Programming Approaches for Large-Scale IoT System Development: State of the Art

Chapter 2
Programming approaches for large-scale IoT systems development: state of the art

Roberto Casadei[1], Fabrizio Fornari[2], Stefano Mariani[3], and Claudio Savaglio[4]

Abstract Software engineers 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 information to decide how to achieve the system goals (*processing* facet), and how to enact such decisions to affect the IoT environment (*actuation* facet). For each of these, one can either develop ad-hoc solutions by relying on mainstream programming languages, or exploit existing IoT-specific software libraries, frameworks, and platforms. In this chapter, we survey the state of the art of "IoT programming", clarifying which programming paradigms and platforms are most commonly adopted, with the goal of uncovering which research areas are mostly active in IoT programming.

2.1 Introduction

Various methods exist for programming IoT systems, often involving ad-hoc combinations of software libraries, frameworks, and platforms. This can lead to confusion about the best approaches and how to integrate them effectively [98, 24].

The intrinsic complexity of IoT programming at scale is due to the heterogeneity of devices, their distribution, and the local-to-global connection [65, 22]. To address this complexity, IoT programming research focuses on three fundamental facets: how to manage the perception of properties of interest from the IoT environment through sensors (*sensing* facet), how to process this information for decision-making to achieve system goals (*processing* facet), and finally how to enact these decisions

University of Bologna, Cesena, Italy
e-mail: roby.casadei@unibo.it
· University of Camerino, Camerino, Italy
e-mail: fabrizio.fornari@unicam.it
· University of Modena and Reggio Emilia, Reggio Emilia, Italy
e-mail: stefano.mariani@unimore.it · Università della Calabria, Rende, Italy
e-mail: csavaglio@dimes.unical.it

by affecting the environment through actuators (*actuation* facet). Accordingly, IoT programming research spans the entire software stack: languages and libraries for programming devices, platforms supporting data streaming and event processing, frameworks for distributed computing and service deployment.

Several surveys have identified the most active research trends, categorizing the driving application domains, and emphasizing open issues and opportunities for further research [68, 74, 89, 86, 80]. However, these surveys tend to focus on a particular approach (e.g., stream processing but not macro-programming), infrastructural layer (e.g., the cloud but not the Edge, platforms but not languages), and/or feature set (e.g., opportunistic deployment but not adaptiveness), failing to clarify how different approaches fit into the overall picture. Therefore, an integrative analysis of the IoT programming research landscape is needed.

For these reasons, we provide a meta-review (i.e. first targets of our review are surveys themselves) of the literature about "IoT programming". Our goal is to address the following questions: (i) what are the research areas contributing the most to IoT programming? (ii) how do they cover the sensing, processing, and actuation facets? (iii) what are the challenges yet to be dealt with?

2.2 Conceptual framework and review method

Every IoT system has three distinct but closely related facets of computation:

- The *sensing facet* is concerned with how to *perceive* the "state of the world" according to the application goals. This facet includes gathering measured data from sensor devices, preprocessing such data to derive more complex situations [76], and possibly even putting such situations into causal relations [77].
- The *processing facet* is concerned with elaborating available data to *decide* which actions to carry out to achieve the system goals. This facet includes simple reactive rule engines producing commands depending on the current situation [47], more complex forms of event processing, agent-oriented computing to have goal-oriented behaviours [85], and so on.
- The *actuation facet* is concerned with how to *affect* the state of the world according to the application goals. This facet includes low-level programming of individual devices with mainstream languages, collective engineering of the global system behavior, and many other different approaches [39].

We place the surveyed literature along these facets to clarify which are most considered and which require further development.

To conduct our survey, we began by performing keyword-based searches on Scopus and DBLP, focusing on papers that were already surveys themselves. We performed several searches including "IoT/CPS" and "programming", such as "IoT programming languages survey", "CPS programming approaches review", and similar combinations. The reason for this methodology is that the topic of IoT programming is so broad and has so many different nuances that it would be impossible to

rigorously scrutinize all the available literature. By focusing on existing surveys, we can realistically analyse them in detail and later perform snowballing (i.e. searching for therein referenced papers) 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 same paper (e.g. conference and journal version)
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.

We then grouped the collected papers into categories to snowball and get a more focused overview of the research landscape.

Stream processing (SP) [12] A computing paradigm that focusses how to gather, process, and analyse continuous flows of data called *streams* within a small time period from the time of receiving. SP gained popularity within the IoT landscape as often times IoT deployments need first and foremost to tackle the challenge of collecting and processing timely 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, 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) [58] Builds on the abstraction of low level event to create 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 enough. Whenever there is the need for complex analytics and insights, and to some extent of closing the feedback loop by controlling the monitored system back, CEP has an edge over other approaches thanks to its native capability of detecting complex patterns over incoming events data and of triggering business processes reactively. For this reason 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, but spanning from 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), to emerging programming paradigms that target a *multitude* of devices at once (macro) [41], by developing a global, high level program that is then "translated" into lower level, device-bound (traditional) programs [39]. A description of each approach is given in the corresponding Section 2.5. 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. On the results retrieved, the same filtering steps already mentioned have been applied. The papers survived are actually the one cited throughout this whole paper, and summarised by Tables 2.1–2.5.

It is worth emphasising a few aspects about 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 2.3, 2.4, and 2.5 for a discussion of each area). On the on hand, this means that each research area brings effective solutions to specific issues of programming large-scale, autonomous IoT systems, on the other hand, that these solutions are only partially solving the problem, hence integrating them is likely the key for better IoT programming.

The surveys and papers collected have been roughly divided in three categories – Tables 2.1, 2.3, and 2.5, one per each macro-area –, depending on their scope and intended goal: a first category (GEN) is focussed on 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, parallelization) of the languages and platforms, and compares them based on the degree of support provided.

2.3 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. 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. The 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 [60].

Amongst the characteristics that a SP platform should present [92], 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 their processing needs across multiple processors and machines. In the most advanced systems, this 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 [21]. 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.

2.3.1 Related Surveys

Many surveys about SP or ESP can be found, witnessing the huge interest in the topic, as summarised in Table 2.1.

2.3.2 Main characteristics

Here, we derive from the surveyed literature the main characteristics of SP platforms (Table 2.2), and compare well-known SP platforms based on the degree of support they provide to those characteristics.

Table 2.1: Main SP and ESP surveys, most recent first.

Ref.	Brief description	Year	Category
[96]	Focus on techniques for Cloud elasticity (mostly QoS-aware resource usage) at operator placement/execution level	2021	DOM
[36]	Historical perspective, emphasising evolution of research landscape, and comparing different SP platforms generations	2020	GEN
[82]	Systematic analysis of strengths and weaknesses of open source SP techs for predictive maintenance in Industry 4.0	2020	DOM
[45]	Comparative study of selected distributed SP & analytics platforms	2019	GEN
[66]	Evaluation of selected open source distributed SP platforms deployed at the "data layer" of Smart City applications	2019	DOM
[94]	Review of scheduling techniques for distributed SP	2019	SPEC
[81]	Focus on techniques for parallelisation and elasticity of SP	2019	SPEC
[32]	Focus on SP mechanisms leveraging resource elasticity in the Cloud	2018	SPEC
[29]	Overview of ESP architectures, use cases, and open research issues	2018	GEN
[103]	Specifically concerned with real-time SP in IoT deployments, hence surveys related techniques, challenges, and technologies	2016	SPEC

- *Programming Model*. It defines if programmers use *declarative* queries, *imperative* programming, or *compositional* programming to specify the operations or execution patterns of a 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 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 a SP system is deployed: e.g., a cluster, the Cloud, Fog, Edge, etc.
- *Parallelization*. Parallel SP reduces queuing latency and increases throughput, as it allows to process multiple streams simultaneously instead of sequentially. With *task* parallelization, multiple operations (i.e., tasks) can run in parallel on the same stream. With *data* parallelization, multiple instances of a 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 organized 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).
- *Data Flow Abstraction*. Which is the abstraction used to handle flow of data between stream operators. Most SP platforms adopt a *Directed Acyclic Graph* (DAG) of operators to structure the processing application in topologies.

[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 2.2: Summary of the SP main characteristics.

	Flink [19]	**Heron** [54]	**Kafka Streams** [51]	**Samza** [70]	**Spark Streaming** [105]	**Storm** [97]
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
Parallelization	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

- *Fault Tolerance*. The ability of a 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 actual language stack supported. The trend is to integrate with mainstream, general purpose languages, rather than to provide custom domain-specific languages.

In Table 2.2 the design choices of each specific SP platform (columns) to implement the corresponding characteristic (rows) is briefly described by keywords.

2.3.3 Summary

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.

2.4 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 [58]. Specifically, they target the definition and detection of high-level situations of interest, or *composite events*, starting from streams of *primitive events* [28]. Event Stream Processing (ESP), as an evolution of SP, and CEP actually indicate different ways to manage events [59]: 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 [61, 27, 49]. CEP foremost goal is to enable abstraction of low level events into a new high level event based on temporal, spatial, or causal patterns. The patterns to be detected are generally specified by domain experts, and the new events generated from patter-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 [56], SASE [102, 40], PEEX [48], RACED [25], and TESLA/T-Rex [26].

2.4.1 Related Surveys

As for SP, most of the collected surveys can be divided in three categories, as depicted in Table 2.3: 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).

Table 2.3: Main CEP surveys, most recent first.

Ref.	Brief description	Year	Category
[37]	Focus on CER (subtask of CEP), especially regarding scale-out mechanisms for query execution	2020	SPEC
[107]	Focus on out-Cloud deployment, in particular adaptations needed for CEP platforms to meet the Fog layer	2020	SPEC
[106]	Overview on the state of 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
[63]	Specifically concerned with 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

2.4.2 Main characteristics

Table 2.4 reports a comparison similar to the one reported in Table 2.2, but in the context of CEP—please notice that columns and rows are inverted w.r.t. Table 2.2 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.
- *Prioritization*. 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 time 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 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 so as to preserve/restore/build causality.
- *Deployment*. How and where CEP solutions are deployed: Centralized, Distributed, Clustered, Cloud, Edge, etc.

As pattern detection and matching is a cornerstone of CEP, it is necessary to expand on it. What enables system designers to define pattern matching is the EQL, a high level programming language for querying events [34, 37]. 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 near real-time applications.
- *Production rules* are 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 [50], that 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 define the pattern.

Table 2.4: Summary of the main characteristics of the surveyed CEP systems (rows and columns are exchanged w.r.t. Table 2.2 to improve layout). NFA = Nondeterministic Finite Automata, BRMS = Business rules management system.

	Filtering	Prioritisation	Patterns	Exceptions	Process triggering	Event hierarchies	Deployment
SASE [102]	Event Algebra		Active Instance Stack (AIS) for events sequence construction on NFA			Primitive types to build Complex types	Centralized
Esper [42]	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 [93]	Siddhi Streaming SQL	Via rabbitmq's AMQP message priority	Pattern and sequence queries via state machines				Centralised as library + microservice on Docker Kubernetes
Cayuga [30]	Automata-based	Priority queues	Sequences of correlated events + "safety conditions" between consecutive events + patterns composition			Via query chains	Centralized
CEP [5]	DOLCE language		Causal and temporal patterns using pre-defined rules based on thresholds + dynamic rules				Distributed on embedded devices
Oracle CEP [72]	Oracle CQL		`CQL MATCH_RECOGNIZE` condition		POJOs triggered by events	Event Processing Networks to create hierarchy of processing agents and events	Clustered
OpenCEP [43]	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 [88]	Automata-based		SQL-like language + filter, union, iteration, time, ...)	Dedicated language operator			Clustered
TIBCO Business Events [39]	Production Rules		Sequencing, duplicate detection, and others		Event-driven process orchestration		Distributed via Docker or Kubernetes + TIBCO Cloud + on premise
EdgeCEP [23]	Specification language for relational expressions		By stating an explicit event key and contents of interest				Edge
BoboCEP [75]	Automata-based		Based on NFA				Distributed on Edge
Decision CEP Engine [44]	Data stream query language						Clustered
StreamInsight [7]	Declarative LINQ						Centralized
Drools Fusion [95]	Production Rules	Rules can have priorities	Improved Rete algorithm + temporal operators		BRMS to trigger business process execution		Centralized
ILOG JRules [79]	Production Rules				BRMS to trigger and support business process execution		Centralized
Spark CEP [84]	Data stream query language						Distributed
Proton [90]	Data stream query language		Filtering + join for multiple events + sequencing + aggregation functions + track values in time			Event Processing Networks to create event hierarchies	Centralized + distributed via STORM
EasyFlinkCEP [38]	Extended regular expressions		EasyFlinkCEP Operator + FlinkCEP's language to define CEP queries + time windows				Clustered

2.4.3 Summary

CEP somewhat encompasses all the three facets of computation, although rather weakly along the actuation one. 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 aggregation of sensory data streams.

2.5 Micro and macro-programming

In the IoT, a common goal is to program the *global* or *collective* behaviour of the entire system or groups of devices. This often entails coordination and information exchange among the devices, so that the overall action is coherent and informed (i.e., based not only on purely local information). In the literature, it is possible to distinguish between two classes of approaches [20]: informally, *micro* approaches that "take the perspective" of individual devices and address the behaviour of specific roles within the system, and *macro* approaches that somewhat abstract multiple devices into a "single conceptual machine" that is the actual target of programming (*macro-programming* [20]). To the best of our knowledge, term macro-programming has started spreading at the beginning of 2000s in the context of research on programming of wireless sensor networks [67]. Micro-programming includes traditional approaches, e.g., based on mainstream programming languages, where the programmer specifies a script of instructions to control passive elements or defines the control loop of each individual active device. Macro-programming, instead, includes approaches that, e.g., abstract a whole network of devices as a database (cf. TinyDB [62]) or as a *spatial computing* system [15]; a notable macro-programming approach is given by the so-called *aggregate computing* paradigm [99].

2.5.1 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. [52] and in a specific section of the survey by Dias et al. [31]—see Table 2.5. 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

Table 2.5: Main MMP surveys, most recent first.

Ref.	Brief description	Year	Category
[20]	Survey about macro-programming approaches, whatever the application domain	2023	SPEC
[31]	Broad view on the development of IoT systems from the standpoint of software engineering practice	2022	GEN
[46]	Macro-programming, but specifically for the IoT domain	2021	DOM
[53]	Visual programming approaches not only in IoT, but education and robotics, too	2021	SPEC
[13]	IoT development platforms, with emphasis on communication and security	2021	GEN
[35]	Review of IoT platforms from an architectural perspective only	2021	SPEC
[83]	DSLs for used in IoT, focus on evaluation methods	2021	SPEC
[57]	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
[78]	High-level study on 13 visual IoT programming languages	2017	GEN
[52]	Focus on distributed programming languages and tools for embedded-systems within the IoT ecosystem	2016	SPEC
[73]	Comprehensive features' analysis and classification (node-centric, database, macro, model-driven) of IoT programming approaches	2015	GEN
[91]	Low-level languages (both mainstreams and dialects) for CPS and embedded systems	2015	GEN
[100]	Overview of conventional composition mechanisms for CPSs (monolithic, object-oriented, modular, component-based, service-oriented	2010	SPEC

much more broad overview over the whole software engineering practice of developing IoT systems. In particular, the overview about development tools provides many useful 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 to distributed/decentralised programming). However, such surveys do not help researchers and practitioners in understanding which one of the many different programming tools are most suited for different "tasks" in IoT systems development.

A more similar work to ours, if restricted to the micro vs. macro categorisation, is by Patel et al. [73], 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 [20, 46]. They both do an excellent job in sorting out the different existing approaches into a taxonomy.

2.5.2 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 Table 2.6. In line with previous sections, this overview is needed to analyse 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 [69], COMPOSE [33], 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 [71], PyoT [11], D'Artagnan [64], SmartSociety platform [87], and EdgeProg [55], aim at enabling 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 [33] 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., [99]), i.e., dynamic groups of devices sharing a goal; *collective interfaces* (e.g., [71]), namely mechanisms to address or refer to a dynamic number of recipients or senders; and *collective tasks* (e.g., [87]), that 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 [69], allow defining 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 [14]. COMPOSE is quite peculiar as it allows expressing the behaviour of IoT devices by means of 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 [39] 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-tiers 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-tiers architecture. D'Artagnan and Aggregate Computing, instead, promote a different

approach: they enable to program 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 a 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 [104]. Macro-programming approaches typically build on reactive, rather than fully proactive, execution models [20, 46], the latter being covered mostly in agent-oriented paradigms [18].

Table 2.6: Summary of the MMP main characteristics.

	Scope	**Main Abstraction**	**Programming Model**	**Architectural/ Deployment View**	**Execution Model**
FRASAD [69]	Individual	Reaction Rules, Messages	Visual block language	Local WSN	Reactive
SMART-BLOCK [14]	Individual	Reaction Rules	Visual block language	Cloud-based	Reactive
COMPOSE [33]	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 [101]	Individual	Nodes, Activities	Imperative, object-oriented DSL	Edge-to-Cloud	Reactive
PyoT [11]	Collective	Resource (REST)	Imperative, object-oriented, macroprogramming DSL	Master-worker	Reactive
DDFlow [71]	Collective	Data Flow	Visual block language, macroprogramming	Master-worker	Reactive
D'Artagnan [64]	Collective	Data Stream, Data Flow	Functional, stream-based, macroprogramming DSL	Logical network any deployment	Reactive
SmartSociety [87]	Collective	Collective task	Object-oriented DSL	Cloud-based	Customizable
EdgeProg [55]	Collective	Reaction rules, Virtual sensors	Rule-based DSL	Edge server + IoT nodes	Reactive
Aggregate Computing [99]	Collective	Computational Field	Functional declarative macroprogramming	Logical network any deployment	Self-organizing

2.5.3 Summary

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

2.6 Conclusions and Future Work

From the survey just presented, a few considerations can be made to give a bird-eye view of the whole research landscape about IoT programming, 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 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) that are anyway general-purpose and often times also closely resembles a mainstream language. This is to say that the "programming experience" may be quite different depending on the kind of IoT systems one wants to develop.

An huge impact on this programming experience is also given by 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 out, but integration of the different approaches in a single software project nevertheless suffers, as developers need to master different paradigms.

However, such an abundance of approaches also has upsides: since each programming approach is conceived with a specific set of functional and non-functional desiderata, each approach ends up ends up being complementary to others.

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.

References

1. Node-Red. `https://nodered.org/`. [Online; accessed 19-January-2022].
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. The design of the borealis stream processing engine. In *Second Biennial Conference on Innovative Data Systems Research, CIDR 2005, Asilomar, CA, USA, January 4-7, 2005, Online Proceedings*, pages 277–289. www.cidrdb.org, 2005.
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. Aurora: A data stream management system. In Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, page 666. ACM, 2003.
4. Asaf Adi and Opher Etzion. Amit - the situation manager. *VLDB J.*, 13(2):177–203, 2004.
5. Adnan Akbar, François Carrez, Klaus Moessner, Juan Sancho, and Juan Rico. Context-aware stream processing for distributed iot applications. In *2nd IEEE World Forum on Internet of Things, WF-IoT 2015, Milan, Italy, December 14-16, 2015*, pages 663–668. IEEE Computer Society, 2015.
6. Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras. Probabilistic complex event recognition: A survey. *ACM Comput. Surv.*, 50(5):71:1–71:31, 2017.
7. Mohamed Ali, Badrish Chandramouli, Jonathan Goldstein, and Roman Schindlauer. The extensibility framework in microsoft streaminsight. In *2011 IEEE 27th International Conference on Data Engineering*, pages 1242–1253. IEEE, 2011.
8. Chinnapong Angsuchotmetee and Richard Chbeir. A survey on complex event definition languages in multimedia sensor networks. In Richard Chbeir, Rajeev Agrawal, and Ismaïl Biskri, editors, *Proceedings of the 8th International Conference on Management of Digital EcoSystems, MEDES 2016, Biarritz, France, November 1-4, 2016*, pages 99–108. ACM, 2016.
9. Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Rajeev Motwani, Itaru Nishizawa, Utkarsh Srivastava, Dilys Thomas, Rohit Varma, and Jennifer Widom. STREAM: the stanford stream data manager. *IEEE Data Eng. Bull.*, 26(1):19–26, 2003.
10. Alexander Artikis, Anastasios Skarlatidis, François Portet, and Georgios Paliouras. Logic-based event recognition. *Knowl. Eng. Rev.*, 27(4):469–506, 2012.
11. Andrea Azzara, Daniele Alessandrelli, Stefano Bocchino, Matteo Petracca, and Paolo Pagano. Pyot, a macroprogramming framework for the internet of things. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems, SIES 2014, Pisa, Italy, June 18-20, 2014*, pages 96–103. IEEE, 2014.
12. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis, editors, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 1–16. ACM, 2002.
13. Leonardo Babun, Kyle Denney, Z. Berkay Celik, Patrick D. McDaniel, and A. Selcuk Uluagac. A survey on iot platforms: Communication, security, and privacy perspectives. 192:108040.
14. Nayeon Bak, Byeong-Mo Chang, and Kwanghoon Choi. Smart block: A visual block language and its programming environment for iot. *Journal of Computer Languages*, 60:100999, 2020.
15. Jacob Beal, Stefan Dulman, Kyle Usbeck, Mirko Viroli, and Nikolaus Correll. Organizing the aggregate: Languages for spatial computing. *CoRR*, abs/1202.5509, 2012.
16. Jacob Beal, Danilo Pianini, and Mirko Viroli. Aggregate programming for the internet of things. *Computer*, 48(9):22–30, 2015.
17. Bruce Belson, Jason Holdsworth, Wei Xiang, and Bronson Philippa. A survey of asynchronous programming using coroutines in the internet of things and embedded systems. *ACM Trans. Embed. Comput. Syst.*, 18(3):21:1–21:21, 2019.

18. Olivier Boissier, Rafael H Bordini, Jomi Hubner, and Alessandro Ricci. *Multi-agent oriented programming: programming multi-agent systems using JaCaMo*. Mit Press, 2020.
19. Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.
20. Roberto Casadei. Macroprogramming: Concepts, state of the art, and opportunities of macroscopic behaviour modelling. *ACM Comput. Surv.*, 55(13s), jul 2023.
21. Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Frederick Reiss, and Mehul A. Shah. Telegraphcq: Continuous dataflow processing. In Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, page 668. ACM, 2003.
22. Moumena A. Chaqfeh and Nader Mohamed. Challenges in middleware solutions for the internet of things. In *2012 International Conference on Collaboration Technologies and Systems (CTS)*, pages 21–26, 2012.
23. Sunyanan Choochotkaew, Hirozumi Yamaguchi, Teruo Higashino, Megumi Shibuya, and Teruyuki Hasegawa. 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*, pages 121–129. IEEE, 2017.
24. Fulvio Corno, Luigi De Russis, and Juan Pablo Sáenz. On the challenges novice programmers experience in developing iot systems: A survey. *J. Syst. Softw.*, 157, 2019.
25. Gianpaolo Cugola and Alessandro Margara. RACED: an adaptive middleware for complex event detection. In Paul Grace and Frank Eliassen, editors, *Proceedings of the 8th Workshop on Adaptive and Reflective Middleware, ARM 2009, held at the ACM/IFIP/USENIX International Middleware Conference, December 1, 2009, Urbana Champaign, IL, USA*, page 5. ACM, 2009.
26. Gianpaolo Cugola and Alessandro Margara. Complex event processing with T-REX. *J. Syst. Softw.*, 85(8):1709–1728, 2012.
27. Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012.
28. Gianpaolo Cugola and Alessandro Margara. *The Complex Event Processing Paradigm*, pages 113–133. Springer International Publishing, Cham, 2015.
29. Miyuru Dayarathna and Srinath Perera. Recent Advancements in Event Processing. *ACM Comput. Surv.*, 51(2), feb 2018.
30. Alan J. Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, and Walker M. White. Cayuga: A general purpose event monitoring system. In *Third Biennial Conference on Innovative Data Systems Research, CIDR 2007, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*, pages 412–422. www.cidrdb.org, 2007.
31. João Pedro Dias, André Restivo, and Hugo Sereno Ferreira. Designing and constructing internet-of-things systems: An overview of the ecosystem. 19:100529.
32. Marcos Dias de Assunção, Alexandre da Silva Veith, and Rajkumar Buyya. Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. *Journal of Network and Computer Applications*, 103(July 2017):1–17, 2018.
33. Charalampos Doukas and Fabio Antonelli. Compose: Building smart & context-aware mobile applications utilizing iot technologies. In *Global Information Infrastructure Symposium-GIIS 2013*, pages 1–6. IEEE, 2013.
34. Michael Eckert, François Bry, Simon Brodt, Olga Poppe, and Steffen Hausmann. A cep babelfish: Languages for complex event processing and querying surveyed. In *Reasoning in Event-Based Distributed Systems*, pages 47–70. Springer, 2011.
35. Giancarlo Fortino, Antonio Guerrieri, Claudio Savaglio, and Giandomenico Spezzano. A review of internet of things platforms through the iot-a reference architecture. In David Camacho, Domenico Rosaci, Giuseppe M. L. Sarné, and Mario Versaci, editors, *Intelligent Distributed Computing XIV, 14th International Symposium on Intelligent Distributed*

Computing, IDC 2021, Virtual Event, 16-18 September 2021, volume 1026 of *Studies in Computational Intelligence*, pages 25–34. Springer.
36. Marios Fragkoulis, Paris Carbone, Vasiliki Kalavri, and Asterios Katsifodimos. A survey on the evolution of stream processing systems. *CoRR*, abs/2008.00842, 2020.
37. Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. Complex event recognition in the big data era: a survey. *VLDB J.*, 29(1):313–352, 2020.
38. Nikos Giatrakos, Eleni Kougioumtzi, Antonios Kontaxakis, Antonios Deligiannakis, and Yannis Kotidis. Easyflinkcep: Big event data analytics for everyone. In Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 3029–3033. ACM, 2021.
39. TIBCO BusinessEventsUser's Guide. Tibco® businessevents. *Software Release*, 1, 2005.
40. Daniel Gyllstrom, Eugene Wu, Hee-Jin Chae, Yanlei Diao, Patrick Stahlberg, and Gordon Anderson. SASE: complex event processing over streams (demo). In *Third Biennial Conference on Innovative Data Systems Research, CIDR 2007, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*, pages 407–411. www.cidrdb.org, 2007.
41. Jane Hillston, Jeremy Pitt, Martin Wirsing, and Franco Zambonelli. Collective adaptive systems: Qualitative and quantitative modelling and analysis (dagstuhl seminar 14512). *Dagstuhl Reports*, 4(12):68–113, 2014.
42. EsperTech Inc. ESPER, 2021.
43. Red Hat Inc. Open CEP, 2021.
44. STRATIO BIG DATA Inc. Decision CEP Engine, 2021.
45. Haruna Isah, Tariq Abughofa, Sazia Mahfuz, Dharmitha Ajerla, Farhana H. Zulkernine, and Shahzad Khan. A survey of distributed data stream processing frameworks. *IEEE Access*, 7:154300–154316, 2019.
46. Iwens Gervásio Sene Júnior, Thalia S. Santana, Renato F. Bulcão-Neto, and Barry Porter. The state of the art of macroprogramming in iot: An update. *J. Internet Serv. Appl.*, 13(1):54–65, 2022.
47. Charbel El Kaed, Imran Khan, Andre Van Den Berg, Hicham Hossayni, and Christophe Saint-Marcel. Sre: Semantic rules engine for the industrial internet-of-things gateways. *IEEE Transactions on Industrial Informatics*, 14(2):715–724, 2018.
48. Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu. Probabilistic event extraction from RFID data. In Gustavo Alonso, José A. Blakeley, and Arbee L. P. Chen, editors, *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, Mexico*, pages 1480–1482. IEEE Computer Society, 2008.
49. Ilya Kolchinsky and Assaf Schuster. Real-time multi-pattern detection over event streams. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 589–606. ACM, 2019.
50. Robert A. Kowalski and Marek J. Sergot. A logic-based calculus of events. *New Gener. Comput.*, 4(1):67–95, 1986.
51. Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, volume 11, pages 1–7, 2011.
52. J. Krishnamurthy and M. Maheswaran. Chapter 5 - programming frameworks for internet of things. In Rajkumar Buyya and Amir Vahid Dastjerdi, editors, *Internet of Things*, pages 79–102. Morgan Kaufmann, 2016.
53. Mohammad Amin Kuhail, Shahbano Farooq, Rawad Hammad, and Mohammed Bahja. Characterizing visual programming approaches for end-user developers: A systematic review. 9:14181–14202.
54. Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M Patel, Karthik Ramasamy, and Siddarth Taneja. Twitter heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 239–250, 2015.

55. Borui Li and Wei Dong. Edgeprog: Edge-centric programming for iot applications. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 212–222. IEEE, 2020.
56. Guoli Li and Hans-Arno Jacobsen. Composite subscriptions in content-based publish/-subscribe systems. In Gustavo Alonso, editor, *Middleware 2005, ACM/IFIP/USENIX, 6th International Middleware Conference, Grenoble, France, November 28 - December 2, 2005, Proceedings*, volume 3790 of *Lecture Notes in Computer Science*, pages 249–269. Springer, 2005.
57. Marten Lohstroh, Hokeun Kim, John C. Eidson, Chadlia Jerad, Beth Osyk, and Edward A. Lee. On enabling technologies for the internet of important things. *IEEE Access*, 7:27244–27256, 2019.
58. David Luckham. *The power of events*, volume 204. Addison-Wesley Reading, 2002.
59. David Luckham. What's the difference between esp and cep. *https://complexevents.com/2020/06/15/whats-the-difference-between-esp-and-cep-2/*, 2006.
60. David Luckham. Causality in event stream analytics. *https://complexevents.com/2020/08/27/causality-in-event-stream-analytics/*, 2020.
61. David C Luckham. *Event processing for business: organizing the real-time enterprise*. John Wiley & Sons, 2011.
62. Samuel Madden, Robert Szewczyk, Michael J. Franklin, and David E. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. In *4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002), 20-21 June 2002, Callicoon, NY, USA*, pages 49–58. IEEE Computer Society, 2002.
63. Nadeem Mahmood, Madiha Khurram Pasha, and Khurram Pasha. Survey of applications of complex event processing (cep) in health domain. *Sukkur IBA Journal of Computing and Mathematical Sciences*, 1(2):88–94, 2017.
64. Adrian Mizzi, Joshua Ellul, and Gordon Pace. D'Artagnan: An embedded DSL framework for distributed embedded systems. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*, pages 1–9, 2018.
65. Dmitry Namiot and Manfred Sneps-Sneppe. On iot programming. *International Journal of Open Information Technologies*, 2(10):25–28, 2014.
66. Hamid Nasiri, Saeed Nasehi, and Maziar Goudarzi. Evaluation of distributed stream processing frameworks for iot applications in smart cities. *J. Big Data*, 6:52, 2019.
67. Ryan Newton and Matt Welsh. Region streams: functional macroprogramming for sensor networks. In Alexandros Labrinidis and Samuel Madden, editors, *Proceedings of the 1st Workshop on Data Management for Sensor Networks, in conjunction with VLDB, DMSN 2004, Toronto, Canada, August 30, 2004*, volume 72 of *ACM International Conference Proceeding Series*, pages 78–87. ACM, 2004.
68. Anne H. Ngu, Mario Gutierrez, Vangelis Metsis, Surya Nepal, and Quan Z. Sheng. IoT Middleware: A Survey on Issues and Enabling Technologies. *IEEE Internet of Things Journal*, 4(1):1–20, 2017.
69. Xuan Thang Nguyen, Huu Tam Tran, Harun Baraki, and Kurt Geihs. Frasad: A framework for model-driven iot application development. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 387–392. IEEE, 2015.
70. Shadi A Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H Campbell. Samza: stateful scalable stream processing at linkedin. *Proceedings of the VLDB Endowment*, 10(12):1634–1645, 2017.
71. Joseph Noor, Hsiao-Yun Tseng, Luis Garcia, and Mani B. Srivastava. Ddflow: visualized declarative programming for heterogeneous iot networks. In Olaf Landsiedel and Klara Nahrstedt, editors, *Proceedings of the International Conference on Internet of Things Design and Implementation, IoTDI 2019, Montreal, QC, Canada, April 15-18, 2019*, pages 172–177. ACM, 2019.
72. Oracle. Oracle CEP, 2021.
73. Pankesh Patel and Damien Cassou. Enabling high-level application development for the internet of things. *Journal of Systems and Software*, 103:62–84, 2015.

74. Charith Perera, Chi Harold Liu, Srimal Jayawardena, and Min Chen. A survey on internet of things from industrial market perspective. *IEEE Access*, 2:1660–1679, 2014.
75. Alexander Power and Gerald Kotonya. Bobocep: Distributed complex event processing for resilient fault-tolerance support in iot. In *6th IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2020, Oxford, UK, August 3-6, 2020*, pages 109–112. IEEE, 2020.
76. Jun Qi, Po Yang, Atif Waraich, Zhikun Deng, Youbing Zhao, and Yun Yang. Examining sensor-based physical activity recognition and monitoring for healthcare using internet of things: A systematic review. *Journal of Biomedical Informatics*, 87:138–153, 2018.
77. Adrienne Raglin, Anshuman Venkateswaran, and Huan Liu. Abductive causal reasoning for internet of things. In *2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 1866–1869, 2019.
78. Partha Pratim Ray. A survey on visual programming languages in internet of things. *Sci. Program.*, 2017:1231430:1–1231430:6, 2017.
79. Chris Rayns, Andy Ritchie, Sriram Balakrishnan, Duncan Clark, Phil Coxhead, Daniel Donnelly, Kallol Ghosh, Daniel Millwood, Nicolas Peulvast, Ian Vanstone, et al. *Patterns: Integrating WebSphere ILOG JRules with IBM Software*. IBM Redbooks, 2011.
80. Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade, and Siobhán Clarke. Middleware for internet of things: A survey. *IEEE Internet Things J.*, 3(1):70–95, 2016.
81. H. Röger and R. Mayer. A comprehensive survey on parallelization and elasticity in stream processing. *ACM Computing Surveys*, 52(2), 2019.
82. R. Sahal, J.G. Breslin, and M.I. Ali. Big data and stream processing platforms for Industry 4.0 requirements mapping for a predictive maintenance use case. *Journal of Manufacturing Systems*, 54:138–151, 2020.
83. Aymen J Salman, Mohammed Al-Jawad, and Wisam Al Tameemi. Domain-specific languages for iot: Challenges and opportunities. 1067(1):012133.
84. Ltd. Samsung Electronics Co. Spark CEP, 2021.
85. Claudio Savaglio, Giancarlo Fortino, Maria Ganzha, Marcin Paprzycki, Costin Bădică, and Mirjana Ivanović. *Agent-Based Computing in the Internet of Things: A Survey*, pages 307–320. Springer International Publishing, Cham, 2018.
86. Claudio Savaglio, Maria Ganzha, Marcin Paprzycki, Costin Badica, Mirjana Ivanovic, and Giancarlo Fortino. Agent-based internet of things: State-of-the-art and research challenges. *Future Gener. Comput. Syst.*, 102:1038–1053, 2020.
87. Ognjen Scekic, Tommaso Schiavinotto, Svetoslav Videnov, Michael Rovatsos, Hong Linh Truong, Daniele Miorandi, and Schahram Dustdar. A programming model for hybrid collaborative adaptive systems. *IEEE Trans. Emerg. Top. Comput.*, 8(1):6–19, 2020.
88. Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter R. Pietzuch. Distributed complex event processing with query rewriting. In Aniruddha S. Gokhale and Douglas C. Schmidt, editors, *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS 2009, Nashville, Tennessee, USA, July 6-9, 2009*. ACM, 2009.
89. Kiran Jot Singh and Divneet Singh Kapoor. Create Your Own Internet of Things: A survey of IoT platforms. *IEEE Consumer Electronics Magazine*, 6(2):57–68, 2017.
90. I. Skarbovsky. IBM Proactive Technology Online (PROTON), 2021.
91. Paul Soulier, Depeng Li, and John R Williams. A survey of language-based approaches to cyber-physical and embedded system development. *Tsinghua Science and Technology*, 20(2):130–141, 2015.
92. Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. *ACM Sigmod Record*, 34(4):42–47, 2005.
93. Sriskandarajah Suhothayan, Kasun Gajasinghe, Isuru Loku Narangoda, Subash Chaturanga, Srinath Perera, and Vishaka Nanayakkara. Siddhi: a second look at complex event processing architectures. In Rion Dooley, Sandro Fiore, Mark L. Green, Cameron Kiddle, Suresh Marru, Marlon E. Pierce, Mary Thomas, and Nancy Wilkins-Diehr, editors, *Proceedings of the 2011*

ACM SC Workshop on Gateway Computing Environments, GCE 2011, Seattle, WA, USA, November 18, 2011, pages 43–50. ACM, 2011.
94. Nicoleta Tantalaki, Stavros Souravlas, and Manos Roumeliotis. A review on big data real-time stream processing and its scheduling techniques. *International Journal of Parallel, Emergent and Distributed Systems*, (March), 2019.
95. The JBoss Drools team. Drools Fusion User Guide, 2021.
96. Riddhi Thakkar and Madhuri Bhavsar. 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, Noida, India, August 4-6, 2022*, pages 81–90. ACM.
97. Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156, 2014.
98. Itorobong S. Udoh and Gerald Kotonya. Developing iot applications: challenges and frameworks. *IET Cyper-Phys. Syst.: Theory & Appl.*, 3(2):65–72, 2018.
99. Mirko Viroli, Jacob Beal, Ferruccio Damiani, Giorgio Audrito, Roberto Casadei, and Danilo Pianini. From distributed coordination to field calculus and aggregate computing. *J. Log. Algebraic Methods Program.*, 109, 2019.
100. Kaiyu Wan, Danny Hughes, Ka Lok Man, and Tomas Krilavicius. 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*, pages 1–7. IEEE Computer Society, 2010.
101. Robert Wenger, Xiru Zhu, Jayanth Krishnamurthy, and Muthucumaru Maheswaran. A programming language and system for heterogeneous cloud of things. In *2nd IEEE International Conference on Collaboration and Internet Computing, CIC 2016, Pittsburgh, PA, USA, November 1-3, 2016*, pages 169–177. IEEE Computer Society, 2016.
102. Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 407–418. ACM, 2006.
103. Keiichi Yasumoto, Hirozumi Yamaguchi, and Hiroshi Shigeno. Survey of real-time processing technologies of iot data streams. *Journal of Information Processing*, 24(2):195–202, 2016.
104. Rachid Zagrouba and Amine Kardi. Comparative study of energy efficient routing techniques in wireless sensor networks. *Inf.*, 12(1):42, 2021.
105. Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.
106. Steffen Zeuch, Eleni Tzirita Zacharatou, Shuhao Zhang, Xenofon Chatziliadis, Ankit Chaudhary, Bonaventura Del Monte, Dimitrios Giouroukis, Philipp M. Grulich, Ariane Ziehn, and Volker Markl. Nebulastream: Complex analytics beyond the cloud. *Open J. Internet Things*, 6(1):66–81, 2020.
107. Ariane Ziehn. Complex event processing for the internet of things. In Ziawasch Abedjan and Katja Hose, editors, *Proceedings of the VLDB 2020 PhD Workshop co-located with the 46th International Conference on Very Large Databases (VLDB 2020), ONLINE, August 31 - September 4, 2020*, volume 2652 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.