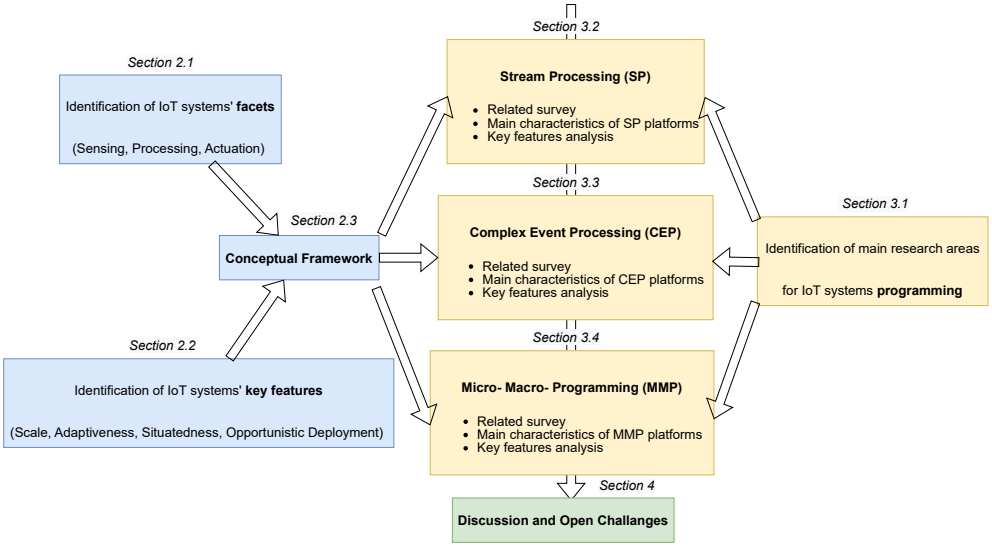
Graphical Abstract

Programming IoT Systems: a Focused Conceptual Framework and Survey of Approaches



Highlights

Programming IoT Systems: a Focused Conceptual Framework and Survey of Approaches

- a conceptual framework for analysing IoT programming approaches is provided
- focus on scale independence, situatedness, adaptiveness, and opportunistic deployment
- state-of-the-art stream, event, micro and macro programming approaches are surveyed
- from the analysis of the state of the art, open challenges are identified

Programming IoT Systems: a Focused Conceptual Framework and Survey of Approaches

Abstract

Any software engineer of Internet of Things (IoT) systems deals with three macro issues: how to perceive the properties of interest through sensors (*sensing* facet), how to process such information to decide what to do to achieve the system goals (*processing* facet), and how to enact such decisions by affecting the IoT system itself and its deployment environment accordingly (*actuation* facet). For each of these, one can either develop ad-hoc solutions from scratch, by relying on mainstream programming languages, or build on top of existing IoT-specific software libraries, frameworks, and platforms. In this paper, we survey the broad state of the art of "IoT programming", with a focus on clarifying which and how *programming paradigms and platforms* deal with four key features demanded by modern IoT systems: *scale-independence*, *situatedness*, *adaptiveness*, and *opportunistic deployment*, along the aforementioned three facets. We motivate such needs by describing compelling contemporary and near future scenarios. Then, we propose a reference *conceptual framework of programming IoT systems* with the goal of *(i)* uncovering which research areas are mostly active in IoT programming, and *(ii)* placing the state of the art at the intersection between the appropriate features and facets, to both *(iii)* clarify which approaches are most suited for different kinds of tasks, and *(iv)* emphasising open challenges. This conceptual framework is a novel contribution in the landscape of IoT programming surveys, and is intended to be a practical aid for researchers and practitioners that are deciding which computational tools (e.g. languages and platforms) to adopt while building their own IoT systems.

Keywords: Internet of Things, IoT programming, stream processing, complex event processing, micro programming, macro programming, survey

1. Introduction

Internet of Things (IoT) programming is the practice of building software for IoT systems and applications, spanning many domains such as intelligent transportation systems, smart buildings and smart cities, industry 4.0, and more [163, 67]. Essentially, IoT programming spans any scenario in which a multitude of *heterogeneous* devices *scattered* across a geographical area are required to collaborate to deliver some *global* functionality tightly depending on some *local* perception / processing. Different approaches to IoT programming exist, often combining software libraries, frameworks, and platforms in an ad-hoc way. This generates confusion about what the best fit is and how to integrate their usage at best [153, 34].

Heterogeneity of devices, their distribution, and the "local-to-global" connection [22] make IoT programming at scale an intrinsically complex task [99, 30]. It generally requires to be dealt with along three fundamental *facets*: (i) how to manage perception of the properties of interest from the IoT environment through sensors (*sensing* facet), (ii) how to process such information for decision-making to achieve the system goals (*processing* facet), and (iii) how to enact such decisions by affecting the environment through actuators (*actuation* facet). Accordingly, IoT programming research spans the whole software stack: languages and libraries for programming devices, platforms supporting data streaming and event processing, frameworks for distributed computing and service deployment. Also, modern IoT applications are increasingly *(i) large-scale* systems performing *(ii) situated* computations while taking advantage of an *(iii) opportunistic deployment* model, where both the application functionalities and the middleware services enabling them are able to *(iv) adapt* to contextual factors such as the location where relevant events take place. Hence, they demand four key *features* – motivated in Section 2.3 – against which the available literature ought to be evaluated.

Many existing surveys do a good job in identifying the most active research trends, categorising the driving application domains, and emphasising open issues and opportunities for further research. However, they tend to focus on a particular approach (e.g., stream processing but not macroprogramming), infrastructural layer (e.g., the cloud but not the Edge, platforms but not languages), and/or feature set (e.g., opportunistic deployment but not situatedness), failing to clarify how the different approaches fit the overall picture—see Table 1. Hence, the IoT programming research landscape

Table 1: Other surveys related to "IoT programming", with main differences with respect to this one.

Ref.	Brief description	Year
[41]	It is limited to a few dimensions of analysis (i.e. communication protocols, data processing, data visualisation, integration, security).	2022
[128]	It is limited to agent-based computing paradigm.	2020
[145]	It focuses on application development, and is limited to OSs, networking gateways, Cloud-hosted platforms, and mainstream programming languages.	2019
[84]	Its topic is Fog programming, and overlaps with IoT programming; however, few Cloud-centric platforms are covered.	2019
[68]	It is limited to OSs and their features (e.g. networking, real-time, communication protocols, etc.).	2018
[104]	It is limited to IoT middlewares, which are analysed by an architectural standpoint.	2017
[136]	It is limited to hardware platforms and their basic software (firmware and OS).	2017
[118]	It is limited to IoT middlewares, analysed from the standpoint of application and deployment (architectural) requirements.	2016
[111]	It adopts a market perspective, covering only commercial products, and analysing them from a user-centric perspective (i.e., not a developer perspective).	2015
[53]	It focuses on information processing at the low level and, in particular, on signal processing, to build higher-level abstractions.	2015

deserves an integrative analysis.

For these reasons, here we provide a comprehensive review of the literature on "IoT programming", while trying to avoid overlap with the already existing surveys as much as possible. In particular, we clarify the state-of-the-art by positioning contributions within a novel conceptual framework we conceive, presented in Section 2, also useful to identify gaps in current research efforts. In particular, we aim to address the following questions: *(i)* what are the research areas contributing the most to IoT programming? *(ii)* what is the level of support provided by state-of-the-art approaches regarding the aforementioned key features (scale-independence, situatedness, adaptiveness, and opportunistic deployment)? *(iii)* how do they cover the sensing, processing, and actuation facets? *(iv)* what are the challenges yet to be dealt with? Accordingly, the contribution of this paper is manyfold:

- we define a conceptual framework to organise the literature about pro-

gramming IoT systems according to *(i)* four key features of modern IoT systems that need to be supported and *(ii)* the three facets of computation involved;

- we uncover the research areas mostly involved in IoT programming, such as Stream Processing (SP), Complex Event Processing (CEP), and micro/macro-programming, and survey their state of the art;
- we categorise approaches in the identified research areas within the defined conceptual framework;
- we discuss open challenges stimulating further research in related areas.

The remainder of the paper is thus organised as follows (and as depicted by Figure 1, so that readers interested in a particular macro-area or facet may directly skim to the desired section). Section 2 describes the conceptual framework. Section 3 presents the survey itself and the research methods used to perform it, organised according to the macro-areas of research that emerged, and focussing on the four key features identified. Section 4 discusses the open challenges yet to be addressed. Finally, Section 5 wraps the paper up with conclusive remarks and an outlook to further research.

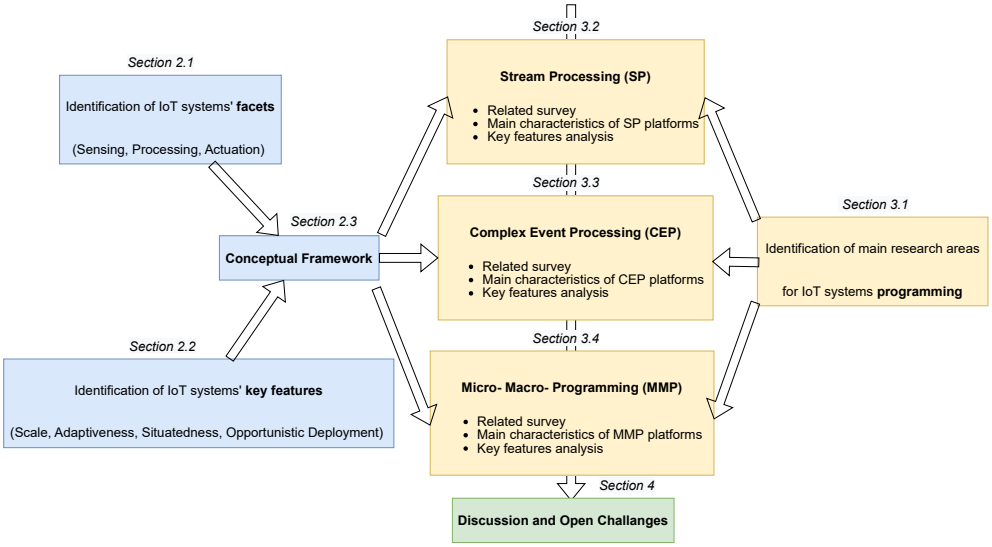


Figure 1: Graphical abstract. Contents of papers' main sections and their relationships.

2. Conceptual framework

We here define the three *facets of computation* typically involved in IoT programming (sensing, processing, actuation), and the four *key features* that modern IoT systems cannot overlook (scale-independence, situatedness, adaptiveness, and opportunistic deployment), motivating their relevance in already existing systems and in others near to come.

By placing the state-of-the-art research approaches at the various "crossroads" among the aforementioned facets and key features (Figure 2), we can provide a bird-eye view of the research landscape. This will help to clarify the role of each approach in programming IoT systems and highlight which functional facets and key features are covered the most (or the least, suggesting gaps for further research).

2.1. Facets of computation

Any IoT system has three distinct but deeply intertwined facets of computation:

Sensing facet. It is the facet concerned with how to *perceive* the "state of the world" according to the application goals—hence the upstream arrow in Figure 2. Such facet may encompass a whole stack of functions and levels of abstraction: gathering measured data from sensor devices, (pre)processing such data to devise out more complex situations [114], and possibly even putting such situations into causal relations [115].

Processing facet. It is the facet concerned with elaborating available data to *decide* which actions to carry out to achieve the system goals—hence the double arrow in Figure 2. Also here, a whole stack of functions and a whole range of different approaches can co-exist: simple reactive rule engines producing commands depending on the current situation [70], more complex forms of event processing, agent-oriented computing to have goal-oriented behaviours [127], etc.

Actuation facet. It is the facet concerned with how to *affect* the "state of the world" according to the application goals—hence the downstream arrow in Figure 2. Again, depending on the business domain, many different approaches and functions belong to this facet, ranging from low-level programming of individual devices with mainstream languages, to collective engineering of the global system behaviour [16].

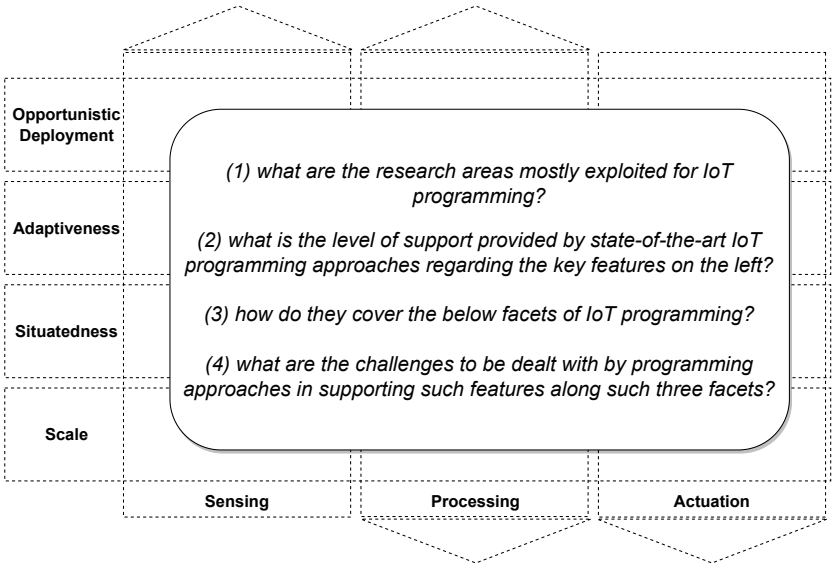


Figure 2: Our IoT programming conceptual framework. Vertical slices represent the three facets of computation in IoT, whereas the horizontal ones denote the four key features for modern IoT deployments we focus on.

These facets are depicted as the vertical slices of Figure 2, and describe the *main* purpose of a given approach or technique. The regions in the figure and the positioning of approaches therein are not to be interpreted in a rigid way. Indeed, the goal of such classification is to clarify what is the main role of a given approach with respect to the IoT programming landscape.

2.2. Key features

Any modern IoT system also needs to deal with four key features, depicted as horizontal slices in Figure 2, that we define below and motivate in Section 2.3, as they bear slightly different meanings depending on the research area where they are used.

Scale-independence. The property of a programming model or platform of being *transparent* to the *number* of devices to program and/or to the *spatial extension* of the system to program, such that the resulting behaviour appears *functionally the same* regardless of these two factors.

Situatedness. The property of being *functionally dependent* on the context (or, environment) and/or the space-time fabric, such that the resulting

behaviour appears *functionally different* depending on *where* devices executing the program are placed (and possibly on the topology of such space), *when* computations take place, and on any other property of interest observed in the computational or physical environment where the program executes [167].

Adaptiveness. The property of being able to *change behaviour* upon changing conditions, such that the resulting *behaviour* may *change* according to environment events, properties modification, system components actions, etc. [75] to either keep delivering the same functionality despite the disruption, or opportunistically exploit the chance to deliver new or improved functionalities [56].

Opportunistic deployment. The ability of a system to deploy and redeploy its components on a target executing infrastructure in the *most convenient* way, based on any property relevant to the application or for the benefit of non-functional properties [25, 58].

2.3. Motivating scenarios

The above-described features are key to many relevant application domains already compelling today, or that will be so in the near future. Table 2, at the end of the section, summarises the motivations described in the following.

Cooperative Driving. In contrast to *autonomous driving*, where the focus is on individual vehicles to hit the streets safely, in *cooperative driving* attention shifts to the *collective* of vehicles that need to cooperate to deal with everyday traffic conditions [47]. Essentially, autonomous vehicles will need to communicate with everything – other vehicles, road infrastructure, pedestrians – to both *(i) co-build* a state of the world, and *(ii) jointly* plan how to behave.

To do so, a suitable IoT platform gathering the required data and executing fundamental services must be available. In the case of, for instance, ride-sharing, intersection crossing, and traffic flow management, data must be collected about offers of ride-sharing trips, corresponding demand, the current situation with traffic congestion, information from the municipality about ongoing road work, etc. Thus, services must *(i)* operate depending on *time* and *space* properties, *(ii)* at a *variable scale* ranging from a single neighbourhood to a whole urban area, *(iii)* deeply tight to *contextual* information

including the *topology* of space, and *(iv)* ready to *adapt* to the unforeseen events of urban traffic. In other words, scale-independence, situatedness, and adaptiveness are all desirable features in the described scenario, if not requirements.

While some of them are somewhat supported by state of art techniques, as better discussed in Section 3, others are either the subject of ongoing research, or yet unexplored.

Smart Environments. Environments that enable the acquisition and the application of knowledge to improve their inhabitants' experience are usually called smart environments [33]. A Smart Home, Office, Hospital, Campus, or whole Smart Cities are examples of heterogeneous smart environments, with different requirements and goals, but share the need for the IoT as key enabler technology.

Universities, for instance, are continuously working to provide high-quality education and research in a safe, secure, and engaging environment. The advent of IoT has been seen as a means to improve the quality of the provided services and the overall involved processes. However, IoT systems deployed in smart campuses often result in being organised as silos lacking interoperability and integration [96], partly due to the separation of concerns which makes it reasonable to focus on the deployment of a specific system at a time. Therefore, once the IoT system is conceived and deployed, any change in its composition, its exposed functionalities, or the addition of new devices, is seen as a treat instead of an opportunity.

Future Smart Campus systems should be able to *adapt* to change, to act cooperatively, to communicate with each other, and to *move computation* to the most appropriate devices (e.g., those that have more resources, those that are located in the proximity of a point of interest, etc.). The data acquired by IoT devices could be managed differently based on the emergent situation (if the underlying processing platform is *situated*). IoT devices could be asked to process data in a faster way at the expense of resource usage since the current situation may be time-critical (as a form of *opportunistic deployment*).

In other words, adaptiveness, opportunistic deployment, and situatedness are all relevant features for these kinds of smart environments.

Industry 4.0. The synergistic exploitation of different technologies, there including IoT, Artificial Intelligence, and Digital Twins, with the goal of digitalising industrial processes and products, and augmenting them with intelligent capabilities, is fuelling the Industry 4.0 revolution. The IoT is a

Table 2: Exemplary scenarios motivating the key features of our conceptual framework.

Scenario	**Scale**	**Situatedness**	**Adaptiveness**	**Opportunistic Deployment**
Cooperative driving	junction neighbourhood city	time space road rules	failures accidents social events	Edge to Cloud
Smart environments	flat building city	time space user prefs	user goals failures social events	Edge to Cloud
Industry 4.0	floor plant	space time plant operations	failures production changes process changes	Edge to Cloud

key enabler: whether the goal is predictive maintenance of machinery and equipment, what-if analysis of business scenarios, or any other functionality, having an IoT deployment is likely to be a requirement [134]. In fact, the full realisation of Industry 4.0 is achieved when two conditions hold: *(i)* the relevant assets (machinery, equipment, products, processes, people, etc.) are monitored and possibly digitalised (e.g. through Digital Twins), and *(ii)* actionable knowledge generated by such digitalisation is put to work automatically, without human intervention for the most part [73].

Depending on many factors among which the size of the factory and its organisation layout, Industry 4.0 services need to be: *(i) scalable*, to allow for arbitrary aggregations of data and service delivery ranging from a single site, single production line, to multiple production lines or even different departments and sites; *(ii)* aware of the specific *locations* where data comes from and where services must be delivered, as well as of the *timing constraints* affecting production lines, and finally of the general *context* of operation (stock levels, urgency of orders, personnel available, logistics, etc.); *(iii)* capable of *adapting* to contingencies and re-configurations, such as the addition of new machinery or the re-design of a production line layout.

2.4. Architectural perspective

It is important to emphasise that the conceptual framework we are proposing should not be interpreted as a means to enforce any specific architecture on IoT systems. The platforms, middleware, and software stacks that will be discussed throughout Section 3 each comes with its own architecture that we do not discuss, intentionally, as we are focussed on the abstractions and mechanisms they provide to deal with the four key features

of IoT programming just described. The architectural prescription about *how* to deliver them, that is, using which components that shall interact in what ways, has already been discussed by "reference" architectures proposed for the IoT in the literature [59, 27, 158]. Furthermore, such reference architectures are mostly feature-agnostic, as their utmost concern is to provide general-purpose services virtually applicable to any IoT deployment—such as device management, discovery, protocol translation, message passing, and such. Our framework, instead, serves to better understand how the currently available IoT programming tools (platforms, models, etc.) deal with the three facets of computation and the four key features of modern IoT systems. In this sense, our proposed conceptual framework is complementary to IoT reference architectures: once the key requirements of the IoT system or application at hand have been outlined, one can use our framework to quickly search for the needed mechanisms as found at the relevant crossroads, then look for the most appropriate architecture to support them based on deployment constraints.

3. State of the art

Now that our conceptual framework has been introduced and motivated, we can actually report on our survey to place literature in there. First, we present in detail the survey methods adopted, then describe the research areas emerged from our search in Sections 3.2, 3.3, and 3.4.

3.1. Research method

Although this is not meant to be a systematic literature review, as we are interested in a focused analysis of the key features and facets, we nevertheless aim to be as less arbitrary and ambiguous as possible in our research, hence here we describe the survey process.

We started by performing a keyword-based search on Scopus and DBLP, considering title, abstract, and keywords of indexed papers, as follows—$\vee$ connects keywords in OR, $\wedge$ in AND, and brackets disambiguate the priority of logic operators:

$$\begin{aligned}&\big(\text{programming } (\text{languages} \vee \text{models} \vee \text{approaches} \vee \text{abstractions} \vee \text{libraries} \vee \text{frameworks})\big)\\ \wedge&(\text{survey} \vee \text{review} \vee \text{SLR} \vee \text{systematic literature review} \vee \text{mapping study} \vee \text{secondary study})\\ \wedge&\big(\text{IoT} \vee \text{internet of things} \vee \text{CPS} \vee \text{cyber-physical systems} \vee (\text{edge} \vee \text{fog}) \text{ computing}\big)\end{aligned}$$

In other words, we performed several searches such as "IoT programming languages survey", "CPS programming approaches review", and similar combinations, targeting papers that already are surveys themselves. The motivation is that the topic of IoT programming is so broad and has so many different nuances that it would be impossible to rigorously scrutinise all the available literature. By focussing on existing surveys, instead, we can realistically analyse them in detail, and later perform snowballing through the most relevant contributions found. The results obtained are further filtered:

1. by comparing title and metadata, we filtered out duplicates and different versions of the same paper (e.g. conference and journal versions)

2. by reading the abstract, we filtered out papers that clearly were out of scope (e.g. no focus on programming)

3. by reading the full paper, we filtered out papers that were either superseded by another one, sensibly overlapped with another one, out of scope, etc.

Then, we grouped the collected papers into categories, to snowball and get a more focussed overview of the research landscape. What emerged from such a search process, is that IoT programming is mostly dealt with by three research macro-areas, which we describe in the following. Thus, this survey organises the collected literature according to these three macro-areas, as reflected by the organisation in Sections 3.2,3.3, and 3.4.

Stream processing (SP) [13] A computing paradigm that allows the gathering, processing, and analysis of continuous flows of data called *streams* within a small time period from the time of receiving. SP gained popularity within the IoT landscape as oftentimes IoT deployments need first and foremost to tackle the challenge of collecting and processing vast amounts of data continuously coming from some system under monitoring, as in the case of Industry 4.0, smart supply chain and logistics, intelligent transportation, and telemedicine. As a consequence, SP became one of the fundamental assets available to developers for programming IoT systems, especially covering the "sensing facet" of our conceptual framework.

Complex Event Processing (CEP) [89] Builds on the abstraction of low-level events into a new high-level one based on temporal, spatial,

or causal relations. CEP is closely related to SP: it shares the general idea of processing potentially *unbounded* streams of data, and many techniques for actually doing so. CEP comes into play for IoT applications not limited to simple monitoring tasks—for which SP is often times fair enough. Whenever there is a need for complex analytics and insight, and to some extent closing the feedback loop by controlling the monitored system back, CEP has an edge over other approaches thanks to its native capability of detect complex patterns over incoming events data and trigger business processes reactively. For these reasons, we put CEP at the "processing facet" of IoT programming.

Micro and Macro Programming (MMP) This category includes approaches not belonging to a well-defined research area, such as traditional programming (micro) where a program is developed with a specific, *single* device in mind (e.g. a computer, a mobile phone, an embedded device), as well as emerging programming paradigms that target a *multitude* of devices at once (macro) [61], by developing a global, high-level program that is then "translated" into lower level, device-bound (traditional) programs [16]. A description of each approach is given in the corresponding Section 3.4. Being these approaches focussed on *controlling* devices programmatically, they mostly belong to the "actuation facet" of our conceptual framework.

We then expanded our original keyword-based search with terms directly stemming from the identified research areas, in a snowballing process—replacing the first row of the original query:

- $\big($(stream $\vee$ flow $\vee$ data flow) (programming $\vee$ processing) $\big)$
- $\big($(behaviour $\vee$ reactive $\vee$ event-driven $\vee$ event) (programming $\vee$ processing) $\big)$

On the results retrieved, the same filtering steps already mentioned have been applied. The process is summarised by Figure 3. The papers that survived are actually the ones cited throughout this whole section, and summarised by Tables 3–7

It is worth emphasising a few aspects of these research areas: *(i)* first, all of them are increasingly used in the context of IoT services and applications, either as components of a broader architecture or in isolation, as key enablers of programming tasks related to monitoring, decision-making,

and distributed control; *(ii)* second, SP mostly deals with the sensing facet, and partially with the processing one, CEP mostly deals with the processing facet, and partially with the sensing and actuation ones, and MMP is primarily concerned with the actuation facet and partially with the processing one, hence they seemingly fit our conceptual framework, and altogether cover it broadly (see Sections 3.2.3, 3.3.3, and 3.4.3 for a discussion of each area). On the one hand, this means that each research area brings effective solutions to specific issues of programming large-scale, autonomous IoT systems, on the other hand, these solutions are only partially solving the problem, hence integrating them could be the key to advancing the state of art in IoT programming.

The following subsections discuss the state-of-the-art in each of these three macro-areas, with a specific focus on the four key features defined in our reference conceptual framework. The surveys and papers collected have been roughly divided into three categories – Tables 3, 5, and 7, one per each macro-area –, depending on their scope and intended goal: a first category (GEN) is focussed on the comparison of available languages and platforms across many **gen**eral criteria, a second one (DOM) is concerned with the usage of languages and platforms in specific application **dom**ains (e.g., Smart Cities and Industry 4.0.), a third one (SPEC) examines a **spec**ific design choice or property (e.g., main abstraction, scheduling techniques, parallelisation) of the languages and platforms, and compares them based on the degree of support provided.

3.2. Stream Processing

Stream Processing (SP) is a programming paradigm that sees application development as processing *continuously* incoming data: given a sequence of data (a stream), a series of operations is applied to each element in order.

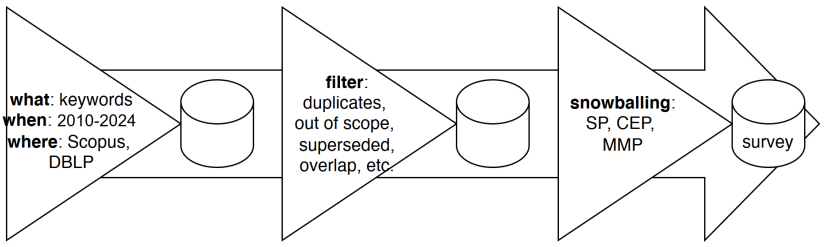


Figure 3: The survey process: after a first general search for "IoT programming", snowballing on the resulting main research areas has been applied.

When data is associated with a timestamp the literature usually changes terminology and talks about Event Stream Processing (ESP) platforms. (E)SP deals with high volumes of data produced and organised in a single *stream*, ordered by timestamps, that have to be processed as fast as possible to provide near real-time insight. A typical example is a stock market feed that should be analysed to determine whether to buy or sell a stock in such a way to gain profit. ESP applications do not normally include event causality or event hierarchies but can [91].

Amongst the characteristics that an SP platform should present [141], a dominant one is the ability to process data on the fly, which can be regarded as a form of *situatedness* along the temporal dimension. Especially in near-real-time systems, waiting for data to be stored (e.g. in a database) before being processed adds unnecessary latency. Modern SP platforms are also capable of partitioning computational load and *scaling* applications automatically, and of distributing its processing needs across multiple processors and machines. In the most advanced systems, the distribution is handled automatically and transparently, as a form of *opportunistic deployment*.

SP dates back to projects such as Aurora/Borealis [3, 2], Stream [9], and TelegraphCQ [28]. Nowadays the most popular SP platforms are Flink, Heron, Kafka Streams, Samza, Spark Streaming, and Storm (all referenced in the following). Several contributions have reported a comparison of these platforms in terms of the functionalities and applications domain they support: we provide a summary of such literature to quickly "frame" the research area and to later emphasise which characteristics of SP platforms do matter the most for our conceptual framework.

3.2.1. Related Surveys

Many surveys about SP or ESP can be found, witnessing the huge interest in the topic, as summarised in Table 3.

In [52] the authors trace back the history of SP systems: they review the main research findings, and emphasise similarities and differences between generations of streaming systems, concluding with a discussion about trends and open problems. In [39] the authors provide a broad overview of event processing system architectures, use cases, and open research topics. The survey in [164] provides a specific account for real-time processing technologies of IoT data streams, and reports about the challenges that the real-time utilisation of IoT data flows presents; it also introduces a middleware called Information Flow of Things (IFoT) [144] proposed as a partial solution to

face some of the mentioned challenges. In [66] a comparative study of distributed SP and analytics frameworks, and a critical review of representative open source (Storm, Spark Streaming, Flink, Kafka Streams) and commercial (IBM Streams) platforms/frameworks is presented. In [100] the authors examine the applicability of employing distributed SP frameworks at the data processing layer of Smart Cities and appraise their adoption and maturity amongst Smart City use cases. The authors report on experiments carried out to evaluate the performance of SP platforms such as Apache Storm, Apache Spark Streaming, and Apache Flink. Ref. [122] focuses on predictive maintenance for the Industry 4.0 domain, by systematically reviewing the strengths and weaknesses of existing open-source technologies concerning the requirements put forth by big data and SP applications within the mentioned

Table 3: Main SP and ESP surveys, most recent first.

Ref.	**Brief description**	**Year**	**Category**
[126]	Covers only the sensing facet of computation, does not consider the programming model	2022	GEN
[148]	Focus on techniques for Cloud elasticity (mostly QoS-aware resource usage) at operator placement/execution level	2021	DOM
[52]	Historical perspective, emphasising evolution of research landscape, and comparing different SP platforms generations	2020	GEN
[122]	Systematic analysis of strengths and weaknesses of open source SP techs for predictive maintenance in Industry 4.0	2020	DOM
[66]	Comparative study of selected distributed SP & analytics platforms	2019	GEN
[100]	Evaluation of selected open source distributed SP platforms deployed at the "data layer" of Smart City applications	2019	DOM
[146]	Review of scheduling techniques for distributed SP	2019	SPEC
[120]	Focus on techniques for parallelisation and elasticity of SP	2019	SPEC
[44]	Focus on SP mechanisms leveraging resource elasticity in the Cloud	2018	SPEC
[39]	Overview of ESP architectures, use cases, and open research issues	2018	GEN
[164]	Specifically concerned with real-time SP in IoT deployments, hence surveys related techniques, challenges, and technologies	2016	SPEC

domain. In [146] the authors present a review of distributed stream management systems along with their strengths and limitations. In particular, they provide a classification of scheduling techniques. In [44] and [148] the authors survey SP engines and mechanisms focusing on the resource elasticity of cloud computing, which allows an application or service to scale out/in according to fluctuating demand. In [120] the authors provide a classification of parallelisation and elasticity mechanisms in SP systems, depending on the specific application requirements.

No survey puts emphasis on either axis of our proposed conceptual framework for IoT programming, but there are SP approaches that deal with some of the key features we focus on, although the terminology is not always consistent with ours. The next Subsection overviews the main characteristics of SP platforms, that are used by the mentioned surveys to categorise the literature, and by us to analyse the support provided to our four key features.

3.2.2. Main characteristics of SP platforms

Many are the characteristics of SP platforms and frameworks that could contribute to meeting the requirements put forth by the four key features of IoT programming we identified in our conceptual framework. Here, we derive from the surveyed literature the main characteristics of SP platforms (Table 4) and compare well-known SP platforms based on the degree of support they provide to those characteristics. This eases the task of devising to what extent the four key features of our conceptual framework are accounted for in the current state of the art—done in the next Subsection.

- *Programming Model*. It defines if programmers use *declarative* queries, *imperative* programming, or *compositional* programming to specify the operations or execution patterns of an SP system. Imperative programming specifies how to generate data structures, how to store temporary intermediate results within these structures, and how to convert these intermediate results into final results. Declarative programming defines what tasks to complete instead of delivering as many details as imperative programming does. Compositional approaches offer basic building blocks for composing custom operators and topologies.

- *Processing Model*. Whether data undergo *batch* processing, *micro-batch* processing, or *native* stream processing (tuple by tuple). Batch processing stores data first, and then applies processing; micro-batch processing runs batch processing on a few data clustered together (the

micro-batch) so that processing occurs more frequently; native stream processing collects and processes data immediately, as they are generated.

- *Deployment.* The infrastructure on which an SP system is deployed: e.g., a cluster, the Cloud, Fog, Edge, etc.

- *Parallelisation.* Parallel SP reduces queuing latency and increases throughput, as it allows the processing of multiple streams simultaneously instead of sequentially. With *task* parallelisation, multiple operations (i.e., tasks) can run in parallel on the same stream. With *data* parallelisation, multiple instances of the same operator can act in parallel on different streams.

- *Stream Primitive.* It refers to the first-class data structure of the streaming system. In Kafka Streams[1], for instance, it is an ordered, replayable, and fault-tolerant sequence of immutable data records, where a data record is defined as a key-value pair. In Samza[2], instead, the stream primitive is a message, processed one at a time upon receiving. Finally, in Spark Streaming[3] Datasets and DataFrames are used, where a Dataset is a distributed collection of data, and a DataFrame is a Dataset organised into named columns—hence conceptually equivalent to a table in a relational database.

- *Delivery Guarantee.* The kind of warranty that data will be delivered from the SP platform to its client applications: at most once, at least once, and effectively once (or exactly once). With *at most once* data will be delivered once or not at all. With *at least once* data is guaranteed to be delivered, but possibly more than once. With *effectively once* data will be delivered exactly once.

- *Scalability.* The kind of scalability operations supported, e.g. adding computational resources to a processing unit (vertical) vs. adding processing units (horizontal).

[1] `https://kafka.apache.org/11/documentation/streams/core-concepts`

[2] `http://samza.apache.org/learn/documentation/1.6.0/core-concepts/core-concepts.html`

[3] `https://spark.apache.org/docs/latest/sql-programming-guide.html`

Table 4: Summary of the SP main characteristics.

	Flink [21]	**Heron** [82]	**Kafka Streams** [79]	**Samza** [106]	**Spark Streaming** [166]	**Storm** [151]
Programming Model	Declarative, Imperative	Imperative	Declarative	Compositional	Declarative, Imperative	Imperative, Compositional
Processing Model	Hybrid	Streaming	Batch/Streaming	Hybrid	Micro-Batch	Streaming, Operator Based, Hybrid (with Trident)
Deployment	Cluster, Cloud, Fog, Local	Cluster, Cloud, Fog	Containers, VMs, Cloud, Edge	Cluster Standalone Public-cloud Containers Bare-metal	Cluster, Cloud, Fog	Cluster, Cloud, Fog
Parallelisation	Task, Data	Task, Data	Task, Data	Task, Data	Task, Data	Task, Data
Stream Primitive	DataStream	Tuple	Stream	Message	DataSet DataFrame	Tuple
Delivery Guarantee	Exactly once	At least once, Effectively once	At least once, At most once, Exactly once	At least once	Exactly once, At-least-once	At least once, Exactly once (with Trident)
Scalability	Horizontal Vertical	Horizontal Vertical	Horizontal Vertical	Horizontal Vertical	Horizontal Vertical	Horizontal Vertical
Data Flow Abstraction	DAG	DAG	DAG	DAG	DAG	DAG
Fault Tolerance	Stream replay Checkpoint	Checkpoint	Stream replay Checkpoint	Incremental Checkpointing	Checkpoint	Acking, Checkpoint, Stream replay, Highly fault-tolerant
API languages	Java, Scala, Python, SQL	C++, Python Java, No SQL	Java, Scala, KSQL	Java, SQL, Python	Java, Scala, R, Python, Spark SQL	Java, Python, SQL

- *Data Flow Abstraction.* This is the abstraction used to handle the flow of data between stream operators. Most SP platforms adopt a *Directed Acyclic Graphs* (DAG) of operators to structure the processing application in topologies.

- *Fault Tolerance.* The ability of an SP system to tolerate failures. Data loss and resource access loss are common failures of distributed SP engines, and common recovery strategies are based on replaying the whole data stream, or restore the engine state to a checkpoint (before the fault) to repeat the processing pipeline only from there.

- *API languages.* The programming paradigm and the actual language stack that is supported. The trend is to integrate with mainstream, general-purpose languages, rather than to provide custom domain-specific languages.

In Table 4 the design choices of each specific SP platform (columns) to implement the corresponding characteristic (rows) are briefly described by keywords. In the next Subsection, we analyse these concerning the four key features of our conceptual framework for IoT programming, intending to respond to our research questions (1) and (2).

3.2.3. Features and facets analysis

As regards the computation facets (vertical slices), SP is mostly concerned with sensing as it is usually adopted to gather sensor measurements in IoT deployments. However, some SP platforms and frameworks also offer processing techniques and algorithms, hence the processing facet is also partially covered. As regards the four key features, instead, the following paragraphs discuss them, and Figure 4 that follows visually depicts a summary.

Scale-independence. Adopting a programming model based on *declarative languages* lets the SP platform neatly separate the application logic from its deployment on an executing infrastructure, hence allowing for run-time *operator placement* and to scale vertically and horizontally more easily [120]. Also representing the application logic as a DAG of stream operators contributes in supporting scale independence, as *task/data parallelism* can then be exploited based on the available hardware and an operator placement strategy.

In summary, *scale-independence* is one of the foremost concerns of the SP community, being the capability of continuously processing arbitrarily large amounts of data coming from (geographically) distributed sources the main driver of SP conception, research, and application.

Situatedness. On the one hand, the very fact that a data stream must be processed in (near) real-time situates that information – and, in turn, the processing activities triggered because of such data –, in time. *Native SP* is the best mechanism for this, as it does not introduce any processing latency. On the other hand, situatedness along the spatial dimension or application-specific contexts is difficult to spot in the surveyed literature. For instance, we found nothing about the possibility to dynamically change the processing pipeline of a stream based on some spatial metadata attached to that stream or any other application-specific property of interest. The only kind of spatial situatedness could be related to Edge SP platforms, in a similar vein as for native stream processing: having the computations carried out directly on Edge devices somewhat situates the computation itself in space. However, this cannot be programmed at will, and is simply a form of situatedness inherited from deployment constraints. As a result, in the current SP platform, scenarios that require situated SP can be realised only by resorting to *custom implementations* on top of the SP system, that are likely to be highly dependent on the specific scenario.

Nevertheless, recent proposals have been made for introducing *situation-awareness* into SP applications. In [131, 132], for instance, a modular service-based architecture for realising situation-aware, adaptive ESP is proposed, where authors define a *custom language* to specify processing templates.

In summary, situatedness beyond the temporal dimension is rarely accounted for, except for a few proposals that however do not aim at defining a generally applicable (across application domains) notion of "situated SP" but, rather, support an application-specific notion and thus develop an ad-hoc solution.

Adaptiveness. Most of the current SP adaptation mechanisms are limited to adjustments of response time or resource usage, hence somewhat overlap with scale independence. Examples of those mechanisms are *stream partitioning*, that is splitting streams to multiple processing nodes e.g. load balancing reasons, and *operator placement* [62], that is deciding where best to place the execution of stream operators on available computational nodes. However, more general forms of adaptation, for instance, based on application-specific policies, or domain-specific dynamics observed at run-time are rarely considered. In this case, imperative languages that fully specify the operations to perform on streams instead of partially delegating them to the underlying SP platform – as declarative approaches do – have the advantage of enabling programmers to arbitrarily implement adaptations, such as operator placement and partitioning strategies, depending on any application-specific property of interest [44]. However, this comes at the cost of sacrificing the automatic mapping of the application DAG to the executing infrastructure.

Opportunistic Deployment. Most of the surveyed SP platforms target a cluster of machines on-premise, or Cloud provisioning as deployment infrastructure. In any case, deployment is fixed: it is rarely considered the opportunity to migrate processing components across the Edge-to-Cloud spectrum.

Some attempts have been made to move SP directly at the Edge: in [48] Edgewise is proposed as an Edge-friendly SP engine which incorporates a congestion-aware scheduler to improve throughput and latency; the LF Edge organisation[4] aims to establish an open, interoperable framework for Edge computing independent of hardware, silicon, Cloud, or operating system. This organisation is carrying on several Edge-oriented projects such as the

[4]LF Edge organisation website: `https://www.lfedge.org/`

open source software frameworks EdgeX Foundry[5], that provides interoperability between heterogeneous devices and applications at the Edge, and Akraino[6], a set of open infrastructures and application blueprints spanning a broad variety of use cases, including 5G, AI, Edge IaaS/PaaS, IoT, for both provider and enterprise Edge domains. In [87] a comparison between open source Edge computing platforms is presented. One of the presented platforms is Apache Edgent[7], explicitly designed for data processing directly on-board Edge devices. Also, Cloud providers like Amazon AWS, Microsoft Azure, and Alibaba Cloud extended some of their Cloud features to tightly integrate with Edge devices: Amazon's GreenGrass[8], Azure IoT Edge[9], Link IoT Edge[10]. They offer some programming and management capabilities on the Edge but push analytics to their Cloud services anyway.

In [119] the authors present a software stack called R-Pulsar that extends Cloud capabilities to local devices and provides a programming model for deciding what, when, and where data gets collected and processed. However, yet again the deployment is fixed: even in the case of R-Pulsar, Cloud capabilities are moved to Edge devices, but no migration across the Edge-to-Cloud spectrum is considered.

3.2.4. Summary

Figure 4 summarises our analysis of the SP macro-area emerged from our survey, by placing the design choices, mechanisms, or abstractions featured in the SP literature (circles) at the crossroads between key features and facets in our conceptual framework.

Summing up, currently available SP solutions offer wide support to scale independence, mainly via load shedding, operator placement, and Cloud-like elasticity. Adaptiveness too is somewhat supported, although mostly restricted to resource usage concerns, hence sensibly overlapping with scale-independence. Another limited support is provided for situatedness, that is almost exclusively focused on the temporal dimension. Finally, whereas there is wide interest in moving SP out of the Cloud, those platforms that also con-

[5]EdgeX Foundry website: `https://www.edgexfoundry.org/`
[6]Akraino website: `https://www.lfedge.org/projects/akraino/`
[7]Apache Edgent website: `https://edgent.incubator.apache.org/`
[8]Amazon's GreenGrass website: `https://aws.amazon.com/it/greengrass/`
[9]Azure IoT Edge website: `https://azure.microsoft.com/services/iot-edge/`
[10]Link IoT Edge website: `https://www.alibabacloud.com/product/linkiotedge`

sider SP at the Edge do not provide mechanisms for run-time modifications, but only enable design-time decisions about where to deploy processing, at the time of writing.

3.3. Complex Event Processing

Complex Event Processing (CEP) systems are concerned with processing streams of *events*, that is data describing situations and happenings in a system under observation [89]. Specifically, they target the definition and detection of high-level situations of interest, or *composite events*, starting from streams of *primitive events* [38]. Event Stream Processing (ESP), as an evolution of SP, and CEP indicate different ways to manage events [90]: ESP is a subset of CEP, with very fast implementations available, that can handle many thousands of events per second, at the price of lower processing capabilities. Nowadays, an increasing number of enterprises are slowly progressing from ESP towards CEP [92, 37, 77]. CEP foremost goal is to enable the abstraction of low-level events into a new high-level event based

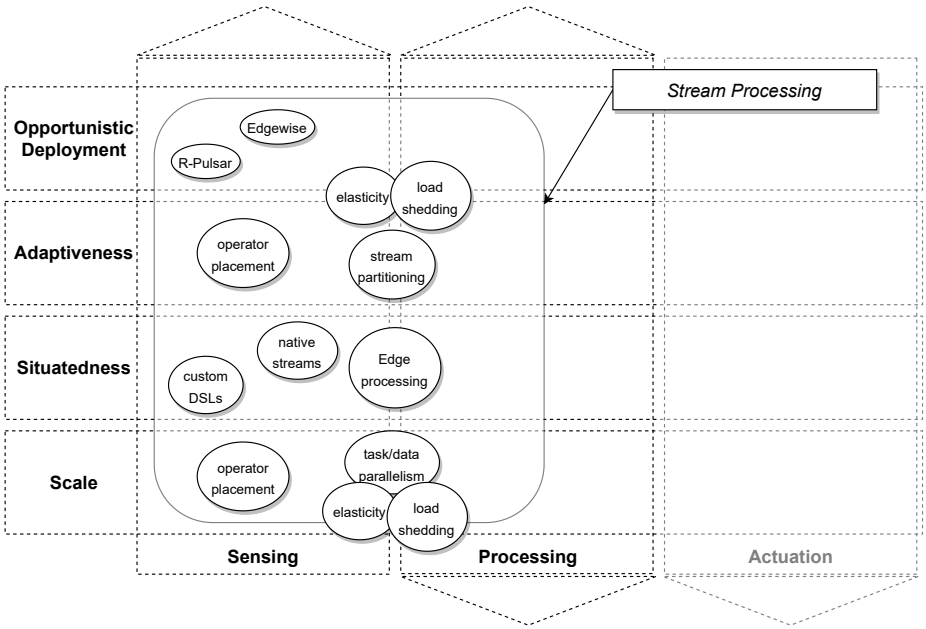


Figure 4: SP abstractions, mechanisms, and techniques placed in our conceptual framework. Placement is not always crisp (e.g. elasticity and load shedding between opportunistic deployment and adaptiveness), as the literature rarely explicitly defines which key feature or processing facet a mechanism supports, hence we devise it out.

on temporal, spatial, or causal patterns. The patterns to be detected are generally specified by domain experts, and the new events generated from pattern matching are viewed as being at a higher level of abstraction than those in input. As such, CEP allows for a stepwise abstraction process, easing the increasing flood of events that modern information systems have to deal with.

CEP is usually traced back to projects such as Amit [4], PADRES [86], SASE [160, 60], PEEX [74], RACED [35], and TESLA/T-Rex [36]. Several contributions have discussed CEP platforms [37, 50, 54]. In the following, we provide a summary of such literature to quickly "frame" the research area, and to later emphasise which characteristics of CEP platforms matter the most for our conceptual framework.

3.3.1. Related Surveys

As for SP, most of the collected surveys can be divided into three categories, as depicted in Table 5: some focus on a generic comparison of the Complex Event Processing systems (GEN), others on the usage of CEP in specific application domains (DOM), and others focus on specific characteristics or mechanisms of the CEP systems, such as querying techniques, parallelisation, and uncertainty handling (SPEC).

In [54] the authors survey Complex Event Recognition (CER), that is, the identification of complex events that satisfy some pattern—in practice, a subtask of CEP. The survey covers the query mechanisms and the algorithmic toolkits for scaling out CER to clustered and geo-distributed deployments. In [173], the authors argue that nowadays applications leverage CEP only in Cloud-based environments although data are generated outside the cloud (especially in the context of IoT). This centralised data collection, before applying data processing, introduces a critical bottleneck, especially for delay-sensitive IoT applications. However, the authors argue that CEP systems are not yet ready to leverage the Fog layer, let alone the Edge, as an extension for Cloud-based processing. In [170] the authors provide an overview of the state of the art of CEP systems, especially with respect to the IoT domain. They argue that not only CEP, but the (often) underlying SP engines have to change to exploit the new capabilities of the IoT. In [93] the authors discuss the application of CEP to the healthcare domain by focussing on usage of sensor devices. They argue that CEP is the appropriate solution for many critical situations in healthcare, to analyse and respond to unexpected events for early prediction and diagnosis, and to re-

Table 5: Main CEP surveys, most recent first.

Ref.	Brief description	Year	Category
[83]	Very detailed SLR about rule-based CEP, also including a thorough evaluation	2024	SPEC
[140]	Focus on CEP applied to business process management, and especially on analsyis of integration architecture	2021	DOM
[156]	Restricted to machine learning-based approaches, with a focus on integration mechanisms and architectures for smart factories	2020	SPEC
[54]	Focus on CER (subtask of CEP), especially regarding scale-out mechanisms for query execution	2020	SPEC
[173]	Focus on out-Cloud deployment, in particular adaptations needed for CEP platforms to meet the Fog layer	2020	SPEC
[170]	Overview on the state-of-the-art of CEP, also discussing the relationship with SP to improve efficacy in IoT deployments	2020	GEN
[6]	Focus on probabilistic CEP for uncertainty handling	2017	SPEC
[93]	Specifically concerned with the application of CEP to healthcare, with a particular focus on how to exploit healthcare sensor devices	2017	DOM
[8]	Focus on languages for CEP applied to multimedia sensor networks	2016	SPEC

duce patients' routine activities by self-assessment and monitoring of their health condition. In [8] the authors identify requirements of a suitable language for processing complex events in multimedia sensor networks. The requirements are illustrated through a motivating scenario related to "Smart Home Automation" applications. Their conclusions show that no existing language can fully address all the requirements. In [6] the authors review CER techniques that handle, to some extent, various types of uncertainty, ranging from incomplete and erroneous data streams to imperfect complex event patterns. They identify several limitations concerning the employed languages, their probabilistic models, and their performance as compared to the purely deterministic cases. Based on those limitations, they highlight directions for future work.

In summary, apart from scaling mechanisms and out-of-Cloud deployment, that could be regarded as one particular aspect of opportunistic deployment, there are no surveys available explicitly focussing on the four key features of our conceptual framework, nor providing for a clear separation of concerns along the three facets of computation we emphasised.

3.3.2. Main characteristics

As done for SP in Subsection 3.2, clarifying which are the main characteristics of CEP approaches and platforms helps to understand how they contribute to supporting the four key features of our conceptual framework. Accordingly, Table 6 reports a comparison similar to the one reported in Table 4, and with the same purpose of easing the key feature analysis presented in the next Subsection, but in the context of CEP—please notice that columns and rows are inverted w.r.t. Table 4 to preserve readability.

- *Filtering.* The ability to discard irrelevant events, so as to reduce the amount of events to be considered for processing, is of paramount importance to keep CEP systems performant and focussed on the application goals. The criteria used for filtering are usually defined at design time, but it is desirable to have ways to at least adjust them at run-time.

- *Prioritisation.* CEP platforms must be able to establish which events to process sooner than others, since they may require immediate actions that critically affect the business. Again, the criteria for establishing the priority order are usually statically defined at design time, but is desirable to have ways to adapt such priorities during run-time, as a reaction to the system dynamics.

- *Patterns.* Detection and matching of event patterns is what essentially defines CEP itself. Such patterns are usually defined at design time and cannot be altered, as the automatic mechanisms of events handling underlying CEP would be compromised. Pattern detection, matching, and querying are complex tasks that are usually exposed to the system designer through a suitable API constituting the *Event Query Language (EQL)*, further discussed below. This is one crucial difference between CEP and SP.

- *Exceptions.* Within CEP, exception handling refers to a pattern that fails to find any match within a specific period. This may indicate the presence of some kind of anomaly, hence an exceptional event should be generated, to undergo CEP on its own.

- *Process triggering.* Patterns of events can be used to trigger business processes that react to expected patterns with the automatic execution

of predefined business actions. This is another notable difference between CEP and SP, as in the latter case process triggering is entirely the responsibility of the application developer.

- *Event hierarchies*. The capability of building layers of events at different levels of abstraction, automatically shaped by the CEP platform, and linked together to preserve/restore/build causality.

- *Deployment*. How and where CEP solutions are deployed: Centralised, Distributed, Clustered, Cloud, Edge, etc.

As pattern detection and matching is a cornerstone of CEP, it is necessary to expand on it before proceeding to the analysis of the key features. What enables system designers to define pattern matching is the EQL, a high-level programming language for querying events [46, 54]. Several types of EQL have been defined in the CEP literature:

- In *event algebras*, complex event queries are expressed by composing single events using different composition operators, such as conjunction of events (all events must happen, possibly at different times), sequence (all events happen in the specified order), and negation within a sequence (an event does not happen in the time between two other events).

- *Data stream query languages* have been developed in the context of relational data stream management systems, hence they are usually based on SQL. They are applied in situations where loading data into a traditional database would require too much time, therefore they target nearly real-time applications.

- *Production rules* is not an EQL strictly speaking, however, they offer a convenient and flexible way of implementing EQL in the form of IF <Condition> THEN <Action> rules, hence are largely used in business rules management systems.

- *Logic languages* express event queries in logic-style formulas [10]. An early representative of this language style is the event calculus [78], which has been used to model event querying and reasoning tasks in languages such as Prolog. Logic languages have strong formal foundations and allow an intuitive specification of complex temporal conditions.

- Finally, in *tree-based* approaches patterns are represented as a tree structure, where leaves are primitive events and internal nodes are the operators contributing to defining the pattern.

The next Subsection places each mechanism, technique, and abstraction at the crossroads of our conceptual framework, to help answer our research questions (1) and (2).

Table 6: Summary of the main characteristics of the surveyed CEP systems (rows and columns are exchanged w.r.t. Table 4 to improve layout). NFA = Nondeterministic Finite Automata, BRMS = Business rules management system.

	Filtering	**Prioritisation**	**Patterns**	**Exceptions**	**Process triggering**	**Event hierarchies**	**Deployment**
SASE [160]	Event Algebra		Active Instance Stack (AIS) for events sequence construction on NFA			Primitive types to build Complex types	Centralised
Esper [63]	Event Processing Language	Priority language construct to control processing order	Pattern matching via regular expressions + event correlation + control on patterns execution (e.g. repeat-until, every-distinct, ...)			Event-type inheritance and polymorphism for hierarchies	Processing is centralised data acquisition distributed
Siddhi [142]	Siddhi Streaming SQL	Via rabbitmq's AMQP message priority	Pattern and sequence queries via state machines				Centralised as library + microservice on Docker Kubernetes
Cayuga [42]	Automata-based	Priority queues	Sequences of correlated events + "safety conditions" between consecutive events + patterns composition			Via query chains	Centralised
CEP [5]	DOLCE language		Causal and temporal patterns using pre-defined rules based on thresholds + dynamic rules				Distributed on embedded devices
Oracle CEP [108]	Oracle CQL		`CQL MATCH_RECOGNIZE` condition		POJOs triggered by events	Event Processing Networks to create hierarchy of processing agents and events	Clustered
OpenCEP [64]	Automata & tree based	Consumption policies and selection strategies	Patterns require structure (sequence, and, etc.) + conditions on each item + time window				Standalone library + Cloud deployment + integration with SP platforms
NextCEP [133]	Automata-based		SQL-like language + filter, union, iteration, time, ...)	Dedicated language operator			Clustered
TIBCO Business Events [57]	Production Rules		Sequencing, duplicate detection, and others		Event-driven process orchestration		Distributed via Docker or Kubernetes + TIBCO Cloud + on premise
EdgeCEP [32]	Specification language for relational expressions		By stating an explicit event key and contents of interest				Edge
BoboCEP [113]	Automata-based		Based on NFA				Distributed on Edge
Decision CEP Engine [65]	Data stream query language						Clustered
StreamInsight [7]	Declarative LINQ						Centralised
Drools Fusion [147]	Production Rules	Rules can have priorities	Improved Rete algorithm + temporal operators		BRMS to trigger business process execution		Centralised
ILOG JRules [117]	Production Rules				BRMS to trigger and support business process execution		Centralised
Spark CEP [125]	Data stream query language						Distributed
Proton [137]	Data stream query language		Filtering + join for multiple events + sequencing + aggregation functions + track values in time			Event Processing Networks to create event hierarchies	Centralised + distributed via STORM
EasyFlinkCEP [55]	Extended regular expressions		EasyFlinkCEP Operator + FlinkCEP's language to define CEP queries + time windows				Clustered

3.3.3. Features and facets analysis

CEP somewhat encompasses all three facets of computation, although rather weakly on the vertical slice corresponding to actuation. W.r.t. to SP, thus, we are moving from (almost) exclusively dealing with the sensing facet of IoT programming, to including more substantially the processing facet into the concerns of IoT developers. This is mostly due to the general capability of CEP to support more comprehensive and advanced forms of processing on input data (events) w.r.t. SP: advanced pattern matching techniques as well as event query languages, together with the capability to express business process triggering rules and policies, make CEP much more suited to program processing pipelines not only focusing on the aggregation of sensory data streams.

As regards the four key features, the following paragraphs thoroughly discuss their coverage as emerging from the surveyed literature.

Scale-independence. Almost all CEP engines support scaling up (vertical), also considering that most of them assume a Cloud or cluster deployment infrastructure. They are designed to take advantage of increasing processing cores, memory, etc. to have an extra degree of parallelism. Further, the in-memory caches often exploited also benefit scaling-up techniques. To attain greater parallelism, scale-out techniques have to be employed. However, the literature witnesses limited support for scaling out (horizontally) offered by CEP platforms. Where the platform does not support native horizontal scaling, multiple CEP engines can be used to create a load-balanced environment: events from the system under control can be routed to a central processing engine, that applies a filter to create a context-based partition of the incoming events. These partitions, in turn, are then sent to the worker CEP engines deployed in a parallel topology, based on load-balancing algorithms. Further correlation, pattern matching, aggregation, and filtering happen in these worker engines. The results are then merged and forwarded to the appropriate consumers. In any case, if the event stream cannot be partitioned using a filter, or if the correlation happens across the event stream partitions, scaling out is not possible.

To tackle the scalability challenge, in [143] the authors propose a parallelisation model called PARS+ that supports stateless and stateful CEP operators and runs them in parallel regions. They use PARS+ as the base parallelisation model in the formulation of an adaptive strategy called ACEP to auto-scale operators. Scaling decisions are governed by a predictive perfor-

mance model that uses a control-theoretic method for estimating the resource and latency costs of each operator at runtime. The computational loads of clustered processing nodes are monitored, and processing nodes in a parallel region are reconfigured at runtime to ensure a balanced load across the network, accruing minimum cost to parallelize a stateful operator.

Situatedness. Situatedness is somewhat natively addressed in CEP, similarly to the case of SP: event streams are naturally situated in time and possibly in space (e.g. in terms of the location where the sensor equipment is deployed), hence a first degree of situatedness w.r.t. the space-time fabric is guaranteed. Also, EQLs enable system designers to express arbitrary pattern-matching schemas, that, to some extent, may support situating events against a general context. For instance, it may be possible to aggregate events about the same "topic", or to correlate events based on some domain-specific property.

However, to achieve greater degrees of situatedness, several authors propose to enhance CEP queries with knowledge semantics leveraging Semantic Complex Event Processing (SCEP), integrating high-level knowledge representation and RDF stream processing. CEP approaches cannot usually semantically interpret and analyse data, that SCEP attempts to address. In [172], for instance, the authors introduce a novel X-CEP query model to enhance CEP queries with knowledge semantics. This model allows users to easily specify event patterns over diverse knowledge concepts in emerging domains, such as IoT and Smart Cities. However, SCEP systems are relatively new and still have limitations to address: one, for instance, is related to their high processing time. On purpose, authors of [40] provide a conceptual model and an implementation of an infrastructure for distributed SCEP (DSCEP), where each SCEP operator can process part of the data and send it to other SCEP operators to parallelize processing and load balance the computational load.

Adaptiveness. Existing CEP systems are mostly based on static rules where the user has to update the rules manually if adaptations become necessary. Experts are often unable to formulate interdependencies and relations between events that are not explicitly known in advance. In addition, in real-time dynamic IoT applications, the context of the application is always changing, thus requiring mechanisms where the rules of the CEP can be optimised automatically. In the literature, there are quite a few examples where researchers have applied machine learning techniques to adapt the rules of

CEP. One such example is given in [94], where authors apply machine learning techniques for rule discovery. In [152], the authors present a mechanism for automating both the initial definition of rules and their updates over time. In [20], a study is proposed on the conjunction between CEP and machine learning, to overcome the problem given by manual definition of rules. There are also efforts toward automatic discovery of CER rules [98, 71, 72], not only adaptation of existing ones.

Opportunistic Deployment. CEP and ESP mostly consider deployment (and scaling, too) within the same infrastructural tier, without means to migrate across the Edge-to-Cloud spectrum. Some work is being conducted to leverage CEP for a mixed Fog-Cloud environment [173], but still without run-time re-deployments. As recognised by recent works [169], however, to enable future IoT applications a CEP system for IoT has to combine the Cloud, the Fog, and the sensors in a single unified platform to leverage their advantages and enable cross-tier optimisations (e.g., fusing, splitting event streams, or operator re-deployment).

There are a few research works that deal with the mechanisms for opportunistic deployment [32]. Even when an optimal combination of CEP operator placement and processing is designed, this decision needs to be opportunistically adapted at runtime, as an optimal combination at the current time may not remain optimal in the long run. Adaptation algorithms should quantify the expected benefit of deployment taking into consideration CEP operator migration costs. For instance, in [112], the authors propose an approach that optimizes operation placement based on the given queries. A general algorithm for task mapping and scheduling (e.g. operator placement) considering resource limitations in wireless sensor networks is formulated in [11]. Some cost functions are introduced in terms of energy consumption in [150, 109]. In [95] the authors make use of workload estimation together with operator-profiling for parallelisation. In [32] authors propose a general CEP system composed of a distributed collaboration of devices at the Edge of the network that allows for the distribution of processing tasks under resource constraints.

3.3.4. Summary

Figure 5 summarises the results of our analysis of the CEP macro-area emerged from our survey, by exploiting our proposed conceptual framework as already done in Subsection 3.2 for SP.

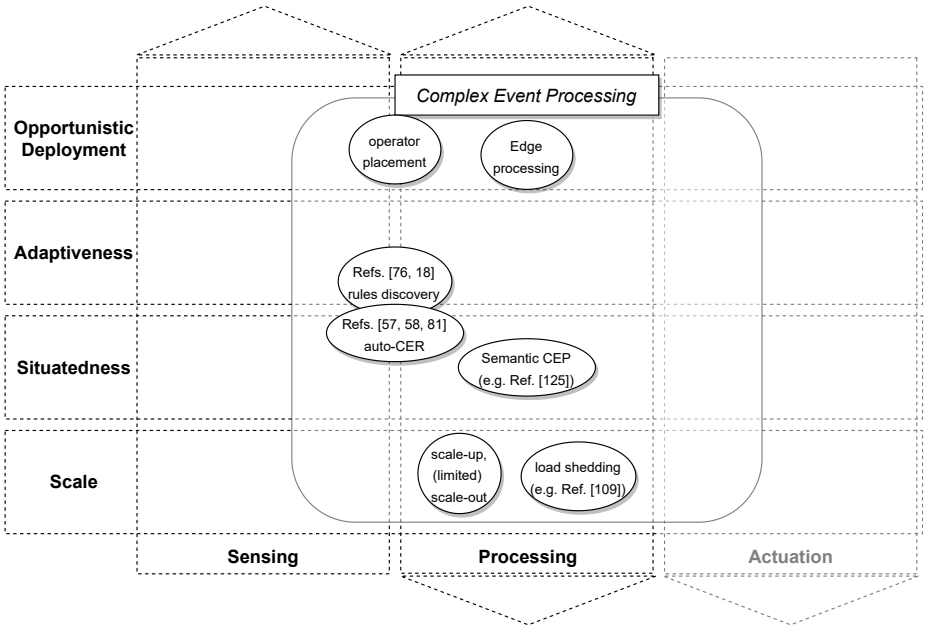


Figure 5: CEP abstractions, mechanisms, and techniques placed in our conceptual framework. The placement of circles is not always crisp as the literature rarely explicitly defines which key feature or processing facet a mechanism supports, hence we devise it out.

Summing up, the situation is better with respect to the case of SP platforms, we speculate due to the advanced processing capabilities of CEP, and their lowest constraints on real-timeliness. Scale independence is well covered by the literature, mostly thanks to Cloud-native elasticity features and CEP load-shedding mechanisms, in a very similar way to SP. Adaptiveness and situatedness have better coverage for CEP than they had for SP: the former may benefit from ongoing research efforts in applying machine learning for the automatic modification and discovery of CEP rules, and the latter is no longer limited to the temporal dimension but may exploit ongoing work about semantic CEP, that can leverage domain-specific relationships amongst events of interest for the application at hand. For opportunistic deployment, instead, there are limitations in line with the SP case: run-time re-deployment of processing components is rarely considered, and when it is, it is never across infrastructural tiers.

3.4. Micro- and Macro- Programming

The distinction between the *micro(-scopic)* and *macro(-scopic)* levels in systems dates back at least to the birth of statistical mechanics. More recently, at least since the 1990s, in the research field of multi-agent systems, terms micro and macro started to be used to distinguish two distinct levels of such a system: the one of individual agents (micro), and the one of groups or *societies* of agents (macro) [135, 159]. Then, as explained in [23], in the early 2000s, term "macro-programming" emerged [103], in the context of wireless sensor networks, to denote programming approaches aimed at capturing collective or global-level behaviour while abstracting micro-level details like routing and communication.

In this work, we decided to keep the micro and macro distinction, for these two levels entail different approaches to programming. In particular, we use term *micro-programming* to denote traditional approaches, mostly based on general-purpose programming languages, aimed at defining the behaviour of a system by adopting the perspective of a single device; in this case, if a system consists of heterogeneous devices, the programmer defines a different program for any different type of device. By contrast, we use term *macro-programming* to denote those approaches, often based on domain-specific languages and declarative paradigms, aimed at specifying the global behaviour of entire networks of devices.

In the rest of the section, we first present related surveys on micro- and macro-programming, then we discuss some key characteristics for understanding corresponding contributions, and finally analyse state-of-the-art contributions on micro- and macro-programming, linking results back to relevant facets and features of our conceptual framework.

3.4.1. Related surveys

To the best of our knowledge, the most comprehensive surveys that *explicitly addresses* IoT programming languages and frameworks are provided in the book chapter by Krishnamurthy et al. [80] and in a specific section of the survey by Dias et al. [43]—see Table 7. The former reviews programming languages specifically for embedded systems (including various flavours of C), message-passing technologies (RPC flavours, COAP, MQTT), coordination languages (Linda, eLinda, Orc, Jolie), and discusses the notion of polyglot programming. The latter, instead, provides a much broader overview of the whole software engineering practice of developing IoT systems. In particular, the overview of development tools provides many useful

Table 7: Main MMP surveys, most recent first.

Ref.	Brief description	Year	Category
[23]	Survey about macro-programming approaches, whatever the application domain	2023	SPEC
[43]	Broad view on the development of IoT systems from the standpoint of software engineering practice	2022	GEN
[69]	Macro-programming, but specifically for the IoT domain	2021	DOM
[81]	Visual programming approaches not only in IoT but education and robotics, too	2021	SPEC
[14]	IoT development platforms, with emphasis on communication and security	2021	GEN
[51]	Review of IoT platforms from an architectural perspective only	2021	SPEC
[123]	DSLs for used in IoT, focus on evaluation methods	2021	SPEC
[88]	Enabling technologies for asynchronous and real-time programming of safety-critical IoT and CPS	2019	SPEC
[17]	Asynchronous programming for IoT and embedded systems	2019	GEN
[116]	High-level study on 13 visual IoT programming languages	2017	GEN
[80]	Focus on distributed programming languages and tools for embedded systems within the IoT ecosystem	2016	SPEC
[110]	Comprehensive features' analysis and classification (node-centric, database, macro, model-driven) of IoT programming approaches	2015	GEN
[139]	Low-level languages (both mainstreams and dialects) for CPS and embedded systems	2015	GEN
[155]	Overview of conventional composition mechanisms for CPSs (monolithic, object-oriented, modular, component-based, service-oriented	2010	SPEC

insights about programming languages and frameworks, such as the kind of programming paradigm promoted (e.g. visual vs. model-driven), that are partially overlapped with our contribution (e.g. they do analyse support for distributed/decentralised programming). However, such surveys do not help researchers and practitioners in understanding which one of the many different programming tools is most suited for different "tasks" (i.e. feature or facet of our framework) in IoT systems development, that is, instead, what our bi-dimensional conceptual framework does.

A more similar work to ours, if restricted to the micro vs. macro categorisation, is by Patel et al. [110], that classifies IoT programming models

into four categories: (i) *node-centric programming*, where the nodes of an IoT system are managed individually (i.e. micro-programming); (ii) *database approach*, where the IoT system is abstracted as a database; (iii) *macro-programming*, where abstractions targeting multiple nodes at once are considered; and (iv) *model-driven development*, where multiple programming perspectives are addressed at once. Also, two surveys specifically targeted at macro-programming have recently been published [23, 69]. They both do an excellent job of sorting out the different existing approaches into a taxonomy. Our survey is somewhat complementary as we provide the additional perspective of our bi-dimensional conceptual framework, which is novel in the literature.

The other literature reviews reported in Table 7 are only marginally related to ours. Both [116] and [81] only examine graphical programming languages, and emphasises usability by non-programmers. In [155], the focus is on the compositionality of software artefacts (e.g. objects, components, services) for CPSs programming. The survey by Lohstroh et al. [88] does include a section on programming models, but only discusses asynchronous and real-time programming. Belson et al. [17] also focus on asynchronous programming, but using co-routines. A similar narrow scope is adopted in [139], where only a few languages (C together with some dialects/extensions, C++, D, Assembly, and Ada) are considered. The brief survey in [123] instead focusses on DSLs and the methods for their evaluation, without deeply discussing the circumstances in which one approach has to be preferred over another, nor how these DSLs fit into the overall process of programming IoT systems. Finally, reference [14] provides some thoughtful discussion about the implications of choosing a kind of language (e.g. DSL vs. mainstream) over another, but is much more focused on analysing IoT execution platforms than programming approaches.

In summary, plenty of surveys are available today analysing "some flavours" of IoT programming, but a comprehensive overview encompassing both micro- and macro-programming is lacking. Moreover, the perspective given by our bi-dimensional conceptual framework is unique, and may aid researchers and practitioners in finding the most suitable approach for their needs.

3.4.2. Main characteristics of MMP

Here follows an overview of the main aspects along which Micro/Macro-Programming (MMP) approaches can be categorised, also reported in Ta-

ble 8. In line with previous sections, this overview is needed to analyse *(i)* the level of support current MMP approaches give to the four key features of modern IoT systems central to our conceptual framework, and *(ii)* whether MMP approaches are mostly dealing with the sensing, processing, or actuation facet of IoT programming.

- *Scope.* *Micro-programming* (or, node-centric) considers one individual device at a time. Such a bottom-up approach, adopted by, e.g., FRASAD [105], COMPOSE [45], NODE-RED [1], and mainstream programming languages such as Java, C, and Python, requires to specify, for each node, its input data, its processing functions, and its actuation commands. Conversely, recent *macro*-programming approaches such as DDFlow [107], PyoT [12], D'Artagnan [97], SmartSociety platform [129], and EdgeProg [85], aim at enabling the programming of an IoT system as a whole, with a global (or, *collective*) perspective.

- *Main Abstraction.* For cloud-based IoT systems, the most recurrent abstraction is the "Service" one: for instance, in COMPOSE [45] behaviours can be modelled as RESTful services. Node-centric IoT systems, instead, mostly leverage on *flow-based* programming abstractions, as networks of "black box" processes communicating through data chunks travelling across predefined connections (or, *wires*). Many are instead the abstractions used for macro-programming, mostly due to the field being relatively new, hence still fragmented amongst the different approaches put forward by different research groups. Some examples are *ensembles* (e.g., [154]), i.e., dynamic groups of devices sharing a goal; *collective interfaces* (e.g., [107]), namely mechanisms to address or refer to a dynamic number of recipients or senders; and *collective tasks* (e.g., [129]), which are operations meant to be carried out collaboratively by a group of devices.

- *Programming Model.* Imperative programming is the most used one, both by micro- and macro-level programming frameworks. Rule-based programming models, such as FRASAD [105], allow the defining of local behaviours of sensor nodes through a set of flexible and extensible rules. A specific nuance of such models is the visual programming model that allows easy implementation of Event-Condition-Action (ECA) rules [15]. COMPOSE is quite peculiar as it allows expressing the behaviour of IoT devices utilising basic logical, string,

and arithmetic operators specified through a JSON document. Programming models for collective behaviours typically rely on a single declarative specification dictating how several distributed components should act and interact. For example, Aggregate programming [16] is a functional, declarative approach for programming the self-organisation logic of networked IoT systems from a global perspective.

- *Architectural/Deployment view.* The majority of the surveyed works follow a standard two-tier architecture where local devices "sense-and-forward" to the Cloud. Only NodeRed, JAMScript and EdgeProg, with the introduction of an intermediate Edge layer, allow also local data processing, shifting to a three-tier architecture. D'Artagnan and Aggregate Computing, instead, promote a different approach: they enable the programming of logical networks of devices and operations, abstracting from the underlying middleware and physical systems, and hence allow expressing IoT systems in a deployment-transparent manner. In such a way, they pave the way to full and opportunistic exploitation of IoT resources, from the edge up to the cloud, with obvious benefits in terms of QoS and QoE.

- *Execution Model.* Most commonly system execution is either *time-driven*, *event-driven*, or *reactive*, which is reasonable, since activity may be triggered by new sensor data or local events. In WSN systems, event-driven and time-driven routing protocols are the main options [165]. Macro-programming approaches typically build on reactive, rather than fully proactive, execution models [23, 69], the latter being covered mostly in agent-oriented paradigms [19].

3.4.3. Features and facets analysis

Amongst the macro areas of research that emerged from our survey, micro and macro-programming approaches to IoT systems development are the ones concerned the most with the actuation facet, in the sense of programming devices to accomplish tasks. Whereas both SP and CEP are mostly developed with data collection and processing, respectively, in mind, and cannot be used as general programming languages executed on-board any device, the programming approaches in this latter broad category are specifically conceived to close this gap. Being general-purpose, these approaches

Table 8: Summary of the MMP main characteristics.

	Scope	Main Abstraction	Programming Model	Architectural/ Deployment View	Execution Model
FRASAD [105]	Individual	Reaction Rules, Messages	Visual block language	Local WSN	Reactive
SMART-BLOCK [15]	Individual	Reaction Rules	Visual block language	Cloud-based	Reactive
COMPOSE [45]	Individual	Smart Objects, Resource (REST)	Imperative, object-oriented DSL	Cloud-based	Customizable
Node-Red [1]	Individual	Data Flow	Visual block language	Edge-to-Cloud	Customizable
JAMScript [157]	Individual	Nodes, Activities	Imperative, object-oriented DSL	Edge-to-Cloud	Reactive
PyoT [12]	Collective	Resource (REST)	Imperative, object-oriented, macroprogramming DSL	Master-worker	Reactive
DDFlow [107]	Collective	Data Flow	Visual block language, macroprogramming	Master-worker	Reactive
D'Artagnan [97]	Collective	Data Stream, Data Flow	Functional, stream-based, macroprogramming DSL	Logical network any deployment	Reactive
SmartSociety [129]	Collective	Collective task	Object-oriented DSL	Cloud-based	Customizable
EdgeProg [85]	Collective	Reaction rules, Virtual sensors	Rule-based DSL	Edge server + IoT nodes	Reactive
Aggregate Computing [154]	Collective	Computational Field	Functional declarative macroprogramming	Logical network any deployment	Self-organizing

may be used for data collection and processing too, but doing so is likely "reinventing the wheel" as SP and CEP are readily available.

As regards the four key features, the following paragraphs thoroughly discuss their coverage as emerging from the surveyed literature.

Scale-independence. Macro-programming languages are scale-independent by definition: the behaviour of the system is collectively defined, no matter what the number of the devices is, and often no matter their deployment topology and/or geographical distribution. This allows works such as DDFlow [107], PyoT [12], D'Artagnan [97], SmartSociety platform [129] and EdgeProg [85] to be programmed in a scale-transparent way. However, they are not immune to bottleneck effects that might concern the performance of cloud-based approaches that heavily rely on transferring data to the Cloud for synthesising commands to be sent back to actuators (or generic devices).

Adaptiveness. The majority of the surveyed works adopt the ECA paradigm, thus inherently adapt their behaviour to events, property modifications, etc. In particular, SmartBlock and NodeRed among the node-centric works, and PyoT and DDFlow among the collective ones, deliver greater adaptiveness thanks to highly expressive rule engines and event listeners, that allows to better react to contextual changes such as network traffic or computation overload and related device failures. However, the same limitations already

discussed for SP (Section 3.2) and CEP (Section 3.3) for other approaches apply here as well: first, rules are most of the time defined once and for at design time, based on events and properties statically specified by domain experts; second, the adaptation of such rules to the application or domain-specific properties and events remain a challenge for most approaches.

Situatedness. The accurate management of a wide variety of sensors (temperature, position, etc.) makes all the surveyed works actually situated in the time-space fabric and/or their environment. Such sensors might be either physically or virtually available but, in any case, allow the programming of situation-aware IoT applications also thanks to first-class abstractions such as "Region", "Resource", "Status", "Neighbourhood", etc. Moreover, semantic technologies are gaining traction to provide a greater degree of situatedness, especially in Web of Things-based systems such as NodeRed and SmartBlock, that incorporate the semantic definitions developed by iot.schema.org [149] and the IoTA calculus [102], respectively. Although this trend is still far from being consolidated, it shows promising interest in expanding the notion of situatedness beyond its current space-time limitations.

Opportunistic Deployment. JAMScript and Node-red extensions [138] allow the programming of applications that dynamically migrate across the Edge-to-Cloud spectrum. This is done, in both cases, through a particular orchestration language and a runtime for supporting its execution. Conversely, works like D'Artagnan, EdgeProg, DDFlow, and Aggregate Computing limit opportunistic deployment to the same architectural layer. DDFlow allows dynamic deployment of the application depending on the available network resources, so to minimize end-to-end latency: in the case of significant network changes, or device failure, DDFlow will reconfigure deployment to preserve application semantics by re-mapping the computational processes onto the network and/or switching to alternative networking protocols. D'Artagnan makes use of inter-node orchestration to dynamically deploy and migrate applications for achieving the desired functional and non-functional goals. Similarly, EdgeProg leverages dynamic linking and loading to deploy the source code of an application on a variety of IoT devices or Edge servers on demand. Finally, the Aggregate Computing paradigm, with its deployment-transparent approach [26], enables the opportunistic exploitation of IoT resources across the device-edge-cloud spectrum [24].

3.4.4. Summary

Figure 6 summarises the results of our analysis of the MMP macro-area with respect to our proposed conceptual framework.

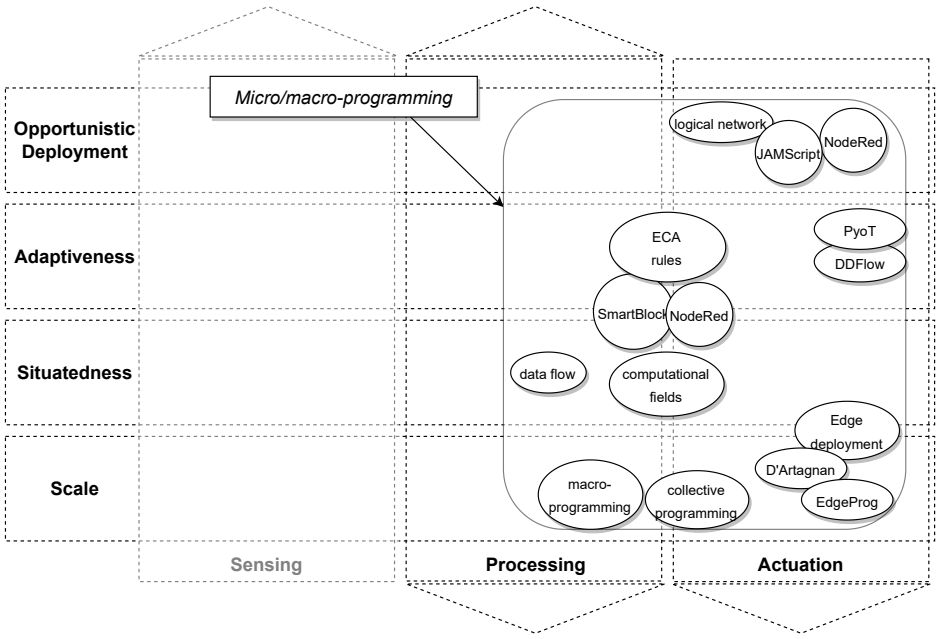


Figure 6: MMP abstractions, mechanisms, and techniques placed in our conceptual framework. The placement of circles is not always crisp as the literature rarely explicitly defines which key feature or processing facet a mechanism supports, hence we devise it out.

Mostly because the MMP category gathers together many different approaches, most of which are general-purpose programming platforms tailored and enhanced to support IoT deployments, MMP approaches collectively have a great coverage of the four key features we focus on in this survey. For instance, under the umbrella term of "collective adaptive systems" lie many approaches specifically aimed at delivering scale independence and adaptiveness. These two features are the most well-covered by the surveyed literature, when considering that application and domain-specific adaptations are gaining attention by resorting to semantics-aware data processing. Situatedness and opportunistic deployment are less represented, but have notable exceptions: for the former, the whole research thread of Aggregate Computing is specifically meant to deliver application and domain-specific situatedness by giving IoT systems designers full control over sensing and actuation capabili-

ties in a fully decentralised way; for the latter, a whole bunch of platforms and frameworks are starting to support cross-tier (re)deployment of processing.

4. Discussion & open challenges

Based on the analysis of the literature about IoT programming just presented in Sections 3.2, 3.3, and 3.4, a few considerations can be made to give a birds-eye view of the whole research landscape, crossing borders across the macro-areas just presented.

The most obvious consideration is that "programming IoT" may actually translate to quite a different activity depending on the specific goals of the application to be built. On the one hand, IoT applications are especially interested in monitoring a given system, for example to provide a dashboard to human supervisors, are usually built by taking off-the-shelf SP or CEP platforms (depending on the sophistication of the required data processing). In this case, "programming" may actually translate to configuring the logical DAG for data processing, that is deciding which operators to apply to what data, and how to put operators in a pipeline, or defining patterns and rules for triggering computations upon detection of a given pattern of events. On the other hand, applications may also be interested in closing the feedback loop between the physical (the things) and the digital (the processing) by sending commands back to the (actuator) devices. In this case, that could regard a swarm robotics scenario, for instance, "programming" usually translates more directly to the traditional programming experience, using mainstream languages (for micro-programming approaches) or Domain Specific Languages (more common for macro-programming approaches). This is to say that the "programming experience" may be quite different depending on the kind of IoT systems one wants to develop.

A huge impact on this programming experience is also given by the fragmentation of abstractions across the research landscape. If in the SP and CEP areas, the abstractions may be somewhat similar, such as the concept of "stream" (of events or data), operator, composition, pattern, etc, and in the micro-programming area abstractions are often the same (object, function, thread), in the realm of macro-programming almost every approach has its own set of abstractions (field, collective, ensemble, etc.). Mappings can be devised, but integration of the different approaches in a single software project nevertheless suffers, as developers need to master different paradigms. A recent effort in this direction of providing re-usable abstractions across re-

search fields is done by the Fluidware framework [168], which fosters event streams and "funnel processes" (as sort of distributed, high-level programs manipulating events) as primary abstractions for implementing highly dynamic and scalable IoT systems.

However, such an abundance of approaches also has upsides: the conceptual framework we conceived is covered in most of its crossroads between the facets (vertical slices) and the key features (horizontal slices). Since each programming approach is conceived with a specific set of functional and non-functional desiderata, each approach ends up covering some specific "slot" in our conceptual framework. Taken altogether, thus, they are quite complementary.

Nevertheless, to be fully harnessed, such complementarity must be exploited in an integrated framework, not by patching together different approaches on an ad-hoc basis. Although our conceptual framework is well covered by the surveyed literature, it is a matter of fact that it is currently not easy at all for a programmer to have all the four key features we highlighted well supported by a single, coherent programming platform.

In the following, we discuss some challenges that horizontally apply to the whole surveyed literature, that we believe must be dealt with to enable next-generation IoT systems and applications—challenges peculiar to each area are already summarised in the surveyed papers.

4.1. Automation vs. adaptation tradeoff

Deploying, managing, upgrading, and similar operations common in software systems are especially challenging in the case of IoT applications and systems. Not only because such systems are usually composed of multiple physical resources that interact in multiple ways, but also because each and every one of these resources may come along with its own technical peculiarities to deal with, such as different communication protocols, software stack, constraints on computational power, etc. Therefore, it is natural that one main goal of the surveyed IoT programming approaches is to provide a middleware able to hide this complexity by automating tasks such as protocol translation, caching, fault tolerance, etc. to the programmer, to let him focus on the application logic as much as possible [31].

However, such automation comes at a price in terms of flexibility. In all those cases where the application logic depends on or should affect some middleware-level concern (e.g. switching prediction models depending on available computational resources, or migrating computations across the

Edge-to-Cloud layers depending on network congestion) such automation may get in the way of the programmer. In fact, for a middleware to automate application-level tasks in a general way, that is, irrespective of the task and the application domain, such tasks must be "black boxes" from the perspective of the middleware. This is what happens with operator placement in Stream Processing and Complex Event Processing, for instance: the middleware (or, programming platform) sets the API to use and the application adheres. In those cases where further adaptation to application-level properties is needed, keeping the same level of automation is not trivial at all [49]. In fact, for instance in the case SP or CEP platforms, although most of the platforms surveyed support custom operators, they explicitly warn the programmer that such operators would not benefit from services such as auto-scaling, parallelisation, and such, as they are operated by the middleware "as-is".

4.2. Infrastructure lock-in

Many non-trivial IoT systems rely on a multi-tier infrastructure to provide their services: some with 2 tiers, the Cloud and the Edge, and some with 3 tiers, including a Fog layer in-between. Some IoT middleware and programming platforms also give the programmer a uniform API to access the same services regardless of whether they are delivered at the Edge or in the Cloud. However, very few provide ways to migrate services at run-time across tiers, but are limited to intra-tier migration (e.g., Cloud to Cloud, or Edge to Edge). Even fewer contributions allow inter-tier migration and also opportunistically re-configure the migrated components so as to either exploit the novel computing opportunities offered by the destination tier [18] (e.g. more compute in case of the Cloud, or additional services such as geolocation-based at the Edge).

Instead, migrating services across tiers could enable a better exploitation of the peculiar properties of each different tier. For instance, at the Edge, there is usually ultra-low latency but also limited bandwidth, hence a service migrated from the Cloud must be reconfigured properly to these new properties, which are likely the opposite of the starting tier (e.g. on Cloud there is usually high latency but plenty of bandwidth). Some IoT Cloud platforms offer offloading of computations across tiers, which is surely a step in the right direction, but usually, such offloading follows a precise direction: from Edge to Cloud (e.g. to save energy or gather more compute power).

All in all, these limitations have the consequence that IoT application programmers are locked into a specific infrastructure at deployment time: they must know and choose, at design time, in which tier their application will be executing, as that will most likely be the tier its will be locked for its whole execution time. Opportunistic deployment, instead, would foster seamless migration from Edge to Cloud [161] (and back) based on both infrastructural (e.g. latency, available services, computing power) and application-specific criteria (e.g. need for local vs. global information, users proximity, real-time requirements).

4.3. Application-specific situatedness

IoT systems and applications are usually situated software systems by definition: they are interested in capturing data coming from the real world, processing such data to make decisions based on context (e.g. when, where, the data belongs to), and affect back the portion of the real-world they have control over. CEP platforms, for instance, usually provide APIs to correlate events based on spatial and temporal patterns, and usually aggregate events into complex hierarchies by relying on such patterns. Such APIs are usually domain-agnostic: to apply to a wide range of applications, they consider general-purpose spatiotemporal aspects such as proximity of events in time and space. However, each and every application domain may have its own criteria for defining high-level events as combinations of low-level ones. Capturing such criteria requires ad-hoc implementations similar to the discussion above about the automation-adaptation trade-off: when limiting themselves to the provided APIs, programmers gain the benefit of automated events correlation, but if they want more control over correlation criteria and mechanisms, they must explicitly implement the application logic defining them.

This it not to mention the intricacies of affecting the real world in turn, where each action that is automated is again situated by definition: it must account for the specific context within which such action is being carried out. The action stage of the "actionable knowledge" loop is, not by chance, often the less supported by SP and CEP platforms, and while amongst the micro and macro programming approaches surveyed, there are many that offer the desired level of application-specific situatedness [101] (for instance, aggregate computing is specifically conceived also for this use case), each comes with its own peculiar set of abstractions and reference computational paradigm [128] (e.g. computational fields manipulation in the case of aggregate computing). This makes integrating them into mainstream IoT platforms quite difficult.

4.4. Fragmentation of abstractions

As already discussed in Section 4, one characteristic of the research landscape about IoT programming that emerges from this comprehensive survey is that most of our conceptual framework is covered, in the sense that there are approaches available trying to deliver a given feature with respect to a given facet. However, what is also apparent from this survey is that there is no "one to rule them all" approach here, as different regions of our conceptual frameworks are covered by different approaches, often belonging to entirely different areas. Also, even within a single area, there may be different approaches to do similar things, which is especially true for micro/macro-programming cases. There, for instance, IoT programming languages, platforms, and paradigms are mostly unique and come with their own set of abstractions, mechanisms, and even computational models (as in the case of aggregate programming or agent-based technology). This leads to fragmentation of abstraction, models, and tools [29].

On the one hand, this means that the technical tooling to exhaustively deal with all the features and facets is mostly there, but that does not imply that it is easy or even possible to tackle all of them at once. Integration of any two approaches usually requires a great deal of effort in (i) understanding the operating principles behind each approach, (ii) finding suitable mappings between abstractions and mechanisms, (iii) designing efficient integrations that are both "natural" to use for programmers coming from either side of the integration, and also efficient.

4.5. Controlled learning

As with almost any other computer science and engineering field of study, nowadays, IoT is increasingly pervaded by Artificial Intelligence (AI) approaches aimed at the most disparate goals such as optimisation, prediction, adaptation [171]. The IoT has been one of the driving forces in AI development, due to the sheer amount of data it made available for training statistical models, for instance [124]. Most AI approaches used in IoT are, in fact, data-driven approaches: they aim at learning a model of behaviour (be it prediction, optimisation, anomaly detection, or whatever) purely from observational data about the phenomenon of interest.

Such data-driven approaches bring along controllability issues [121], regardless of the application domain: how to guarantee that the learnt behaviour stays within some application-specific safety boundaries? Note that

safety here has its broadest meaning possible: it can refer to moral implications (e.g. denying loans based on gender or ethnic group), health risk (e.g. an AI should avoid actions that could harm people no matter what), or technical issues (e.g. operating equipment in unconventional ways). In the specific realms of the IoT such issues are even more relevant than in other domains (e.g. gaming or trading, for instance), as the "actionable knowledge" feedback loop closes when the IoT software system can autonomously affect the portion of the real world it is perceiving in turn. The risk of carrying out dangerous practical actions, here, is high (e.g. mismanagement of a mobile vehicle).

A long-standing issue in AI is guaranteeing controllability and predictability also in the case of autonomous behaviours, such as those achieved by data-driven machine learning approaches or agent-oriented technology. Promising work is being done in various fields such as adjustable autonomy [130], fairness and safety in AI [162], that should be transferred to the IoT domain as soon as possible [76].

5. Summary and Future Scope

In this survey paper, we provided a comprehensive overview of the research landscape in IoT programming, broadly intended to include the programming "tools" (abstractions, languages, platforms, middleware) that researchers and practitioners can exploit to program their IoT systems' working logic. Despite the many surveys available on the topic to date, as witnessed by Tables 3,5,7, this is the first of its kind to provide a reference conceptual framework that *(i)* comprehensively covers the vast research landscape on IoT programming without a-priori filters (e.g. on techniques, applications, architectures, etc.), and *(ii)* clearly positions IoT programming approaches along engineering facets (Section 2.1) and modern applications' requirements (Section 2.2).

By doing so, researchers and practitioners are greatly facilitated in carrying out two essential tasks: (i) decompose their IoT system at hand along the three facets of computation, making its design easier to define, and (ii) find which programming tools can currently support the desired non-functional properties at best. In fact, we visually placed contributions in the literature within such a framework, also getting the added benefits of *(i)* quickly uncovering the research areas mostly involved in IoT programming, such as SP,

CEP, micro-, and macro-programming, as well as *(ii)* clearly highlighting the open issues and challenges not currently solved by the available literature.

To provide further assistance to readers, we categorised the literature resulting from our review in three macro areas of research (corresponding to Sections 3.2,3.3,3.4), and have deepened our analysis of the IoT programming models, languages, and platforms available to date in those areas. Since we carefully described the main properties of the approaches available in a given area, it becomes easy to further browse the literature in a specific area of interest to find the desired specific approach. Finally, we provided a summarising discussion about the "bird-eye-view" considerations that can be made on the basis of this literature review, especially describing several cross-cutting research challenges that remain to be addressed, stimulating further research in related areas.

Given all the above, we believe the presented survey can be a good "compass" guiding researchers and practitioners in IoT programming to either find new research challenges to deal with, or novel approaches to readily exploit in their IoT systems.

Also, this survey can provide a solid ground to kick-start future surveys on the same topic or to further deepen the analysis of either the proposed conceptual framework or the macro-area of research that emerged. Future surveys can indeed rely on this one to get a quick overview of the main references in the IoT programming literature, by simply checking out the works mentioned in Tables 3,5,7. Then, they can build upon these by only covering the newer literature. The conceptual framework we described has been conceived to stay relevant for many years to come, as it is based on staple engineering facets, and considers key features that are still open issues to date. Hence, novel literature can readily fit within the same framework at no cost. However, in the case novel key features become relevant, our conceptual framework can still work as a practical guideline to organise and analyse those new features in a new framework. Summing up, we believe that the presented survey can effectively serve as a blueprint for future surveys in the field.

Acknowledgements

This work has been supported by the MIUR PRIN 2017 Project "Fluidware" (N. 2017KRC7KT) and the EU/MUR FSE REACT-EU PON R&I 2014-2020.

References

[1] 2024. Node-Red. https://nodered.org/. [Online; accessed 15-April-2024].

[2] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Çetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stanley B. Zdonik. 2005. The Design of the Borealis Stream Processing Engine. In *Second Biennial Conference on Innovative Data Systems Research*. www.cidrdb.org. http://cidrdb.org/cidr2005/papers/P23.pdf

[3] Daniel J. Abadi, Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, C. Erwin, Eduardo F. Galvez, M. Hatoun, Anurag Maskey, Alex Rasin, A. Singer, Michael Stonebraker, Nesime Tatbul, Ying Xing, R. Yan, and Stanley B. Zdonik. 2003. Aurora: A Data Stream Management System. In *ACM SIGMOD International Conference on Management of Data*. https://doi.org/10.1145/872757.872855

[4] Asaf Adi and Opher Etzion. 2004. Amit - the situation manager. *VLDB Journal* (2004). https://doi.org/10.1007/s00778-003-0108-y

[5] Adnan Akbar, François Carrez, Klaus Moessner, Juan Sancho, and Juan Rico. 2015. Context-aware stream processing for distributed IoT applications. In *2nd IEEE World Forum on Internet of Things*. https://doi.org/10.1109/WF-IoT.2015.7389133

[6] Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras. 2017. Probabilistic Complex Event Recognition: A Survey. *Comput. Surveys* (2017). https://doi.org/10.1145/3117809

[7] Mohamed Ali, Badrish Chandramouli, Jonathan Goldstein, and Roman Schindlauer. 2011. The extensibility framework in Microsoft StreamInsight. In *IEEE 27th International Conference on Data Engineering*.

[8] Chinnapong Angsuchotmetee and Richard Chbeir. 2016. A survey on complex event definition languages in multimedia sensor networks. In

8th International Conference on Management of Digital EcoSystems. ACM.

[9] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Rajeev Motwani, Itaru Nishizawa, Utkarsh Srivastava, Dilys Thomas, Rohit Varma, and Jennifer Widom. 2003. STREAM: The Stanford Stream Data Manager. *IEEE Data Eng. Bull.* 26, 1 (2003), 19–26. http://sites.computer.org/debull/A03mar/paper.ps

[10] Alexander Artikis, Anastasios Skarlatidis, François Portet, and Georgios Paliouras. 2012. Logic-based event recognition. *Knowledge Engineering Review* (2012). https://doi.org/10.1017/S0269888912000264

[11] MHA Awadalla. 2013. Task mapping and scheduling in wireless sensor networks. *IAENG International Journal of Computer Science* (2013).

[12] Andrea Azzara, Daniele Alessandrelli, Stefano Bocchino, Matteo Petracca, and Paolo Pagano. 2014. PyoT, a macroprogramming framework for the Internet of Things. In *9th IEEE International Symposium on Industrial Embedded Systems*. https://doi.org/10.1109/SIES.2014.6871193

[13] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. 2002. Models and Issues in Data Stream Systems. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. https://doi.org/10.1145/543613.543615

[14] Leonardo Babun, Kyle Denney, Z. Berkay Celik, Patrick McDaniel, and A. Selcuk Uluagac. 2021. A survey on IoT platforms: Communication, security, and privacy perspectives. *Computer Networks* (2021). https://doi.org/10.1016/j.comnet.2021.108040

[15] Nayeon Bak, Byeong-Mo Chang, and Kwanghoon Choi. 2020. Smart Block: A visual block language and its programming environment for IoT. *Journal of Computer Languages* (2020). https://doi.org/10.1016/j.cola.2020.100999

[16] Jacob Beal, Danilo Pianini, and Mirko Viroli. 2015. Aggregate Programming for the Internet of Things. *Computer* 48, 9 (2015), 22–30. https://doi.org/10.1109/MC.2015.261

[17] Bruce Belson, Jason Holdsworth, Wei Xiang, and Bronson Philippa. 2019. A Survey of Asynchronous Programming Using Coroutines in the Internet of Things and Embedded Systems. *ACM Trans. Embed. Comput. Syst.* 18, 3 (2019), 21:1–21:21. https://doi.org/10.1145/3319618

[18] Luiz F. Bittencourt, Roger Immich, Rizos Sakellariou, Nelson L. S. da Fonseca, Edmundo R. M. Madeira, Marília Curado, Leandro Villas, Luiz A. DaSilva, Craig A. Lee, and Omer F. Rana. 2018. The Internet of Things, Fog and Cloud continuum: Integration and challenges. *Internet Things* 3-4 (2018), 134–155. https://doi.org/10.1016/J.IOT.2018.09.005

[19] Olivier Boissier, Rafael H Bordini, Jomi Hubner, and Alessandro Ricci. 2020. *Multi-agent oriented programming: programming multi-agent systems using JaCaMo*. Mit Press.

[20] Ralf Bruns, Jürgen Dunkel, and Norman Offel. 2019. Learning of complex event processing rules with genetic programming. *Expert Systems Application* (2019). https://doi.org/10.1016/j.eswa.2019.04.007

[21] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* (2015).

[22] Roberto Casadei. 2023. Artificial Collective Intelligence Engineering: A Survey of Concepts and Perspectives. *Artificial Life* (2023). https://doi.org/10.1162/ARTL_A_00408

[23] Roberto Casadei. 2023. Macroprogramming: Concepts, State of the Art, and Opportunities of Macroscopic Behaviour Modelling. *Comput. Surveys* (2023). https://doi.org/10.1145/3579353

[24] Roberto Casadei, Giancarlo Fortino, Danilo Pianini, Andrea Placuzzi, Claudio Savaglio, and Mirko Viroli. 2022. A Methodology and Simulation-Based Toolchain for Estimating Deployment Performance of Smart Collective Services at the Edge. *IEEE Internet Things Journal* (2022). https://doi.org/10.1109/JIOT.2022.3172470

[25] Roberto Casadei, Giancarlo Fortino, Danilo Pianini, Wilma Russo, Claudio Savaglio, and Mirko Viroli. 2019. A development approach for collective opportunistic Edge-of-Things services. *Information Sciences* (2019). https://doi.org/10.1016/j.ins.2019.05.058

[26] Roberto Casadei, Danilo Pianini, Andrea Placuzzi, Mirko Viroli, and Danny Weyns. 2020. Pulverization in Cyber-Physical Systems: Engineering the Self-Organizing Logic Separated from Deployment. *Future Internet* (2020). https://doi.org/10.3390/FI12110203

[27] Everton Cavalcante, Marcelo Pitanga Alves, Thais Batista, Flavia Coimbra Delicato, and Paulo F. Pires. 2015. An Analysis of Reference Architectures for the Internet of Things. In *Proceedings of the 1st International Workshop on Exploring Component-Based Techniques for Constructing Reference Architectures* (Montréal, QC, Canada) *(CobRA '15)*. Association for Computing Machinery, New York, NY, USA, 13–16. https://doi.org/10.1145/2755567.2755569

[28] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Frederick Reiss, and Mehul A. Shah. 2003. TelegraphCQ: Continuous Dataflow Processing. In *ACM SIGMOD International Conference on Management of Data*. https://doi.org/10.1145/872757.872857

[29] Lu Chao, Xiaohui Peng, Zhiwei Xu, and Lei Zhang. 2019. Ecosystem of Things: Hardware, Software, and Architecture. *IEEE Proceedings* (2019). https://doi.org/10.1109/JPROC.2019.2925526

[30] Moumena A. Chaqfeh and Nader Mohamed. 2012. Challenges in middleware solutions for the internet of things. In *International Conference on Collaboration Technologies and Systems*. https://doi.org/10.1109/CTS.2012.6261022

[31] Tianyi Chen, Sergio Barbarossa, Xin Wang, Georgios B. Giannakis, and Zhi-Li Zhang. 2019. Learning and Management for Internet of Things: Accounting for Adaptivity and Scalability. *Proc. IEEE* 107, 4 (2019), 778–796. https://doi.org/10.1109/JPROC.2019.2896243

[32] Sunyanan Choochotkaew, Hirozumi Yamaguchi, Teruo Higashino, Megumi Shibuya, and Teruyuki Hasegawa. 2017. EdgeCEP: Fully-Distributed Complex Event Processing on IoT Edges. In *13th International Conference on Distributed Computing in Sensor Systems, DCOSS 2017, Ottawa, ON, Canada, June 5-7, 2017*. IEEE, 121–129. https://doi.org/10.1109/DCOSS.2017.14

[33] Diane J. Cook and Sajal K. Das. 2005. *Smart environments - technology, protocols and applications*. Wiley.

[34] Fulvio Corno, Luigi De Russis, and Juan Pablo Sáenz. 2019. On the challenges novice programmers experience in developing IoT systems: A Survey. *Journal of Systems and Software* (2019). https://doi.org/10.1016/j.jss.2019.07.101

[35] Gianpaolo Cugola and Alessandro Margara. 2009. RACED: an adaptive middleware for complex event detection. In *8th Workshop on Adaptive and Reflective Middleware*. ACM. https://doi.org/10.1145/1658185.1658190

[36] Gianpaolo Cugola and Alessandro Margara. 2012. Complex event processing with T-REX. *Journal of Systems and Software* (2012). https://doi.org/10.1016/j.jss.2012.03.056

[37] Gianpaolo Cugola and Alessandro Margara. 2012. Processing flows of information: From data stream to complex event processing. *Comput. Surveys* (2012). https://doi.org/10.1145/2187671.2187677

[38] Gianpaolo Cugola and Alessandro Margara. 2015. *The Complex Event Processing Paradigm*. Springer International Publishing. https://doi.org/10.1007/978-3-319-20062-0_6

[39] Miyuru Dayarathna and Srinath Perera. 2018. Recent Advancements in Event Processing. *Comput. Surveys* (2018). https://doi.org/10.1145/3170432

[40] Vitor Pinheiro de Almeida, Sukanya Bhowmik, Markus Endler, and Kurt Rothermel. 2020. DSCEP: An Infrastructure for Distributed Semantic Complex Event Processing. *CoRR* (2020). https://arxiv.org/abs/2002.05869

[41] Luca De Nardis, Alireza Mohammadpour, Giuseppe Caso, Usman Ali, and Maria-Gabriella Di Benedetto. 2022. Internet of Things Platforms for Academic Research and Development: A Critical Review. *Applied Sciences* (2022). https://doi.org/10.3390/app12042172

[42] Alan J. Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, and Walker M. White. 2007. Cayuga: A General Purpose Event Monitoring System. In *Third Biennial Conference on Innovative Data Systems Research*. www.cidrdb.org.

[43] João Pedro Dias, André Restivo, and Hugo Sereno Ferreira. 2022. Designing and constructing internet-of-Things systems: An overview of the ecosystem. *Internet Things* (2022). https://doi.org/10.1016/j.iot.2022.100529

[44] Marcos Dias de Assunção, Alexandre da Silva Veith, and Rajkumar Buyya. 2018. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications* 103, July 2017 (2018), 1–17. https://doi.org/10.1016/j.jnca.2017.12.001 arXiv:1709.01363

[45] Charalampos Doukas and Fabio Antonelli. 2013. COMPOSE: Building smart & context-aware mobile applications utilizing IoT technologies. In *Global Information Infrastructure Symposium-GIIS 2013*. IEEE. https://doi.org/10.1109/giis.2013.6684373

[46] Michael Eckert, François Bry, Simon Brodt, Olga Poppe, and Steffen Hausmann. 2011. A CEP Babelfish: Languages for Complex Event Processing and Querying Surveyed. In *Studies in Computational Intelligence*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-19724-6_3

[47] Cristofer Englund, Lei Chen, Jeroen Ploeg, Elham Semsar-Kazerooni, Alexey Voronov, Hoai Hoang Bengtsson, and Jonas Didoff. 2016. The Grand Cooperative Driving Challenge 2016: boosting the introduction of cooperative automated vehicles. *IEEE Wireless Communications* (2016). https://doi.org/10.1109/MWC.2016.7553038

[48] Sergi Esteves, Nico Janssens, Bart Theeten, and Luis Veiga. 2017. Empowering Stream Processing through Edge Clouds. *ACM SIGMOD Record* (2017). https://doi.org/10.1145/3156655.3156661

[49] Kaneez Fizza, Abhik Banerjee, Prem Prakash Jayaraman, Nitin Auluck, Rajiv Ranjan, Karan Mitra, and Dimitrios Georgakopoulos. 2023. A Survey on Evaluating the Quality of Autonomic Internet of Things Applications. *IEEE Commun. Surv. Tutorials* 25, 1 (2023), 567–590. https://doi.org/10.1109/COMST.2022.3205377

[50] Ioannis Flouris, Nikos Giatrakos, Antonios Deligiannakis, Minos Garofalakis, Michael Kamp, and Michael Mock. 2017. Issues in complex event processing: Status and prospects in the Big Data era. *Journal of Systems and Software* (2017). https://doi.org/10.1016/j.jss.2016.06.011

[51] Giancarlo Fortino, Antonio Guerrieri, Claudio Savaglio, and Giandomenico Spezzano. 2021. A Review of Internet of Things Platforms Through the IoT-A Reference Architecture. In *14th International Symposium on Intelligent Distributed Computing (Studies in Computational Intelligence)*. Springer. https://doi.org/10.1007/978-3-030-96627-0_3

[52] Marios Fragkoulis, Paris Carbone, Vasiliki Kalavri, and Asterios Katsifodimos. 2020. A Survey on the Evolution of Stream Processing Systems. *CoRR* (2020). https://arxiv.org/abs/2008.00842

[53] Frieder Ganz, Daniel Puschmann, Payam M. Barnaghi, and François Carrez. 2015. A Practical Evaluation of Information Processing and Abstraction Techniques for the Internet of Things. *IEEE Internet Things J.* 2, 4 (2015), 340–354. https://doi.org/10.1109/JIOT.2015.2411227

[54] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. 2020. Complex event recognition in the Big Data era: a survey. *VLDB Journal* (2020). https://doi.org/10.1007/s00778-019-00557-w

[55] Nikos Giatrakos, Eleni Kougioumtzi, Antonios Kontaxakis, Antonios Deligiannakis, and Yannis Kotidis. 2021. EasyFlinkCEP: Big Event Data Analytics for Everyone. In *30th ACM International Conference on Information and Knowledge Management*. https://doi.org/10.1145/3459637.3482094

[56] Z. Guessoum. 2004. Adaptive agents and multiagent systems. *IEEE Distributed Systems Online* (2004). https://doi.org/10.1109/MDSO.2004.10

[57] TIBCO BusinessEventsUser's Guide. 2005. TIBCO® BusinessEvents. *Software Release* (2005).

[58] Bin Guo, Zhiwen Yu, Xingshe Zhou, and Daqing Zhang. 2012. Opportunistic IoT: Exploring the social side of the internet of things. In *IEEE 16th International Conference on Computer Supported Cooperative Work in Design*. https://doi.org/10.1109/CSCWD.2012.6221932

[59] Jasmin Guth, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Lukas Reinfurt. 2016. Comparison of IoT platform architectures: A field study based on a reference architecture. In *2016 Cloudification of the Internet of Things, CIoT 2016, Paris, France, November 23-25, 2016*. IEEE, 1–6. https://doi.org/10.1109/CIOT.2016.7872918

[60] Daniel Gyllstrom, Eugene Wu, Hee-Jin Chae, Yanlei Diao, Patrick Stahlberg, and Gordon Anderson. 2007. SASE: Complex Event Processing over Streams (Demo). In *Third Biennial Conference on Innovative Data Systems Research*. www.cidrdb.org.

[61] Jane Hillston, Jeremy Pitt, Martin Wirsing, and Franco Zambonelli. 2014. Collective Adaptive Systems: Qualitative and Quantitative Modelling and Analysis (Dagstuhl Seminar 14512). *Dagstuhl Reports* (2014). https://doi.org/10.4230/DagRep.4.12.68

[62] Martin Hirzel, Robert Soulé, Scott Schneider, Bugra Gedik, and Robert Grimm. 2013. A catalog of stream processing optimizations. *Comput. Surveys* (2013). https://doi.org/10.1145/2528412

[63] EsperTech Inc. 2021. *ESPER*. https://www.espertech.com/esper/

[64] Red Hat Inc. 2021. *Open CEP*. https://github.com/ilya-kolchinsky/OpenCEP/

[65] STRATIO BIG DATA Inc. 2021. *Decision CEP Engine*. https://github.com/Stratio/Decision#decision-cep-engine

[66] Haruna Isah, Tariq Abughofa, Sazia Mahfuz, Dharmitha Ajerla, Farhana H. Zulkernine, and Shahzad Khan. 2019. A Survey of Distributed Data Stream Processing Frameworks. *IEEE Access* (2019). https://doi.org/10.1109/ACCESS.2019.2946884

[67] S. M. Riazul Islam, Daehan Kwak, MD. Humaun Kabir, Mahmud Hossain, and Kyung-Sup Kwak. 2015. The Internet of Things for Health Care: A Comprehensive Survey. *IEEE Access* (2015). https://doi.org/10.1109/ACCESS.2015.2437951

[68] Farhana Javed, Muhammad Khalil Afzal, Muhammad Sharif, and Byung-Seo Kim. 2018. Internet of Things (IoT) Operating Systems Support, Networking Technologies, Applications, and Challenges: A Comparative Review. *IEEE Communications Surveys and Tutorials* (2018). https://doi.org/10.1109/COMST.2018.2817685

[69] Iwens Gervásio Sene Júnior, Thalia S. Santana, Renato F. Bulcão-Neto, and Barry Porter. 2022. The state of the art of macroprogramming in IoT: An update. *Journal of Internet Services and Applications* (2022). https://doi.org/10.5753/jisa.2022.2372

[70] Charbel El Kaed, Imran Khan, Andre Van Den Berg, Hicham Hossayni, and Christophe Saint-Marcel. 2018. SRE: Semantic Rules Engine for the Industrial Internet-Of-Things Gateways. *IEEE Transactions on Industrial Informatics* (2018). https://doi.org/10.1109/TII.2017.2769001

[71] Nikos Katzouris, Alexander Artikis, and Georgios Paliouras. 2017. Parallel Online Learning of Event Definitions. In *27th International Conference on Inductive Logic Programming (Lecture Notes in Computer Science)*. Springer. https://doi.org/10.1007/978-3-319-78090-0_6

[72] Nikos Katzouris, Alexander Artikis, and Georgios Paliouras. 2019. Parallel online event calculus learning for complex event recognition. *Future Generation Computing Systems* (2019). https://doi.org/10.1016/j.future.2018.11.033

[73] Setrag Khoshafian and Carolyn Rostetter. 2015. Digital prescriptive maintenance. *Internet of Things, Process of Everything, BPM Everywhere* (2015).

[74] Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu. 2008. Probabilistic Event Extraction from RFID Data. In *Proceedings of the 24th International Conference on Data Engineering*. IEEE Computer Society. https://doi.org/10.1109/ICDE.2008.4497596

[75] Cornel Klein, Reiner Schmid, Christian Leuxner, Wassiou Sitou, and Bernd Spanfelner. 2008. A Survey of Context Adaptation in Autonomic Computing. In *Fourth International Conference on Autonomic and Autonomous Systems*. https://doi.org/10.1109/ICAS.2008.23

[76] Ibrahim Kök, Feyza Yildirim Okay, Ozgecan Muyanli, and Suat Özdemir. 2023. Explainable Artificial Intelligence (XAI) for Internet of Things: A Survey. *IEEE Internet Things Journal* (2023). https://doi.org/10.1109/JIOT.2023.3287678

[77] Ilya Kolchinsky and Assaf Schuster. 2019. Real-Time Multi-Pattern Detection over Event Streams. In *International Conference on Management of Data*. ACM. https://doi.org/10.1145/3299869.3319869

[78] Robert A. Kowalski and Marek J. Sergot. 1986. A Logic-based Calculus of Events. *New Generation Computing* (1986). https://doi.org/10.1007/BF03037383

[79] Jay Kreps, Neha Narkhede, Jun Rao, et al. 2011. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*.

[80] J. Krishnamurthy and M. Maheswaran. 2016. Chapter 5 - Programming frameworks for Internet of Things. In *Internet of Things*. Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-805395-9.00005-8

[81] Mohammad Amin Kuhail, Shahbano Farooq, Rawad Hammad, and Mohammed Bahja. 2021. Characterizing Visual Programming Approaches for End-User Developers: A Systematic Review. *IEEE Access* (2021). https://doi.org/10.1109/access.2021.3051043

[82] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy, and Siddarth Taneja. 2015. Twitter Heron: Stream Processing at Scale. In *ACM SIGMOD International Conference on Management of Data*. ACM. https://doi.org/10.1145/2723372.2742788

[83] Shashi Shekhar Kumar and Sonali Agarwal. 2024. Rule based complex event processing for IoT applications: Review, classification and challenges. *Expert Systems* (2024). https://doi.org/10.1111/EXSY.13597

[84] Dapeng Lan, Amir Taherkordi, Frank Eliassen, and Geir Horn. 2019. A Survey on Fog Programming: Concepts, State-of-the-Art, and Research Challenges. In *2nd International Workshop on Distributed Fog Services Design*. ACM. https://doi.org/10.1145/3366613.3368120

[85] Borui Li and Wei Dong. 2020. EdgeProg: Edge-centric Programming for IoT Applications. In *IEEE 40th International Conference on Distributed Computing Systems*. IEEE. https://doi.org/10.1109/icdcs47774.2020.00038

[86] Guoli Li and Hans-Arno Jacobsen. 2005. Composite Subscriptions in Content-Based Publish/Subscribe Systems. In *6th International Middleware Conference (Lecture Notes in Computer Science)*. Springer. https://doi.org/10.1007/11587552_13

[87] Jiayue Liang, Fang Liu, Shen Li, and Zhenhua Cai. 2019. A Comparative Research on Open Source Edge Computing Systems. In *5th International Conference on Artificial Intelligence and Security (Lecture Notes in Computer Science)*. Springer. https://doi.org/10.1007/978-3-030-24265-7_14

[88] Marten Lohstroh, Hokeun Kim, John C. Eidson, Chadlia Jerad, Beth Osyk, and Edward A. Lee. 2019. On Enabling Technologies for the Internet of Important Things. *IEEE Access* 7 (2019), 27244–27256. https://doi.org/10.1109/ACCESS.2019.2901509

[89] David Luckham. 2002. *The power of events*. Addison-Wesley Reading.

[90] David Luckham. 2006. What's the Difference Between ESP and CEP. https://complexevents.com/2020/06/15/whats-the-difference-between-esp-and-cep-2/

[91] David Luckham. 2020. Causality in Event Stream Analytics. https://complexevents.com/2020/08/27/causality-in-event-stream-analytics/

[92] David C Luckham. 2011. *Event processing for business: organizing the real-time enterprise*. John Wiley & Sons.

[93] Nadeem Mahmood, Madiha Khurram Pasha, and Khurram Pasha. 2017. Survey of Applications of Complex Event Processing (CEP) in Health Domain. *Sukkur IBA Journal of Computing and Mathematical Sciences* (2017). http://journal.iba-suk.edu.pk:8089/sibajournals/index.php/sjcms/article/view/21

[94] Alessandro Margara, Gianpaolo Cugola, and Giordano Tamburrelli. 2014. Learning from the past: automated rule generation for complex event processing. In *8th ACM International Conference on Distributed Event-Based Systems*. ACM. https://doi.org/10.1145/2611286.2611289

[95] Ruben Mayer, Boris Koldehofe, and Kurt Rothermel. 2015. Predictable Low-Latency Event Detection With Parallel Complex Event Processing. *IEEE Internet Things Journal* (2015). https://doi.org/10.1109/JIOT.2015.2397316

[96] Nasro Min-Allah and Saleh Alrashed. 2020. Smart campus—A sketch. *Sustainable Cities and Society* (2020). https://doi.org/10.1016/j.scs.2020.102231

[97] Adrian Mizzi, Joshua Ellul, and Gordon Pace. 2018. D'Artagnan: An Embedded DSL Framework for Distributed Embedded Systems. In *Real World Domain Specific Languages Workshop*. ACM. https://doi.org/10.1145/3183895.3183899

[98] Raef Mousheimish, Yehia Taher, and Karine Zeitouni. 2017. Automatic Learning of Predictive CEP Rules: Bridging the Gap between Data Mining and Complex Event Processing. In *11th ACM International Conference on Distributed and Event-based Systems*. ACM. https://doi.org/10.1145/3093742.3093917

[99] Dmitry Namiot and Manfred Sneps-Sneppe. 2014. On IoT programming. *International Journal of Open Information Technologies* (2014). http://injoit.org/index.php/j1/article/view/142

[100] Hamid Nasiri, Saeed Nasehi, and Maziar Goudarzi. 2019. Evaluation of distributed stream processing frameworks for IoT applications in

Smart Cities. *Journal of Big Data* (2019). https://doi.org/10.1186/s40537-019-0215-2

[101] Elena Nazzi and Tomas Sokoler. 2011. Walky for embodied microblogging: sharing mundane activities through augmented everyday objects. In *13th Conference on Human-Computer Interaction with Mobile Devices and Services*. ACM. https://doi.org/10.1145/2037373.2037461

[102] Julie L Newcomb, Satish Chandra, Jean-Baptiste Jeannin, Cole Schlesinger, and Manu Sridharan. 2017. IOTA: a calculus for internet of things automation. In *ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. https://doi.org/10.1145/3133850.3133860

[103] Ryan Newton and Matt Welsh. 2004. Region streams: functional macroprogramming for sensor networks. In *1st international workshop on Data management for sensor networks*. ACM Press. https://doi.org/10.1145/1052199.1052213

[104] Anne H. Ngu, Mario Gutierrez, Vangelis Metsis, Surya Nepal, and Quan Z. Sheng. 2017. IoT Middleware: A Survey on Issues and Enabling Technologies. *IEEE Internet of Things Journal* (2017). https://doi.org/10.1109/JIOT.2016.2615180

[105] Xuan Thang Nguyen, Huu Tam Tran, Harun Baraki, and Kurt Geihs. 2015. FRASAD: A framework for model-driven IoT Application Development. In *IEEE 2nd World Forum on Internet of Things*. IEEE. https://doi.org/10.1109/WF-IoT.2015.7389085

[106] Shadi A Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H Campbell. 2017. Samza: stateful scalable stream processing at LinkedIn. *Proceedings of the VLDB Endowment* (2017). https://doi.org/10.14778/3137765.3137770

[107] Joseph Noor, Hsiao-Yun Tseng, Luis Garcia, and Mani B. Srivastava. 2019. DDFlow: visualized declarative programming for heterogeneous IoT networks. In *International Conference on Internet of Things Design and Implementation*. ACM. https://doi.org/10.1145/3302505.3310079

[108] Oracle. 2021. *Oracle CEP*. https://docs.oracle.com/cd/E21764_01/doc.1111/e14476/overview.htm#CEPGS106

[109] Heemin Park and Jae Won Lee. 2012. Task Assignment in Wireless Sensor Networks via Task Decomposition. *Information Technology Control* (2012). https://doi.org/10.5755/j01.itc.41.4.887

[110] Pankesh Patel and Damien Cassou. 2015. Enabling high-level application development for the Internet of Things. *Journal of Systems and Software* (2015). https://doi.org/10.1016/j.jss.2015.01.027

[111] Charith Perera, Chi Harold Liu, Srimal Jayawardena, and Min Chen. 2014. A Survey on Internet of Things From Industrial Market Perspective. *IEEE Access* (2014). https://doi.org/10.1109/ACCESS.2015.2389854

[112] Peter R. Pietzuch, Jonathan Ledlie, Jeffrey Shneidman, Mema Roussopoulos, Matt Welsh, and Margo I. Seltzer. 2006. Network-Aware Operator Placement for Stream-Processing Systems. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006*. IEEE Computer Society, 49. https://doi.org/10.1109/ICDE.2006.105

[113] Alexander Power and Gerald Kotonya. 2020. BoboCEP: Distributed Complex Event Processing for Resilient Fault-Tolerance Support in IoT. In *6th IEEE International Conference on Big Data Computing Service and Applications*. IEEE. https://doi.org/10.1109/BigDataService49289.2020.00024

[114] Jun Qi, Po Yang, Atif Waraich, Zhikun Deng, Youbing Zhao, and Yun Yang. 2018. Examining sensor-based physical activity recognition and monitoring for healthcare using Internet of Things: A systematic review. *Journal of Biomedical Informatics* (2018). https://doi.org/10.1016/j.jbi.2018.09.002

[115] Adrienne Raglin, Anshuman Venkateswaran, and Huan Liu. 2019. Abductive Causal Reasoning for Internet of Things. In *IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation*. https://doi.org/10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00328

[116] Partha Pratim Ray. 2017. A Survey on Visual Programming Languages in Internet of Things. *Sci. Program.* 2017 (2017), 1231430:1–1231430:6. https://doi.org/10.1155/2017/1231430

[117] Chris Rayns, Andy Ritchie, Sriram Balakrishnan, Duncan Clark, Phil Coxhead, Daniel Donnelly, Kallol Ghosh, Daniel Millwood, Nicolas Peulvast, Ian Vanstone, et al. 2011. *Patterns: Integrating WebSphere ILOG JRules with IBM Software*. IBM Redbooks.

[118] Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade, and Siobhán Clarke. 2016. Middleware for Internet of Things: A Survey. *IEEE Internet Things J.* 3, 1 (2016), 70–95. https://doi.org/10.1109/JIOT.2015.2498900

[119] Eduard Gibert Renart, Daniel Balouek-Thomert, and Manish Parashar. 2019. An Edge-Based Framework for Enabling Data-Driven Pipelines for IoT Systems. In *IEEE International Parallel and Distributed Processing Symposium Workshops*. https://doi.org/10.1109/IPDPSW.2019.00146

[120] H. Röger and R. Mayer. 2019. A comprehensive survey on parallelization and elasticity in stream processing. *Comput. Surveys* (2019). https://doi.org/10.1145/3303849

[121] Stuart Russell. 2019. *Human compatible: AI and the problem of control*. Penguin Uk.

[122] R. Sahal, J.G. Breslin, and M.I. Ali. 2020. Big data and stream processing platforms for Industry 4.0 requirements mapping for a predictive maintenance use case. *Journal of Manufacturing Systems* (2020). https://doi.org/10.1016/j.jmsy.2019.11.004

[123] Aymen J Salman, Mohammed Al-Jawad, and Wisam Al Tameemi. 2021. Domain-Specific Languages for IoT: Challenges and Opportunities. (2021). https://doi.org/10.1088/1757-899X/1067/1/012133

[124] Farzad Samie, Lars Bauer, and Jörg Henkel. 2019. From Cloud Down to Things: An Overview of Machine Learning in Internet of Things. *IEEE Internet Things Journal* (2019). https://doi.org/10.1109/JIOT.2019.2893866

[125] Ltd. Samsung Electronics Co. 2021. *Spark CEP*. https://github.com/samsung/spark-cep

[126] Yuya Sasaki. 2022. A Survey on IoT Big Data Analytic Systems: Current and Future. *IEEE Internet Things Journal* (2022). https://doi.org/10.1109/JIOT.2021.3131724

[127] Claudio Savaglio, Giancarlo Fortino, Maria Ganzha, Marcin Paprzycki, Costin Bădică, and Mirjana Ivanović. 2018. *Agent-Based Computing in the Internet of Things: A Survey*. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-66379-1_27

[128] Claudio Savaglio, Maria Ganzha, Marcin Paprzycki, Costin Badica, Mirjana Ivanovic, and Giancarlo Fortino. 2020. Agent-based Internet of Things: State-of-the-art and research challenges. *Future Generation Computing Systems* (2020). https://doi.org/10.1016/J.FUTURE.2019.09.016

[129] Ognjen Scekic, Tommaso Schiavinotto, Svetoslav Videnov, Michael Rovatsos, Hong Linh Truong, Daniele Miorandi, and Schahram Dustdar. 2020. A Programming Model for Hybrid Collaborative Adaptive Systems. *IEEE Transactions on Emerging Topics in Computing* (2020). https://doi.org/10.1109/TETC.2017.2702578

[130] Paul Scerri, David V. Pynadath, and Milind Tambe. 2002. Towards Adjustable Autonomy for the Real World. *Journal of Artificial Intelligence Research* (2002). https://doi.org/10.1613/JAIR.1037

[131] Marc Schaaf. 2018. A Service Based Architecture for Situation-Aware Adaptive Event Stream Processing. In *Tenth International Conference on Advanced Service Computing*.

[132] Marc Schaaf. 2019. Towards a Microservices-based Distribution for Situation-aware Adaptive Event Stream Processing. In *Eleventh International Conference on Advanced Service Computing*.

[133] Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter R. Pietzuch. 2009. Distributed complex event processing with query rewriting. In *3rd ACM International Conference on Distributed Event-Based Systems*. https://doi.org/10.1145/1619258.1619264

[134] Bruno S Sergi, Elena G Popkova, Aleksei V Bogoviz, and Tatiana N Litvinova. 2019. *Understanding Industry 4.0: AI, the Internet of Things, and the Future of Work*. Emerald Group Publishing.

[135] David Servat, Edith Perrier, Jean-Pierre Treuil, and Alexis Drogoul. 1998. *When Agents Emerge from Agents: Introducing Multi-scale Viewpoints in Multi-agent Simulations*. Springer Berlin Heidelberg. https://doi.org/10.1007/10692956_13

[136] Kiran Jot Singh and Divneet Singh Kapoor. 2017. Create Your Own Internet of Things: A survey of IoT platforms. *IEEE Consumer Electronics Magazine* (2017). https://doi.org/10.1109/MCE.2016.2640718

[137] I. Skarbovsky. 2021. *IBM Proactive Technology Online (PROTON)*. https://github.com/ishkin/Proton

[138] Román Sosa, Csaba Kiraly, and Juan D Parra Rodriguez. 2018. Offloading Execution from Edge to Cloud: A Dynamic Node-RED Based Approach. In *IEEE international conference on cloud computing technology and science*. https://doi.org/10.1109/cloudcom2018.2018.00039

[139] Paul Soulier, Depeng Li, and John R. Williams. 2015. A survey of language-based approaches to Cyber-Physical and embedded system development. *Tsinghua Science and Technology* (2015). https://doi.org/10.1109/TST.2015.7085626

[140] Christoph Stoiber and Stefan Schönig. 2021. Event-Driven Business Process Management enhancing IoT - a Systematic Literature Review and Development of Research Agendas. In *Innovation durch Informationssysteme - WI als zukunftsweisende Wissenschaft, 16. Internationale Tagung Wirtschaftsinformatik (WI 2021), March 09-11, 2021, Universität Duisburg-Essen, Germany*. AISeL. https://aisel.aisnet.org/wi2021/EManagementofdigitalprocesses20/Track20/1

[141] Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. 2005. The 8 requirements of real-time stream processing. *ACM SIGMOD Record* (2005). https://doi.org/10.1145/1107499.1107504

[142] Sriskandarajah Suhothayan, Kasun Gajasinghe, Isuru Loku Narangoda, Subash Chaturanga, Srinath Perera, and Vishaka Nanayakkara. 2011. Siddhi: a second look at complex event processing architectures. In *ACM SC Workshop on Gateway Computing Environments*. https://doi.org/10.1145/2110486.2110493

[143] Mohammadmehdi Talebi, Mohsen Sharifi, and Mohammadhesam Kalantari. 2021. ACEP: an adaptive strategy for proactive and elastic processing of complex events. *Journal of Supercomputing* (2021). https://doi.org/10.1007/s11227-020-03454-0

[144] Jose Paolo Talusan, Francis Tiausas, Keiichi Yasumoto, Michael Wilbur, Geoffrey Pettet, Abhishek Dubey, and Shameek Bhattacharjee. 2019. Smart Transportation Delay and Resiliency Testbed Based on Information Flow of Things Middleware. In *IEEE International Conference on Smart Computing*. https://doi.org/10.1109/SMARTCOMP.2019.00022

[145] Giacomo Tanganelli, Carlo Vallati, and Enzo Mingozzi. 2019. Rapid Prototyping of IoT Solutions: A Developer's Perspective. *IEEE Internet Computing* (2019). https://doi.org/10.1109/MIC.2019.2927202

[146] Nicoleta Tantalaki, Stavros Souravlas, and Manos Roumeliotis. 2019. A review on big data real-time stream processing and its scheduling techniques. *International Journal of Parallel, Emergent and Distributed Systems* March (2019). https://doi.org/10.1080/17445760.2019.1585848

[147] The JBoss Drools team. 2021. *Drools Fusion User Guide*. https://docs.jboss.org/drools/release/5.3.0.Final/drools-fusion-docs/html_single/#d0e79

[148] Riddhi Thakkar and Madhuri Bhavsar. 2022. Achieving multilevel elasticity for distributed stream processing systems in the cloud environment: A review and conceptual framework. In *Proceedings of the 2022 Fourteenth International Conference on Contemporary Computing, IC3-2022*. ACM, 81–90. https://doi.org/10.1145/3549206.3549224

[149] Aparna Saisree Thuluva, Darko Anicic, Sebastian Rudolph, and Malintha Adikari. 2020. Semantic Node-RED for rapid development of interoperable industrial IoT applications. *Semantic Web* (2020). https://doi.org/10.3233/sw-200405

[150] Yuan Tian, Eylem Ekici, and Füsun Özgüner. 2005. Energy-constrained task mapping and scheduling in wireless sensor networks. In *IEEE 2nd International Conference on Mobile Adhoc and Sensor Systems*. https://doi.org/10.1109/MAHSS.2005.1542802

[151] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy Ryaboy. 2014. Storm@twitter. In *ACM SIGMOD International Conference on Management of Data (SIGMOD/PODS'14)*. https://doi.org/10.1145/2588555.2595641

[152] Yulia Turchin, Avigdor Gal, and Segev Wasserkrug. 2009. Tuning complex event processing rules using the prediction-correction paradigm. In *Third ACM International Conference on Distributed Event-Based Systems*. https://doi.org/10.1145/1619258.1619272

[153] Itorobong S. Udoh and Gerald Kotonya. 2018. Developing IoT applications: challenges and frameworks. *IET Cyper-Physical Systems: Theory & Applications* (2018). https://doi.org/10.1049/iet-cps.2017.0068

[154] Mirko Viroli, Jacob Beal, Ferruccio Damiani, Giorgio Audrito, Roberto Casadei, and Danilo Pianini. 2019. From distributed coordination to field calculus and aggregate computing. *Journal of Logics and Algebraic Methods of Programming* (2019). https://doi.org/10.1016/j.jlamp.2019.100486

[155] Kaiyu Wan, Danny Hughes, Ka Lok Man, and Tomas Krilavicius. 2010. Composition challenges and approaches for cyber physical systems. In *Proceedings of the 1st IEEE International Conference on Networked Embedded Systems for Enterprise Applications, NESEA 2010, November 25-26, 2010, Suzhou, China*. IEEE Computer Society, 1–7. https://doi.org/10.1109/NESEA.2010.5678065

[156] Jonas Wanner, Christopher Wissuchek, and Christian Janiesch. 2020. Machine Learning and Complex Event Processing. A Review of Real-time Data Analytics for the Industrial Internet of Things. *Enterpise Modelling and Information Systems Architecture International Journal of Concepts Modelling* (2020). https://doi.org/10.18417/EMISA.15.1

[157] Robert Wenger, Xiru Zhu, Jayanth Krishnamurthy, and Muthucumaru Maheswaran. 2016. A Programming Language and System for Heterogeneous Cloud of Things. In *2nd IEEE International Conference on Collaboration and Internet Computing*. https://doi.org/10.1109/CIC.2016.033

[158] Michael Weyrich and Christof Ebert. 2016. Reference Architectures for the Internet of Things. *IEEE Softw.* 33, 1 (2016), 112–116. https://doi.org/10.1109/MS.2016.20

[159] Michael J. Wooldridge. 2009. *An Introduction to MultiAgent Systems, Second Edition*. Wiley.

[160] Eugene Wu, Yanlei Diao, and Shariq Rizvi. 2006. High-performance complex event processing over streams. In *ACM SIGMOD International Conference on Management of Data*. https://doi.org/10.1145/1142473.1142520

[161] Yulei Wu. 2021. Cloud-Edge Orchestration for the Internet of Things: Architecture and AI-Powered Data Processing. *IEEE Internet Things J.* 8, 16 (2021), 12792–12805. https://doi.org/10.1109/JIOT.2020.3014845

[162] Khensani Xivuri and Hossana Twinomurinzi. 2021. A Systematic Review of Fairness in Artificial Intelligence Algorithms. In *20th IFIP WG 6.11 Conference on e-Business, e-Services and e-Society (LNCS)*. Springer. https://doi.org/10.1007/978-3-030-85447-8_24

[163] Li Da Xu, Wu He, and Shancang Li. 2014. Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics* (2014). https://doi.org/10.1109/TII.2014.2300753

[164] Keiichi Yasumoto, Hirozumi Yamaguchi, and Hiroshi Shigeno. 2016. Survey of Real-time Processing Technologies of IoT Data Streams.

Journal of Information Processing (2016), 195–202. https://doi.org/10.2197/ipsjjip.24.195

[165] Rachid Zagrouba and Amine Kardi. 2021. Comparative Study of Energy Efficient Routing Techniques in Wireless Sensor Networks. *Information* (2021). https://doi.org/10.3390/info12010042

[166] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: cluster computing with working sets. (2010).

[167] Franco Zambonelli and H. Van Dyke Parunak. 2003. Signs of a Revolution in Computer Science and Software Engineering. In *Engineering Societies in the Agents World III*. Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/3-540-39173-8_2

[168] Franco Zambonelli, Mirko Viroli, Giancarlo Fortino, and Barbara Re. 2019. Towards Adaptive Flow Programming for the IoT: The Fluidware Approach. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 549–554. https://doi.org/10.1109/PERCOMW.2019.8730736

[169] Steffen Zeuch, Ankit Chaudhary, Bonaventura Del Monte, Haralampos Gavriilidis, Dimitrios Giouroukis, Philipp M. Grulich, Sebastian Breß, Jonas Traub, and Volker Markl. 2020. The NebulaStream Platform for Data and Application Management in the Internet of Things. In *10th Conference on Innovative Data Systems Research*. www.cidrdb.org. http://cidrdb.org/cidr2020/papers/p7-zeuch-cidr20.pdf

[170] Steffen Zeuch, Eleni Tzirita Zacharatou, Shuhao Zhang, Xenofon Chatziliadis, Ankit Chaudhary, Bonaventura Del Monte, Dimitrios Giouroukis, Philipp M. Grulich, Ariane Ziehn, and Volker Markl. 2020. NebulaStream: Complex Analytics Beyond the Cloud. *Open Journal of Internet Things* (2020). https://www.ronpub.com/ojiot/OJIOT_2020v6i1n07_Zeuch.html

[171] Jing Zhang and Dacheng Tao. 2021. Empowering Things With Intelligence: A Survey of the Progress, Challenges, and Opportunities in Artificial Intelligence of Things. *IEEE Internet of Things Journal* (2021). https://doi.org/10.1109/JIOT.2020.3039359

[172] Qunzhi Zhou, Yogesh Simmhan, and Viktor K. Prasanna. 2017. Knowledge-infused and consistent Complex Event Processing over real-time and persistent streams. *Future Generation Computing Systems* (2017). https://doi.org/10.1016/j.future.2016.10.030

[173] Ariane Ziehn. 2020. Complex Event Processing for the Internet of Things. In *VLDB PhD Workshop co-located with the 46th International Conference on Very Large Databases (CEUR Workshop Proceedings)*.