [20,21]. In SCM-IoT, tables, table entries, and entry fields are also called notebooks, pages, and topics, respectively. The table model establishes a fixed number of table entries, each sharing the same fixed number and type of fields. On the contrary, in the Linda model, tuples can have a distinct and arbitrary structure and even introduce new ones dynamically during execution. Nonetheless, the SCM-IoT restriction on the table structure can considerably improve the efficiency of the representation and processing of the data space used as a coordination medium. However, this does not limit the expressiveness of the model since several tables can be simultaneously allocated, each one with its structure and managed by its corresponding editor. The edition rules determine the type of editor that manages the table contents. Data production and subscription dependencies can form data flows among tables in more complex applications. Moreover, to increase the model efficiency, additional editors of the same type can be introduced for the same table as long as they do not interfere with each other acting on the same data.
3. As in Linda, in SCM-IoT the content of the data space is accessed and modified by a reduced set of primitive operations. However, by establishing the data space with a fixed structure, the SCM-IoT coordination primitives differ from the Linda primitives in that they do not use the availability principle as a coordination mechanism. In Linda, the out() operation makes available a new tuple in the data space, while the blocking in() operation waits for an available tuple matching the description specified by the operation. In comparison, in SCM-IoT the update() operation modifies the contents of an entry, while the notify() and collect() operations are the operations that synchronize the activities of the participants. However, the SCM-IoT coordination mechanism does not rely on the principle of availability since the table entries already exist and invariably remain there. Instead, coordination is attained when the information content of the entries satisfies some given condition. Thus, the blocking operation of Linda is replaced in SCM-IoT by subscriptions and notifications on the emergence of relevant data.
4. In SCM-IoT, the static nature of the table is not a conceptual limitation of the model but a non-functional constraint that ensures greater efficiency during coordination. A design based on fixed-structure tables may give applications greater clarity by allowing all logically related entries (i.e., those that follow the same editing rules) to be grouped under the same table. However, restructuring a table is possible and requires not updating table entries while performing a collection operation. As can be seen, this condition can still be fulfilled by blocking access until the collection is completed. Once completed, constructive reconfiguration of the table that includes additional fields or entries has no consequences on the suspended applications and can be resumed afterward. It depends entirely on the specification of new applications that the addition of new fields or entries in the table does not interfere with the assumptions on which previous applications were designed. On the other hand, destructive restructuring (deleting existing and in-use fields or entries) must be studied on the basis that each application requires.

According to [20], the SCM-IoT coordination model is orthogonal to the computational model. Decoupled interaction is due to the use of primitive operations that allow synchronized access to the contents of the coordination medium. A table-structured coordination medium can be represented as a bidimensional array or an XML DOM structure in most programming languages.

Finally, the design of the SCM-IoT primitive operations could be described in terms of design patterns [83]. In addition to the observer (publisher–subscriber) pattern, other patterns have also been used, such as:

a. Mediator (reduces the complexity of the many direct communication among producers and subscribers to only one with the editor);
b. State (the editor modifies its behavior in response to the data pattern detected in the table, including the notification to subscribers);
c. Iterator (the collect() operation is an iterator over each entry of the table);
d. Interpreter (the dynamic definition of the logical conditions, including relationals (EQUAL, LIKE, BETWEEN) and connectives (AND, OR, NOT, IMPLIES) that describe the relevant data patterns can be implemented to evaluate at runtime the logical conditions given in the application rules);
e. Command (the definition of the basic operations that act upon the actuators).

These design patterns can be used to build an implementation that is easier to build, test, modify, and reuse.

5. Discussion

Regarding the computational models described, even though both models, CPN and BPEL, are widely accepted and used to design and construct interoperable distributed systems, there are still severe problems to be solved to satisfy the needs demanded by IoT systems. Among the most critical problems that have been identified in the development of IoT systems are the lack of dynamism, limited expressiveness, and the lack of continuity in the development process.

5.1. Lack of Dynamism

CPNs are an abstract computational model that allows describing and studying the relationships between parts of a system characterized by their concurrent, asynchronous, distributed, parallel, non-deterministic, and stochastic properties. For the construction of IoT systems, CPNs support tools for modeling, visualization, analysis, validation, and verification of behaviors within a system, as they provide: hierarchical descriptions, interactive simulations, and an intuitive and attractive graphic representation. Although the abstract model in CPN allows describing the expected functionality of each service, they also have severe limitations due to the lack of sound dynamics. An example of these limitations is its inability to include logical conditions in service requests correctly interpreted at simulation time, such as passing the condition "$p <> p'$" as a parameter in an update request. The reason is that the variables "p" and "p'" that appear in this expression do not have a denotation as parameters in the update transition U (Figure 2), since it depends entirely on the names of the variables used in the annotations corresponding entry arches. In other words, there is no way to denote a transition in CPN as a strictly evaluated function that can return zero, one, or more output values. In such denotation, the function's formal parameters may correspond to the annotations on the input arcs, and the values evaluated by the function may correspond to the annotations on the output arcs. Unfortunately, the CPNs do not impose order in the entry (and exit) places and arches. Although modern programming languages use keywords to designate formal parameters directly, CPNs do not use keywords either in the input arcs or in the input places. Finally, there is still the problem of defining an appropriate semantics for this mechanism that combines functions without side effects in the set of tokens of the CPNs. In summary, CPNs limit the expressive capacity in the dynamism of the abstract model that IoT systems require, where interoperable components can be integrated with service demands under arbitrary conditions. On the other hand, BPEL exhibits the same lack of dynamism that prevents accepting requests that include conditions of interest during the subscription. The limitations in both models impose severe limitations on the dynamism in the provision of services demanded by the IoT.

A solution to solve the lack of dynamism in the provision of services lies in the possibility of compiling or dynamically interpreting the expressions included in the service provision rules. From the relational database model perspective, the SCM-IoT model notebooks can be implemented as tables and the editing services as SQL data manipulation

and query language commands. Still, SQL expressions must be compiled before being executed, which could be done dynamically after subscription. However, the problem with this possible solution is that it demands more computational infrastructure for the component where the database administrator resides.

However, the SCM-IoT model does not limit the dynamism to establish an interaction structure between clients and editors, making it possible to discover an editor that offers the most convenient publishing services for the client's applications. In order to find an editor offering the required services, it is possible to start by discovering the most suitable table structure for the client's application. For this purpose, the ANTARES algorithm [84] can be used to discover the table with the topics required by the application and then identify the corresponding editor. Following this approach, a natural language processing algorithm [85] can be used to build an overlay network among clients (information producers and subscribers). Advanced selection and discovery information services can also be offered in a virtual structure built on an initially unstructured environment. These algorithms provide the means to introduce service discovery improving dynamism into the SCM-IoT model as future work.

5.2. Limitations

The SCM-IoT model has the following limitations:

- The model is constrained to applications with a fixed interaction structure among producers and subscribers of information. Further study is needed to address the problem of fast-changing interaction structures;
- The model is oriented to IoT systems with low message traffic, low speed, and moderate latency. The complexity of the rules can be high, although it is limited by the storage space available to represent the appropriate size of the notebooks;
- The model proposes an architecture with a broker that provides storage services, analysis, and coordination between clients. The service architecture uses SOA interoperability standards for the composition of web services;
- The participant interaction rules with the storage medium should be designed to reduce competition for shared data so that they are mutually exclusive in updates and notifications that do not include co-readings. This limits the applications as the SCM-IoT model lacks tools to analyze conflicts between the interaction rules. Although tools such as CPN tools can help remedy this limitation, it is necessary to continue studying the properties of the representation of the SCM-IoT model with CPN;
- The application of the rules must always end, leading the environment to a stable state where no further changes are observed. Otherwise, the medium would necessarily have to be modified by the application without terminating at least one rule. For this reason, the SCM-IoT model excludes some applications such as complex event processing [86] that process potentially infinite flows of events;
- The platform-dependent computational model reveals considerable inefficiency for database CRUD operations implemented on XML and XSL The inefficiency is because the entire XML document is almost entirely copied except for the updated field. Fortunately, XML document databases [87] offer a more efficient alternative in implementing CRUD operations.

6. Conclusions

SCM-IoT is a model for developing applications in the context of IoT that has a logically centralized data-oriented architecture, which allows addressing artificial intelligence and automated learning applications. These applications involve the collaboration of intelligent agents that share a storage medium to exchange information and build solutions. Among the advantages that the SCM-IoT model provides are the following:

- The simplicity and flexibility of the mechanism for integrating new participants;
- The decoupling between the activities of the participants, which gives them a high degree of independence and autonomy;

- The distinction between two types of clients according to the storage and publishing services offered by the editor acting as a broker;
- The flexibility of the proposed producer–editor–subscriber interaction pattern allows distinguishing clients' roles according to the services provided by the publisher and integrating them as interoperable components.

Although the SCM-IoT model is similar to the active database model, there are substantial differences that are described below in the form of advantages:

- The tables used in IoT applications are small, so the size, complexity, and cost of their administration that they demand is inconvenient;
- DBs also suffer from a lack of dynamism to accept queries dynamically formulated by subscribers and incorporate new processing rules;
- The technology used is built on the W3C XML standard;
- Available tools such as XPath and XSLT provide enormous flexibility for modeling complex data that is difficult to define in the standard relational database model;
- For the representation of knowledge and automated reasoning, XML technology provides a more suitable way to develop artificial intelligence applications based on RuleML [88].

Finally, future work intends to increase the expressiveness of the service composition mechanism in the IoT to include conditions and actions in service requests. For this purpose, the XML databases for the notebooks' administration will be investigated to determine if the computational infrastructure they require conforms to the profile that IoT applications typically demand.

Likewise, due to the capacity of integration and application to the environment of the SCM-IoT model, it is considered to extrapolate it to other application domains such as industrial and transportation and logistics, since in these domains there are also composition problems of services although with other particular characteristics that have to be considered. Finally, it is intended to incorporate new intelligent devices that have the new technical characteristics that are increasing according to the new challenges and challenges to be addressed in the development of solutions in the context of the IoT.

Author Contributions: Conceptualization, I.M.-C., J.O.O.-A. and G.A.-H.; Data curation, J.O.O.-A.; Formal analysis, L.R.-M. and A.L.-C.; Funding acquisition, G.A.-H., L.R.-M. and J.L.S.-C.; Investigation, J.L.S.-C.; Methodology, I.M.-C. and J.O.O.-A.; Project administration, G.A.-H.; Resources, G.A.-H.; Software, I.M.-C., J.O.O.-A. and L.R.-M.; Supervision, L.R.-M. and A.L.-C.; Validation, I.M.-C., G.A.-H. and L.R.-M.; Visualization, J.L.S.-C. and A.L.-C.; Writing—original draft, I.M.-C. and J.O.O.-A.; Writing—review and editing, J.O.O.-A. and G.A.-H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Mexico's National Council of Science and Technology (CONACYT) and the Public Secretariat of Education (SEP) through the Sectorial Fund of Research for Education, grant number A1-S-51808 and the project 52–2016: "Application of Big Data and Semantic Web techniques to Develop Intelligent Systems", a postdoctoral grant, and a doctoral grant.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy concerns of study.

Acknowledgments: This work was supported by Mexico's National Technological Institute (TecNM) and sponsored by both Mexico's National Council of Science and Technology (CONACYT) and the Secretariat of Public Education (SEP) through the PRODEP project (Programa para el Desarrollo Profesional Docente).

Conflicts of Interest: The authors declare that there are no potential conflicts of interest with respect to the publication of this article.

References

1. Atzori, L.; Iera, A.; Morabito, G. The Internet of Things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805. [CrossRef]
2. Li, S.; Xu, L.D.; Zhao, S. The Internet of Things: A Survey. *Inf. Syst. Front.* **2015**, *17*, 243–259. [CrossRef]
3. Kortuem, G.; Kawsar, F.; Sundramoorthy, V.; Fitton, D. Smart Objects as Building Blocks for the Internet of Things. *IEEE Internet Comput.* **2010**, *14*, 44–51. [CrossRef]
4. Welbourne, E.; Battle, L.; Cole, G.; Gould, K.; Rector, K.; Raymer, S.; Balazinska, M.; Borriello, G. Building the Internet of Things Using RFID. *IEEE Internet Comput.* **2009**, *13*, 48–55. [CrossRef]
5. Bandyopadhyay, D.; Sen, J. Internet of Things—Applications and Challenges in Technology and Standardization. *Wirel. Pers. Commun.* **2011**, *58*, 49–69. [CrossRef]
6. Botta, A.; De Donato, W.; Persico, V.; Pescape, A. Integration of Cloud Computing and Internet of Things: A Survey. *J. Future Gener. Comput. Syst.* **2015**, *56*, 684–700. [CrossRef]
7. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog Computing and Its Role in the Internet of Things. In Proceedings of the First Edition of the MCC Worshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012; pp. 13–16. [CrossRef]
8. Luthra, S.; Garg, D.; Mangla, S.K.; Berwal, Y.P.S. Analizing challenges to Internet of Things (IoT) adoption and diffusion: An Indian context. *Proc. Comput. Sci.* **2018**, *15*, 733–739. [CrossRef]
9. Tokognon, C.A.; Gao, B.; Tian, G.Y.; Yan, Y. Structural Health Monitoring Framework Based on Internet of Things: A Survey. *IEEE Internet Things J.* **2017**, *4*, 619–635. [CrossRef]
10. Pisching, M.A.; Junquiera, F.; Santos Filho, D.J.; Miyagi, P.E. Service Composition in the Cloud-Based Manufacturing Focused on the Industry 4.0. *Technol. Innov. Cloud-Based Eng. Syst. DoCEIS 2015 IFIP Adv. Inf. Commun. Technol.* **2015**, *450*, 65–72. [CrossRef]
11. Yang, Z.; Li, D. IoT Information Service Composition Driven by User Requirement. In Proceedings of the IEEE 17th International Conference on Computational Science and Engineering, Chengdu, China, 19–21 December 2014; pp. 509–513. [CrossRef]
12. Macker, J.P.; Taylor, I. Orchestration and analysis of decentralized workflows within heterogeneous networking infrastructures. *Future Gener. Comput. Syst.* **2017**, *75*, 388–401. [CrossRef]
13. Chen, L.; Englund, C. Choreographing services for smart cities: Smart traffic demonstration. In Proceedings of the IEEE 85th Vehicular Technology Conference (VTC Spring), Sydney, NSW, Australia, 4–7 June 2017; pp. 1–5. [CrossRef]
14. Han, S.N.; Khan, I.; Lee, G.M.; Crespi, N.; Glitho, R.H. Service composition for IP smart object using real-time Web protocols: Concept and research challenges. *Comput. Stand. Interfaces* **2016**, *43*, 79–90. [CrossRef]
15. Mathew, J.; John, J.; Kumar, S. New trends in healthcare supply chain. International Annual Conference, Production and Operations Management Society. 2013, pp. 1–10. Available online: https://www.pomsmeetings.org/confpapers/043/043-0259.pdf (accessed on 24 February 2022).
16. De Vries, J.; Huijsman, R. Supply chain management in health services: An overview. *Supply Chain. Manag. Int. J.* **2011**, *16*, 159–165. [CrossRef]
17. Chacon-Troya, D.P.; Gonzalez, O.O.; Campoverde, P.C. Domotic application for the monitoring and control of residential electricalloads. In Proceedings of the 2017 IEEE 37th Central America and Panama Convention (CONCAPAN XXXVII), Managua, Nicaragua, 15–17 November 2017; pp. 1–6. [CrossRef]
18. Machorro-Cano, I.; Alor-Hernández, G.; Paredes-Valverde, M.A.; Ramos-Deonati, U.; Sánchez-Cervantes, J.L.; Rodríguez-Mazahua, L. PISIoT: A Machine Learning and IoT-Based Smart Health Platform for Overweight and Obesity Control. *Appl. Sci.* **2019**, *9*, 3037. [CrossRef]
19. Machorro-Cano, I.; Alor-Hernández, G.; Paredes-Valverde, M.A.; Rodríguez-Mazahua, L.; Sánchez-Cervantes, J.L.; Olmedo-Aguirre, J.O. HEMS-IoT: A Big Data and Machine Learning-Based Smart Home System for Energy Saving. *Energies* **2020**, *13*, 1097. [CrossRef]
20. Carriero, N.J.; Gelernter, D.; Matson, T.G.; Sherman, A.H. The Linda Alternative to message-passing systems. *Parallel Comput.* **1994**, *2*, 633–655. [CrossRef]
21. Ahuja, S.; Carriero, N.; Gelernter, D. Linda and Friends. *Comput. IEEE* **1986**, *19b*, 26–34. [CrossRef]
22. Sun, M.; Zhou, Z.; Wang, J.; Du, C.; Gaaloul, W. Energy-Efficient IoT Service Composition for Concurrent Timed Applications. *Future Gener. Comput. Syst.* **2019**, *100*, 1017–1030. [CrossRef]
23. Rapti, E.; Karageorgos, A.; Gerogiannis, V.C. Decentralised Service Composition using Potential Fields in Internet of Things Applications. *Proc. Comput. Sci.* **2015**, *52*, 700–706. [CrossRef]
24. Pang, Z.; Zheng, L.; Tian, J.; Kao-Walter, S.; Dubrova, E.; Chen, Q. Design of a terminal solution for integration of in-home health care devices and services towards the Internet-of-Things. *Enterp. Inf. Syst.* **2013**, *9*, 86–116. [CrossRef]
25. Swiatek, P. ComSS—Platform for Composition and Execution of Streams Processing Services. In *Intelligent Information and Database Systems, Lecture Notes in Computer Science 2015*; Springer: Cham, Switzerland, 2015; Volume 9012, pp. 494–505. [CrossRef]
26. Dijkman, R.M.; Sprenkels, B.; Peeters, T.; Janssen, A. Business models for the Internet of Things. *Int. J. Inf. Manag.* **2015**, *35*, 672–678. [CrossRef]
27. Shehu, U.G.; Safdar, G.A.; Epiphaniou, G. Network-aware Composition for Internet of Thing Services. *Trans. Netw. Commun.* **2015**, *3*, 45–48. [CrossRef]
28. Vidyasankar, K. A Transaction Model for Executions of Compositions of Internet of Things Services. *Proc. Comput. Sci.* **2016**, *83*, 195–202. [CrossRef]

29. Gierej, S. The framework of business model in the context of Industrial Internet of Things. *Proc. Eng.* **2017**, *182*, 206–212. [CrossRef]
30. Ju, J.; Kim, M.S.; Ahn, J.H. Prototyping Business Models for IoT Service. *Proc. Comput. Sci.* **2016**, *91*, 882–890. [CrossRef]
31. Urbieta, A.; González-Beltrán, A.; Mokhtar, S.B.; Hossain, M.A.; Capra, L. Adaptive and context-aware service composition for IoT-based smart cities. *Future Gener. Comput. Syst.* **2017**, *76*, 262–274. [CrossRef]
32. Salle, A.D.; Gallo, F.; Perucci, A. Dependable Composition of Software and Services in the Internet of Things: A Biological Approach. In *Software Engineering and Formal Methods, Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2016; Volume 9509, pp. 312–323. [CrossRef]
33. Baker, T.; Asim, M.; Tawfik, H.; Aldawsari, B.; Buyya, R. An energy-aware service composition algorithm for multiple cloud-based IoT applications. *J. Netw. Comput. Appl.* **2017**, *89*, 96–108. [CrossRef]
34. Yamaoka, H.; Itakura, K.; Takahashi, E.; Nakagawa, G.; Michaelis, J.; Kanemasa, Y.; Ueki, M.; Matsumoto, T.; Take, R.; Tanie, S.; et al. Dracena: A Real-Time IoT Service Platform Based on Flexible Composition of Data Streams. In Proceedings of the IEEE/SICE International Symposium on System Integration, Paris, France, 14–16 January 2019; pp. 596–601. [CrossRef]
35. Krishna, A.; Pallec, M.L.; Mateescu, R.; Noirie, L.; Salaun, G. IoT Composer: Composition and Deployment of IoT Applications. In Proceedings of the IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Montreal, QC, Canada, 25–31 May 2019; pp. 19–22. [CrossRef]
36. Ridhawi, I.A.; Aloqaily, M.; Boukerche, A.; Jaraweh, Y. A Blockchain-Based Decentralized Composition Solution for IoT Services. In Proceedings of the ICC 2020–2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–6.
37. Lakhdari, A.; Bouguettaya, A.; Mistry, S.; Azadeh, G.N. Composing Energy Services in a Crowdsourced IoT Environment. In Proceedings of the IEEE Transactions on Services Computing, Sydney, NSW, Australia, 22–24 April 2020; pp. 1–14. [CrossRef]
38. Arellanes, D.; Lau, K.-K. Evaluating IoT service composition mechanisms for the scalability of IoT systems. *Future Gener. Comput. Syst.* **2020**, *108*, 827–848. [CrossRef]
39. Abusafia, A.; Bouguettaya, A.; Mistry, S. Incentive-Based Selection and Composition of IoT Energy Services. In Proceedings of the IEEE International Conference on Services Computing (SCC), Beijing, China, 18–24 October 2020; pp. 304–311. [CrossRef]
40. Cano, J.; Rutten, E.; Delaval, G.; Benazzouz, Y.; Gurgen, L. ECA Rules for IoT Environment: A Case Study in Safe Design. In Proceedings of the IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems Workshops, London, UK, 8–12 September 2014; pp. 116–121. [CrossRef]
41. Giang, N.K.; Lea, R.; Blackstock, M.; Leung, V. On building smart city IoT applications: A coordination-based perspective. In Proceedings of the 2nd International Workshop on Smart; ACM: New York, NY, USA, 2016; p. 7. [CrossRef]
42. Cheng, B.; Zhu, D.; Zhao, S.; Chen, J. Situation-aware iot service coordination using the event-driven soa paradigm. *IEEE Trans. Netw. Serv. Manag.* **2016**, *13*, 349–361. [CrossRef]
43. Belkeziz, R.; Jarir, Z. IoT Coordination: Designing a context-driven architecture. In Proceedings of the International Conference on Signal-Image Technology & Internet-Based Systems (SITIS), Jaipur, India, 4–7 December 2017; pp. 388–395. [CrossRef]
44. García-Magariño, I.; Gray, G.; Muttukrishnan, R.; Asif, W. Agent-based IoT Coordination for Smart Cities Considering Security and Privacy. In Proceedings of the 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Granada, Spain, 22–25 October 2019; pp. 221–226. [CrossRef]
45. Belkeziz, R.; Jarir, Z. An Overview of the IoT Coordination Challenge. *Int. J. Serv. Sci. Manag. Eng. Technol. IJSSMET* **2020**, *11*, 99–115. [CrossRef]
46. Cubo, J.; Nieto, A.; Pimentel, E. A Cloud-Based Internet of Things Platform for Ambient Assisted Living. *Sensors* **2014**, *14*, 14070–14105. [CrossRef]
47. Qu, C.; Liu, F.; Tao, M.; Deng, D. An OWL-S Based Specification Model of Dynamic Entity Services for Internet of Things. *J. Ambient. Intell. Humaniz. Comput.* **2016**, *7*, 73–82. [CrossRef]
48. Yu, J.; Bang, H.C.; Lee, H.; Lee, Y.S. Adaptive Internet of Things and Web of Things convergence platform for Internet of reality services. *J. Supercomput.* **2016**, *72*, 84–102. [CrossRef]
49. Bergesio, L.; Bernardos, A.M.; Casar, J.R. An Object-Oriented Model for Object Orchestration in Smart Environments. *Proc. Comput. Sci.* **2017**, *109C*, 440–447. [CrossRef]
50. Wen, Z.; Yang, R.; Garraghan, P.; Lin, T.; Xu, J.; Rovatsos, M. Fog orchestration for internet of things services. *IEEE Internet Comput.* **2017**, *21*, 16–24. [CrossRef]
51. Ren, M.; Ren, L.; Jain, H. Manufacturing service composition model based on synergy effect: Asocial network analysis approach. *Appl. Soft Comput.* **2018**, *70*, 288–300. [CrossRef]
52. Pahl, C.; El Ioini, N.; Helmer, S.; Lee, B. An architecture pattern for trusted orchestration in IoT edge clouds. In Proceedings of the 2018 Third International Conference on Fog and Mobile Edge Computing (FMEC), Barcelona, Spain, 23–26 April 2018; pp. 63–70. [CrossRef]
53. Malik, S.; Ahmad, S.; Kim, D. A Novel Approach of IoT Services Orchestration Based on Multiple Sensor and Actuator Platforms Using Virtual Objects in Online IoT App-Store. *Sustainability* **2019**, *11*, 5859. [CrossRef]
54. Ren, W.; Sun, Y.; Luo, H.; Obaidat, M.S. A New Scheme for IoT Service Function Chains Orchestration in SDN-IoT Network Systems. *IEEE Syst. J.* **2020**, *13*, 4081–4092. [CrossRef]

55. Rafique, W.; Zhao, X.; Yu, S.; Yaqoob, I.; Imran, M.; Dou, W. An Application Development Framework for Internet-of-Things Service Orchestration. *IEEE Internet Things J.* **2020**, *7*, 4543–4556. [CrossRef]
56. Serhani, M.A.; El-Kassabi, H.T.; Shuaib, K.; Navaz, A.N.; Benatallah, B.; Beheshti, A. Self-adapting cloud services orchestration for fulfilling intensive sensory data-driven IoT workflows. *Future Gener. Comput. Syst.* **2020**, *108*, 583–597. [CrossRef]
57. Rodríguez-Valenzuela, S.; Holgado-Terriza, J.A.; Gutiérrez-Guerrero, J.M.; Muros-Cobos, J.L. Distributed Service-Based Approach for Sensor Data Fusion in IoT Environments. *Sensors* **2014**, *14*, 19200–19228. [CrossRef]
58. Dar, K.; Taherkordi, A.; Baraki, H.; Eliassen, F.; Geihs, K. A resource oriented integration architecture for the Internet of Things: A business process perspective. *Pervasive Mob. Comput.* **2015**, *20*, 145–159. [CrossRef]
59. Duhart, C.; Sauvage, P.; Bertelle, C. A Resource Oriented Framework for Ser-vice Choreography over Wireless Sensor and Actor Networks. *Int. J. Wirel. Inf. Netw.* **2016**, *23*, 173–186. [CrossRef]
60. Cherrier, S.; Ghamri-Doudane, Y.; Lohier, S.; Roussel, G. D-LITe: Building Internet of Things Choreographies. *arXiv* **2016**, arXiv:1612.05975.
61. Blanc, S.; Bayo-Monton, J.L.; Campelo, J.C.; Fernandez-Llatas, C. Process Choreography in Wireless Sensor and Actuator Networks: A proposal to achieve Network Virtualization. *Int. J. Actor-Netw. Theory Technol. Innov.* **2016**, 1–11.
62. Montali, M.; Plebani, P. IoT-based Compliance Checking of Multi-partyBusiness Processes modeled with Commitments. In *European Conference on Service-Oriented and Cloud Computing*; Springer: Cham, Switzerland, 2017; Volume 10465, pp. 179–195. [CrossRef]
63. Seeger, J.; Deshmukh, R.A.; Bröring, A. Running Distributed and Dynamic IoT Choreographies. In Proceedings of the 2018 IEEE Global Internet of Things Summit (GIoTS) Proceedings, Bilbao, Spain, 4–7 June 2018; Volume 2, pp. 33–38. [CrossRef]
64. Singhal, N.; Sakthivel, U.; Raj, P. Selection Mechanism of Micro-Services Orchestration vs. Choreography. *Int. J. Web Semant. Technol. IJWesT* **2019**, *10*, 25. [CrossRef]
65. Arreaga, N.X.; Blanc, S.; Rivas, L.V.; Palanca, S. Implementation of choreography services for precision agriculture based on real-time monitoring and control system using WSN. *EURASIP J. Wirel. Commun. Netw.* **2021**, *1*, 1–20. [CrossRef]
66. Ding, Z.; Yang, R. Modeling and Analysis for Mobile Computing Systems Based on Petri Nets: A Survey. *IEEE Access* **2018**, *6*, 68038–68056. [CrossRef]
67. Llorens, M.; Oliver, J. Structural and dynamic changes in concurrent systems: Reconfigurable Petri nets. *IEEE Trans. Comput.* **2004**, *53*, 1147–1158. [CrossRef]
68. Lamport, L. Turing Lecture: The Computer Science of Concurrency: The Early Years. *Commun. ACM* **2015**, *58*, 71–76. [CrossRef]
69. Nielsen, M.; Sassone, V.; Winskel, G. Relationships between models of concurrency. In *A Decade of Concurrency Reflections and Perspectives. REX 1993. Lecture Notes in Computer Science*; De Bakker, J.W., de Roever, W.P., Rozenberg, G., Eds.; Springer: Berlin/Heidelberg, Germany, 1994; Volume 803, pp. 425–476. [CrossRef]
70. Lee, E.A.; Sangiovanni-Vincentelli, A. A Framework for Comparing Models of Computation. *IEEE Trans. Comput. Aided Des. Circuits Syst.* **1998**, *17*, 1217–1229. [CrossRef]
71. Castellano, L.; De Michelis, G.; Pomello, L. Concurrency vs interleaving: An instructive example. *Bull. EATCS* **1987**, *31*, 12–15.
72. Milner, R. *A Calculus of Communicating Systems, Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 1982; Volume 92, pp. 11–26.
73. Hoare, C.A.R. Communicating sequential processes. *Commun. ACM* **1978**, *21*, 666–677. [CrossRef]
74. Ojo, K.; González, Y.; Cano, E.E.; Rovetto, C.A. Modelado del funcionamiento de un dispositivo para el control de la asistencia estudiantil mediante Redes de Petri Coloreadas. II Congreso Internacional en Inteligencia Ambiental, Ingeniería de Software y Salud Electrónica y Móvil–AmITIC 201, Chiriqui, Panama, 12–14 September 2018; 1, pp. 13–20.
75. Zhou, J.; Reniers, G. A Petri-net based simulation analysis approach for cascading effect of vapor cloud explosions. *J. Loss Prev. Process Ind.* **2017**, *48*, 118–125. [CrossRef]
76. Corradini, A. Concurrent computing: From Petri nets to graph grammars. *Electron. Theor. Comput. Sci.* **1995**, *2*, 56–70. [CrossRef]
77. Milner, R. *Communicating and Mobile Systems: The Pi Calculus*, 1st ed.; Cambridge University Press: New York, NY, USA, 1999; pp. 1–174.
78. Jensen, K.; Kristensen, L.M. Colored petri nets: A graphical language for formal modeling and validation of concurrent systems. *Commun. ACM* **2015**, *58*, 61–70. [CrossRef]
79. Jensen, K.; Kristensen, L.; Wells, L. Coloured petri nets and CPN tools for modelling and validation of concurrent systems. *Int. J. Softw. Tools Technol. Transf.* **2007**, *9*, 213–254. [CrossRef]
80. Tan, C.W.; Goh, A. Implementing ECA rules in an active database. *Knowl.-Based Syst.* **1999**, *12*, 137–144. [CrossRef]
81. Dittrich, K.R.; Gatziu, S.; Geppert, A. *The Active Database Management System Manifesto: A Rulebase of ADBMS Features. Rules in Database Systems. RIDS 1995. Lecture Notes in Computer Science*; Sellis, T., Ed.; Springer: Berlin/Heidelberg, Germany, 1995; Volume 985, pp. 3–20. [CrossRef]
82. Forgy, C. On the Efficient Implementation of Production Systems. Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, MA, USA, 1979.
83. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*; Addison-Wesley Professional: Boston, MA, USA, 1994.

84. Forestiero, A.; Mastroianni, C.; Spezzano, G. Antares: An ant-inspired P2P information system for a self-structured grid. In Proceedings of the 2nd Bio-Inspired Models of Network, Information and Computing Systems, Budapest, Hungary, 10–12 December 2007; pp. 151–158. [CrossRef]
85. Forestiero, A.; Papuzzo, G. Agents-based algorithm for a distributed information system in Internet of Things. *IEEE Internet Things J.* **2021**, *8*, 16548–16558. [CrossRef]
86. Luckham, D. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, 1st ed.; Addison-Wesley Professional: Boston, MA, USA, 2002; p. 376.
87. Pavlovic-Lazetic, G. Native XML databases vs. relational databases in dealing with XML documents. *Kragujev. J. Math* **2007**, *30*, 181–199.
88. RuleML Home. Available online: http//www.ruleml.org (accessed on 8 November 2021).