applied sciences

MDPI

Review

Context-Aware End-User Development Review

Victor Ponce * and Bessam Abdulrazak

Ambient Intelligence Laboratory (AMI-Lab), Université de Sherbrooke, Sherbrooke, QC J1K 2R1, Canada; Bessam.Abdulrazak@USherbrooke.ca
* Correspondence: Victor.Ponce@USherbrooke.ca

Abstract: Context-aware application development frameworks enable context management and environment adaptation to automatize people's activities. New technologies such as 5G and the Internet of Things (IoT) increase environment context (from devices/services), making functionalities available to augment context-aware applications. The result is an increased deployment of context-aware applications to support end-users in everyday activities. However, developing applications in context-aware frameworks involve diverse technologies, so that it traditionally involves software experts. In general, context-aware applications are limited in terms of personalization for end-users. They include configurations to personalize applications, but non-software experts can only change some of these configurations. Nowadays, advances in human–computer interaction provide techniques/metaphors to approach non-software experts. One approach is end-user development (EUD)—a set of activities and development tools that considers non-software experts as application builders. In this paper, we present our analysis of existing EUD approaches for building context-aware applications. We present a literature review of 37 screened papers obtained from research databases. This review aims to identify the methods, techniques, and tools proposed to build context-aware applications. Specifically, we reviewed EUD building techniques and implementations. Building techniques include metaphors/interaction styles proposed for application specification, composition, and testing. The implementations include a specification method to integrate and process context on the target application platforms. We also present the adoption trend and challenges of context-aware end-user development.

Keywords: context-aware; end-user development; literature review; metaphor; interaction style

check for updates

Citation: Ponce, V.; Abdulrazak, B. Context-Aware End-User Development Review. *Appl. Sci.* **2022**, *12*, 479. https://doi.org/10.3390/app12010479

Academic Editor: Byung-Gyu Kim

Received: 8 November 2021
Accepted: 16 December 2021
Published: 4 January 2022

1. Introduction

Internet of Things (IoT) and ubiquitous applications add context-aware techniques to accomplish the desired adaptation given the rapidly growing data produced by different resources (e.g., sensors, services, events) [1]. Traditionally, context-awareness considers end-users (i.e., non-technical people) as receivers of computational services through applications; and software experts (i.e., technical people) as enablers. However, the huge amount of data produced by the flooding of resources increases end-users cognitive load. The end-users are manipulating diverse technologies and smart devices, which increases the elements on which users have to decide. An essential aspect of context-aware systems is to provide mechanisms to augment the transparency [2,3] while decreasing the cognitive load associated with using the system. One dimension of this load is linked to application development and can be handled by improving the usability of application building tools.

Nowadays, building tools are not the only target for software experts; they empower end-users to build/personalize applications by themselves. Thereby, end-users achieve their goals when a building tool's outcome is an application that represents their intentions [4]. A primary challenge is improving usability while applying software concepts to guide application development [5,6]. The usability targets comprehensible representations and proper human–computer interaction, such as adopting interface metaphors. Applying software concepts involves introducing techniques into existing users' workflow, managing the unplanned, implicit, opportunistic, instinctive, and self-priority user intents [7]. Another challenge is to include advanced methods (e.g., machine learning) and cooperative work into application building tools [8].

Diverse solutions in the literature propose usable/useful building tools to empower non-software experts (Section 2). Recent application building tools adopt End-User Programming (EUP), End-User Development (EUD), and End-User Software Engineering (EUSE) for gathering intents and support development [4,6,9]. These EUP, EUD, and EUSE are three fields applied to development tools and are active research topics [9]:

- EUP's central aspect is that programmers build a solution rather than a program [7] (e.g., a spreadsheet solution (i.e., a program) for managing the expenses).
- EUD incorporates methods, techniques, and tools that allow end-users to create, modify, or extend software artifacts [8]. It complements EUP with additional aspects such as testing, debugging, reusing, and sharing artifacts.
- EUSE addresses software quality aspects [7]. It adds systematic and disciplined activities for application development.

We have conducted a literature review to analyze context-aware end-user development, i.e., how application building tools support creating context-aware applications. We applied EUP/EUD/EUSE concepts to guide our review that aims to answer the following research questions:

1. What building techniques are applied to develop context-aware applications by end-users. Building techniques include metaphors/interaction styles (e.g., programming by demonstration) proposed for application specification, composition, and testing.
2. What are the development tools' implementation approaches? The development tool implementations include a description of the method (e.g., machine learning) to integrate, compose, and process context.

We queried research databases selecting 37 relevant papers after screening based on inclusion criteria. Then, we classified the research outcome based on the application building techniques and analyzed tool implementation methods to get insights and challenges of how the tools integrate context to enhance context-aware processes.

The paper is structured as follows: In Section 2, we introduce the main aspects of EUP, EUD, and EUSE applied to develop context-aware applications. Section 3 details the review method that we applied for the study. Section 4 describes the outcome of the review related to the application building techniques. Section 5 examines tool implementations, with limitations and challenges given by the applied techniques and development/deployment platforms. Finally, we present the conclusions of the review in Section 6.

2. Background: End-User Developing Context-Aware Applications

End-users perceive applications' usefulness to meet their requirements (e.g., motives, goals, preferences, needs). To better meet end-user requirements, software developers incorporate user information represented in profile/preferences. Then, computers can correlate individual attributes and available computer services. However, the requirements can vary, and computers must adapt to the variations, e.g., a sick end-user temporarily changes a usual activity for resting. A significant aspect of managing varying user requirements is to empower end-users (non-technical people) rather than software experts (i.e., technical people). End-users are aware of what they need and can directly express their requirements to improve computer service provision.

In this section, we introduce relevant mechanisms applied to empower end-users in building applications. They refer to requirement elicitation, development, and running of applications. Then, we present the involvement of end-user in developing context-aware applications.

2.1. End Users as Application Builders

Using end-user building tools supplement the traditional programming style (i.e., programming with text based on a language) and text-based conventions (e.g., source code indentation). Various end-user building tools incorporate intuitive programming styles (e.g., visual programming languages) and metaphors (e.g., spreadsheet), facilitating requirement elicitation and application development. For example, end-users can create applications using spreadsheets (e.g., OpenOffice Calc) without writing source code. Users can also reuse and share spreadsheet applications; then, other users can improve these applications. Table 1 lists examples of end-user tools and their programming interaction styles/metaphors we found in the literature [6,9]. The table shows how end-user building tools can be augmented, e.g., combining spreadsheets with other development styles/metaphors to improve usability.

Table 1. Examples of end-user development tools.

Programming Interaction Style/Metaphor	**Description or Examples**
Based on language	GUI for programming languages, scripts, extended annotation or parameterization, natural language processing
Based on spreadsheet	Microsoft Excel, OpenOffice/LibreOffice Calc
Programming by demonstration (or similar, e.g., by example, by teaching, by an animation, by a game)	The user "gives" or "shows" the system a reference of the desired behavior (e.g., a prototype, a set of actions). Examples: Stagecast creator, Toontalk
Based on math/specification	GUI for formulas, formal methods, specification language
Visual programming language	LabView, Web Mashups App Inventor (http://appinventor.mit.edu, accessed on 1 November 2021) Scratch (https://scratch.mit.edu, accessed on 1 November 2021) Modkit (http://www.modkit.com, accessed on 1 November 2021)
Rule-based	GUI for rules, e.g., IFTTT (https://ifttt.com, accessed on 1 November 2021)
Combination of styles/metaphors	Forms/3 (Visual and Spreadsheet styles) NaturalMash (Visual and natural language processing styles) Excel macros (Visual, Programming by example and script styles) Web design tools such as Dreamweaver (Visual and HTML languages styles) Node-RED (https://nodered.org, accessed on 1 November 2021) (Visual programming and GUI to add/edit source code)

We adopted End-User Development (EUD) to guide our review. EUD is an approach that aims to empower end-users to build and manipulate applications. EUD subsumes End-user programming (EUP), and both target programming tasks with easy-to-use and easy-to-understand tools [8]. The amount of research in EUD has been increasing through the years [10] since end-user development introduces the advantage of capitalizing on users' knowledge of their requirements, enhancing the development process. Furthermore, researchers also promote end-user software engineering (EUSE) to help non-technical people. EUSE incorporates traditional software engineering methods to EUD [7]. An important consideration from software engineering is the Quality of Software materialized as functional correctness given by users, who know their intentions [4]. However, because people's activities/behavior change, EUSE is characterized by fuzzy, unplanned, implicit, overconfident, and opportunistic intents. Thus, EUD/EUSE is still a challenge for trending technologies in each step of the application lifecycle [6,7,11]. Table 2 summarizes the EUD/EUSE adoption in the application lifecycle with their challenges, which supported our analysis of end-user building tools and methods for context-aware applications.

Table 2. EUD/EUSE challenges.

Step	Description/Progress	Challenges for Context-Aware Applications
Requirement elicitation	End-users are imprecise in what they want or what they need to do. Existing approaches consider: (1) Decrease imprecision by asking (a system then explains behaviors). (2) Similarly, all debugging strategies help when developers refine requirements.	Providing an iterative development with a proper methodology to relate other development steps such as design and debug.
Design/ creation	Visual design patterns such as What You See Is What You Get (WYSIWYG) for direct manipulation and immediate feedback. Techniques to complement design: (1) Waiting for external suggestion/improvement, (2) Provide suggestions (e.g., expert system or combinatorial options), (3) Provide peer suggestions (e.g., search inside an organization), (4) Searches from meta-design [12] (i.e., design for designers [13]) or repositories (e.g., community).	Adapting from traditional software engineering, e.g., design patterns to visual design patterns. With the flooding of smart objects, a new challenge is manipulating the changing context for applications' design/creation.
Reuse/ collaboration	Facilitates problem-solving due to social creativity, but it is a critical part of quality because of error propagation. Support searching in repositories (e.g., by interest), assisting end-users in finding (deciding) appropriate reusable components.	Adapting for heterogeneous end-users with diverse backgrounds (e.g., domain experts, children).
Verification and validation	Verification by immediate feedback (e.g., by visualization). UI is the current validation method, including visual/tangible hints such as colors, arrows, vibrations. E.g., What You See Is What You Test (WYSIWYT [14])—an approach for systematic testing in spreadsheets. Likewise, WYSIWYT/ML [15] introduces Machine Learning (ML) to predict and prioritize software parts to test, enhance measuring coverage, and facilitate monitoring changes.	Verifying/validating applications with appropriate inputs in pervasive computing where smart sensors/objects are diverse in values, format, e.g., the temperature in °C/F.
Debugging	Adaptations from traditional software engineering include insert prints (continuing executions) and assertions (halting executions when false), and trigger debugging options. Advanced techniques include (1) reasoning backward on executed events/interactions, e.g., issue asking-explain; (2) time-traveling debugger, modifying values, pausing, rewinding, and replaying events.	Real-time debugging and debugging dynamic context, e.g., in multiple environments through a city.
Adaptation	Existing approaches consider: (1) connections, with fixed integration (e.g., copy/paste based on a standard data structure) and soft integration (e.g., scripts, macros), (2) implicit coupling defined by the end-user, (3) communication to external entities, e.g., a database, (4) augment/change functionality, e.g., extend trusted Bluetooth devices.	Providing an appropriate integration and extension due to the diversity of resources in ubiquitous technologies.
Deploy	Traditional tools consider fixed adaptation when deployed using parameterization and templates. They apply diverse interaction styles, e.g., (a) direct manipulation using preference forms, wizards, zoom level, color buttons, (b) language using macros, annotations, voice, scripting, natural language.	A challenge is introducing a context-aware adaptation, e.g., to adapt without user intervention.

2.2. Context and Context-Awareness

Enabling technologies, resources, and services for context-aware applications is diverse—all physical or virtual things could be regarded as context [16]. Ubiquitous technologies and IoT increase ambient computational capabilities, making even more resources and services available for applications [17]. Relevant to user and application interaction, context is any information used to characterize the situation of an entity (entity is a person, a place, an object, or an application) [18], including the characteristics of the entity and application's domain [19]. In autonomic computing, entities also represent the self as context, working with locally accessible information that includes their settings, operations, and semantics [20]. Computational systems represent the context through context management processes when the context is available. The system then applies different data processing and reasoning techniques, providing adaptation (e.g., ambient digital support) through context-awareness.

Context-awareness produces a reaction or pro-action based on the context gathered from the self-system and the environment; or integrating knowledge from the entities; the awareness interprets synchronous, asynchronous, situation-based, and social-based information changes. Schilit and Theimer introduced context-aware computing as a capability of the applications to discover and react based on variation in the environment [21]. Dey and Abowd extended the scope of awareness, highlighting the context's significance, where information and services are relevant to the user's task [22]. Roy et al. considered the scope of context-aware agents' mission performing their micro assessments [16].

Due to the diversity of resources in environments and available technologies, new techniques/metaphors are available to approach non-software experts to develop context-aware applications. This review regards context as a digital representation of real conditions, considering internal and external system aspects that provide features for context-awareness.

2.3. Context-Aware Application Development Involving End Users

End-user empowering for application/service development has been a goal in pervasive computing [23]. An approach is to focus on end-users and involve them in the application design process [24]. User-centered design/development (UCD) is an applied framework to involve end-users in context-aware application development [25,26]. However, modeling context (and mainly user activities) using UCD is challenging because the activity context is dynamic, implicating unfeasible management of the design/implementation (e.g., modeling all user activities). Likewise, testing is critical because it is challenging to consider all users and situations' contexts.

To overcome the complexity of involving end-users in the application design process, context-aware application development targets end-users as application builders. A first approach is to model user and situation profiles and allow end-users to change fixed configurations. For example, the design of activity models regarding limited user profile/environment and fixed activities [19,27]. The disadvantage is that this approach restricts end-users to the available parameters for personalization.

Another approach is programming frameworks: mobile computing's progress also provides solutions that assist end-users in building applications. Emerging frameworks simplify complex application development, including visual programming and techniques, e.g., Apache Cordova [28] enables mobile web application development for various mobile platforms. Through plugins, it allows for managing the context of devices and integrates external services. Others are platform-specific or purpose-specific frameworks, e.g., App Inventor [29] facilitates application development for non-technological expert users. Modkit [30] uses Scratch [31,32] for Arduino [33] (Arduino is an open-source electronics platform. It allows integrating sensors and actuators by coupling boards) to build interactive hardware projects without advanced hardware programming knowledge. Catrobat [34] is a mobile end-user programming solution inspired by Scratch for developing educational applications. The disadvantage is that most context-aware programming frameworks require software experts (i.e., technical people) to adapt generic functionalities to targeted applications.

End-user development tools are more straightforward mobile end-user tools (not frameworks) accessible to users to create personalized applications or own actions. These tools allow for using ambient resources to support user requirements. For example, Ur et al. evaluated IFTTT [35] that helps to create condition → action rules (if-then rules called Recipes) from pre-defined services available as building blocks (called Channels). In IFTTT, a diverse context can exist (called Ingredients) and depends on the available services. The IFTTT application builder tool is available on various mobile platforms and the Web. It integrates diverse channels, e.g., Facebook, Dropbox, Android device context, weather, e-mail. Users can discover, access, and contribute to collections of shared Recipes. Users can then personalize the recipes, e.g., download into Dropbox the Facebook photos where the user is tagged. Lucci and Paternò [36] analyzed three mobile tools that release the technical knowledge for programming context-aware applications: Tasker (context → action), Atooma (if → then), and Locale (condition [setting]. These tools incorporate user-friendly vocabulary and interaction to develop applications but limit logical operations and expressiveness. In the current era of cloud computing and the Internet of Things, EUD is also progressing. New solutions involve the integration of smart objects and cloud computing services. For example, IFTTT includes intelligent assistants and smart home provider channels such as Amazon Alexa [37] and SmartThings [38].

Similarly, Workflow for iOS [39] (now Shortcuts for iOS and iPadOS) is another tool for composing a workflow (sequence of actions). To create a new application, users drag and drop pre-existing actions and create a workflow representing the user's intention on the application. For example, take a picture, save to Dropbox, tweet the picture, post the picture to Facebook, and then send it by e-mail. Users can add a complete application to the quick launching menu or home screen for a future run or plug the workflow into other mobile device's applications. Workflow for iOS includes a shared gallery for contributing and discovering other users' applications.

2.4. The Rationale for the Review

End-user development promotes direct end-user elicitation rather than a developer (i.e., not the end-user) elicitation/interpretation of end-users knowledge into the system's knowledge [40]. New tools empower end-users who have an in-depth knowledge of their activities and evolve the applications in the presence of changes [41]. Nowadays, end-users are active actors in software adaptation and can build applications tailored to their needs. However, EUD challenges comprise having useful/usable development tools while accessing dynamic context for context-aware applications. This review provides an overview of context-aware application building techniques and implementation (i.e., what is behind the tools). We also discuss the challenges to keep pervasive systems adaptation while enhancing applications with functionalities that allow end-users to fulfill their needs. Frameworks (e.g., Modkit for Arduino) or other complex tools (e.g., for industrial control such as SCADA—Supervisory Control and Data Acquisition—allows gathering data from control hardware in industries such as oil/gas distribution control systems) can also be categorized as end-user development tools. However, we review research efforts with simpler tools and techniques for testbed implementation or rapid simulations.

3. Review Method

We conducted a literature review with the following research question: What EUP, EUD, and EUSE approaches and techniques researchers have proposed to build context-aware applications? The method (Figure 1) begins with defining research questions and the search string for searching in research databases. Then, we screened the papers following the inclusion criteria presented in Table 3, selecting 37 research efforts for analysis. Finally, we evaluated the selected papers and classified them for review.

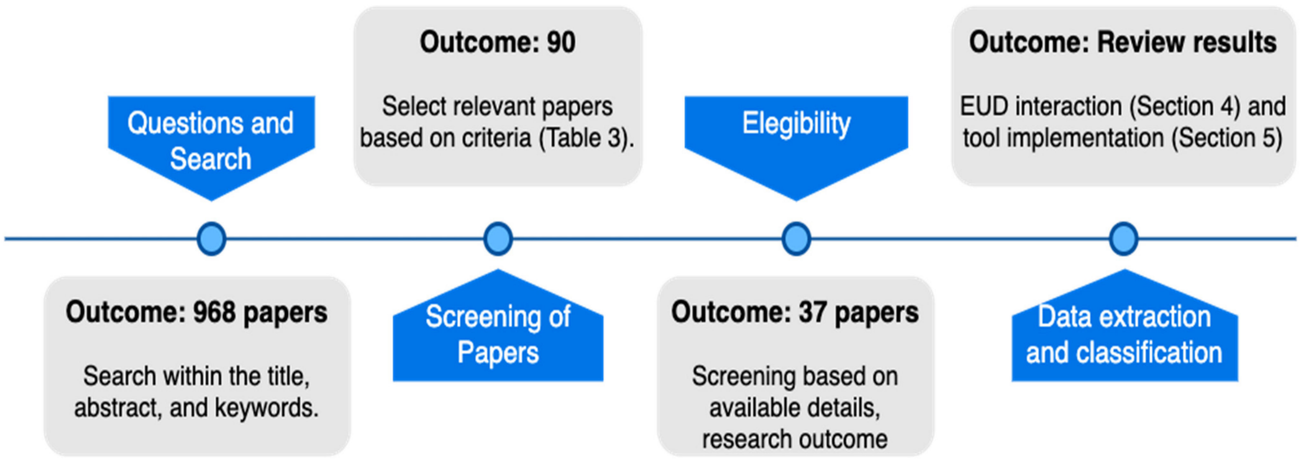
Questions and Search
Outcome: 968 papers
Search within the title, abstract, and keywords.
Outcome: 90
Select relevant papers based on criteria (Table 3).
Screening of Papers
Elegibility
Outcome: 37 papers
Screening based on available details, research outcome
Outcome: Review results
EUD interaction (Section 4) and tool implementation (Section 5)
Data extraction and classification

Figure 1. Review method.

Table 3. Screening of papers.

Consideration	Criteria for Inclusion	Phase
Year of publication	No regarded but restricted by the search keywords: "context-aware" + EUP, EUD, EUSE. The search results show research from 2001.	1st, 2nd, 3rd
Relationship with the Subject	The tool includes a visual/graphical interface for context-aware application development. The paper must consider a sensor/actuator context integration/simulation and an adaptation involving such context.	1st
Type of document	The research is published in proceedings or journal papers indexed on Scopus, IEEE, or ACM.	1st
Available and comprehensible	The paper presents arguments and details related to EUP, EUD, EUSE, and UI concepts with enough information to evaluate.	2nd
Research outcome	The paper describes a solution with the implementation of a tool for development. We discarded when a paper only refers to service integration, e.g., using IFTTT, Web mashups, or templates without describing the "programming" implementation.	3rd
Target end-user	For technical users (i.e., with some software knowledge) and non-technical users. We discarded when the target is exclusively for software experts.	3rd

3.1. Research Questions and Generic Search String

The following are the specific dimensions that we intend to analyze:

1. What building techniques are applied to develop context-aware applications by end-users? We analyzed metaphors and interaction styles applied to create, modify, test, share, and deploy applications in this dimension.
2. What are the development tools implementation approaches? In this dimension, we analyzed the implementation technology of the tools, including context-awareness.

We conducted searches in SCOPUS, IEEE, and ACM. Although other sources could be considered, we selected these sources because most of their indexed journals are accessible to the university community. We searched using the title, abstract, and keywords with the following generic string:

(end-user AND (program* OR develop* OR software*) AND context-aware*)

3.2. Inclusion/Exclusion Criteria

We divided the inclusion/exclusion criteria into three phases of relevant literature for the review (Table 3). In the first phase, we queried research databases obtaining 968 documents. We then obtained 492, discarding papers mainly because they are duplicate, inaccessible (due to license restrictions), or unrelated to the subject. In the second phase, we screened in detail the papers, primarily based on the available details. The major criteria are comprehension aspects related to UI building techniques (based on EUP, EUD, EUSE). After the second phase, we obtained 90 for an in-depth reading. In the third phase, we selected 37 papers for the review, based on the following considerations:

- Research outcome (Table 3): The paper includes enough detail to answer the research questions. We discarded tools that are not described at the implementation level.
- Target end-user (Table 3): End-user application building tools have been proposed for software experts (technical people) and non-software experts (non-technical people). Even though traditional end-user empowering mechanisms [8,40] involve many users, we consider end-user development for non-software experts. However, some approaches propose a level of software experts' involvement before end-user development (e.g., to create APIs, models, meta-design/collaboration). We included those approaches because they target an end-user development process with fewer restrictions.
- On the other hand, we discarded approaches that require software experts or advanced technical aspects. For example, the Context Modeling Toolkit (CMT) proposes a context model to augment the template-based development of rules [42]. We discarded CMT because the authors propose the involvement of software programmers in the development process. Similarly, other solutions use human-readable configurations (e.g., plain text, XML). We also discarded them because it is highly probable that non-technical users (i.e., non-software experts) encounter difficulties configuring required resources, actions, and services.

3.3. Data Extraction and Classification

Table 4 lists the analyzed papers by year, summarizing the programming interaction and composition metaphor. The reviewed papers target two categories of end-users: technical end-users (i.e., users with some knowledge of software) and non-technical users. Figure 2 shows the distribution of research over time, with 26 out of 37 papers targeting EUD only for non-technical users. The papers presented implementations for a diversity of context, development, and running environments (Table 5).

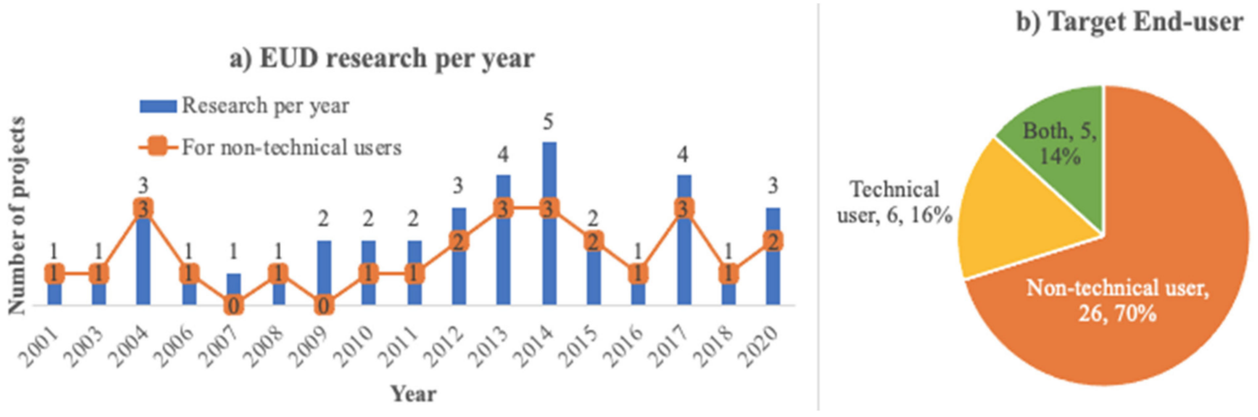

Figure 2. (**a**) EUD research over time; (**b**) target end-user.

Table 4. EUD techniques and metaphors.

Paper	Year	EUD Interaction		Metaphor *
		Non-Technical	**Technical ***	
AutoHAN [43]	2001	Advanced interaction		A 2D adjacency of cubes
iCAP [44]	2003	Demonstrate rules		Sketch rules
a CAPpella [41]	2004	Demonstrate actions		Stream of information
CAMP [45]	2004	Assembling pieces		Magnet poetry
Topiary [46]	2004	Demonstrate actions		Map, storyboarding scenarios
Context Studio [47]	2006	Selecting rules		Folder-file
VisualRDK [48]	2007		Assembling pieces	Visual blocks represent programming constructs
ActivityDesigner [49]	2008	Demonstrate rules		Storyboarding activities + autocomplete and selection
Situations API [50]	2009		API	API
Persona [51]	2009	Selecting rules	API	Document-centric/API
iPlumber [52]	2010		Language	Rule language
Atomate [53]	2010	Selecting rules		Global address book
OPEN [54]	2011	Selecting rules	Language	App templates/Rule language
MicroApps [55]	2011	Flow		Jigsaw (blocks + visual aids to represent types and connection compatibility)
Preuveneers et al. [56]	2012	Assembling pieces		Visual blocks representing sensors/actuators
IVO [57]	2012		Model-based	Workflow
MNFL [58]	2012	Flow		Blocks represent programming constructs and input/output, visual aids to represent types and connection compatibility
Rodríguez et al. [59]	2013		Assembling pieces	Nodes and arcs represent RDF graphs; blocks represent statements and operators
Context Cloud [60]	2013	Selecting rules		Widgets
GALLAG Strip [61]	2013	Demonstrate actions		Comic strip
SSB [62]	2013	Selecting rules		Widgets
S-API [63]	2014	Selecting rules	Flow	Icons representing commands/API/graph
VS-CaSP [64]	2014	Selecting rules		3D smart building
SPOK [65]	2014	Language		Smart keyboard
Puzzle [66]	2014	Flow		jigsaw + visual and tap aids to compose
Mayer et al. [67]	2014		Language	Sensing and actuation primitives
Martín et al. [68]	2015	Selecting rules		Widgets, map, WYSIWYG editor
Dobby [69]	2015	Assembling pieces		Programming by selection (selection of blocks)
T4Tags 2.0 [70]	2016	Advanced interaction		Tangible tokens attached to objects/appliances
SITE [71]	2017	Selecting rules	Language	Widgets/scripts
CoRE [72]	2017	Language		Editor to edit logical expressions
TARE [73]	2017	Selecting rules		Widgets with colors for context categories
EPIDOSITE [74]	2017	Demonstrate actions		Widgets (smartphone UI)
GrOWTH [75]	2018	Specify goals		Natural language abstraction for Web-based app, voice, mobile app (widgets)
VASH [76]	2020	Demonstrate actions		Web browsing actions + voice commands + selection
EasyContext [77]	2020	Selecting rules	Language	Cards to represent contextual rules
CAPturAR [78]	2020	Advanced interaction		Virtual object drawing, with a human avatar

* API: Application programming interface. WYSIWYG: What You See Is What You Get. RDF: Resource Description Framework.

Table 5. EUD implementation approaches.

Paper	Tool Implementation		Design *			Context	Running *			
	Approach	Expressiveness	PC	Mo	Ph		PC	Mo	Pl	Ph
AutoHAN	Language	Ontology, linguistic terms	✓		✓	Appliances, virtual devices			✓	✓
iCAP	Rule-based	IF-THEN	✓			activity, location, people, time			✓	
a CAPpella	Machine Learning	Annotations	✓			Sensors (e.g., video), actuators (e.g., turn on/off a light)	✓			
CAMP	Language	Vocabulary (a subset of natural language), reword dictionary	✓			Smart home (e.g., cameras, interactive displays)			✓	
Topiary	Algorithms	Spatial relations (e.g., out, enters) and proximity (e.g., near)	✓			Location (i.e., people, places, things), temporal		✓		
Context Studio	Rule-based	Ontology, IF-THEN		✓		Mobile (e.g., location), actions (e.g., call contact)		✓		
VisualRDK	Language	Events; commands for context (e.g., set), for a process (e.g., wait)	✓			All, represented as an abstract component	✓			
ActivityDesigner	Algorithms + Language	Activity model, activity query language, speech matching	✓			Activities, simulated context, sensors (e.g., location)	✓	✓	✓	
Situations API	Rule-based	References (input context), parameters, listeners	✓			Sensors, users			✓	
Persona	FSM	Past preferences	✓		✓	Artifact, action, interaction modality, contextual	✓			✓
iPlumber	Rule-based	Domain ontology (e.g., smart home)	✓			All, integrated with context wrappers	✓			
Atomate	Rule-based + Language	Simplified natural grammar	✓			Personal information (e.g., people, places, events), time	✓			
OPEN	Rule-based	Domain ontology, high-level ontology, application ontology	✓			All, integrated with context wrappers	✓			
MicroApps	Language	Pre-conditions (trigger), implicit loops		✓		Sensors, Web services, mobile (native components)		✓		
Preuveneers et al.	Rule-based	Ontology	✓			Smart home devices	✓			
IVO	Rule-based + FSM	IF-THEN-ELSE, events, geographical, proximity	✓			Temporal, proximity, location		✓		
MNFL	Language	Blocks for computing (e.g., opposite, combine math)	✓			Video/audio, motion sensors	✓			
Rodríguez et al.	Rule-based	Ontology, IF-THEN, loops	✓			All, integrated with ontologies for smart spaces	✓			
Context Cloud	Rule-based	IF-THEN	✓			All, integrated with providers			✓	
GALLAG Strip	Rule-based	IF-THEN		✓		Smart home sensors		✓		
SSB	Rule-based	Register/subscribe service rules	✓			Home appliances and sensors			✓	
S-API	Rule-based + Language	Language type compatibility. Commands for functions, events	✓	✓		All, integrated with ontologies for cross-smart spaces	✓	✓		
VS-CaSP	Rule-based	IF-THEN	✓			Virtual and real sensors	✓			
SPOK	Rule-based + Language	Pseudo-natural syntax, IF-THEN, WHILE, WHEN		✓		Smart home and Web services			✓	
Puzzle	Language	Jigsaw language		✓		Sensors and Web services		✓		
Mayer et al.	Language	Get/set value (sensor/actuator), stateless actuators (e.g., trigger)	Undefined			Smart things	Undefined			
Martín et al.	Rule-based + Algorithms	IF-THEN, proximity, location algorithms	✓			All, integrated with providers			✓	
Dobby	Rule-based + Language	Language for selection, IF-THEN	✓			*-as-a-service			✓	
T4Tags 2.0	Rule-based	WHEN(trigger-condition-action)		✓	✓	Sensor, temporal, service		✓		✓
SITE	Rule-based + Language	Fuzzy rules, context-free grammar	✓		✓	Smart home	✓			✓
CoRE	Rule-based + Language	Except, when, if-then, and based on natural language	✓			All, integrated by middleware			✓	
TARE	Rule-based	IF-THEN, WHEN, AND, OR	✓			All, defined as category, subcategory, entity			✓	

Table 5. *Cont.*

Paper	Tool Implementation		Design *			Context	Running *			
	Approach	Expressiveness	PC	Mo	Ph		PC	Mo	Pl	Ph
EPIDOSITE	Rule-based	Annotations		✓		IoT (Smart home) and IR, accessed by phone apps		✓		
GrOWTH	AI planning	Ontology	✓			All, integrated by components of the architecture	✓			
VASH	Language	Annotations	✓			Web elements, mouse, keyboard events	✓			
EasyContext	Rule-based	IF-THEN based on a Backus-Naur form (BNF) grammar	✓			Based on Google Awareness API		✓		
CAPturAR	Rule-based	IF-THEN, events			✓	Human actions, context attributes, events	✓			✓

* PC: Personal Computer. Mo: Mobile. Pl (Platform) refers to a context or a service platform. Ph (Physical) refers to running applications that involve physical objects such as smart objects, objects with tags, media cubes, virtual/augmented reality.

In the following Section 4, we describe the review's outcome related to the application building techniques. Then, we present the implementation approaches and challenges for future research in Section 5.

4. Application Building Techniques

The reviewed papers revealed different techniques for the application building process. Some tools adopt well-known metaphors and interaction styles in EUP/EUD/EUSE [6], e.g., programming by demonstration. Others adapt development artifacts (e.g., language, ontologies, models) to facilitate the development process by reducing the technicalities. In this section, we focus on the context-awareness building steps of the development process. We analyzed the papers and identified 11 techniques. Even though some tools applied more than one technique, we describe the most relevant technique applied for the tool.

4.1. Programming Artifact Abstractions

The first technique focuses on adaptations from traditional software development. Specifying actions using programming artifact abstractions targets technical users (i.e., users with some knowledge of software). The technique decreases the complexity of traditional programming components, e.g., API, scripts. As a result, the components become usable/useful for non-software experts.

Situation API, presented by Dey et al. [50], is an abstraction of context-aware infrastructure to facilitate building applications for developers and interface designers. The aim is to improve the end-user interaction in two aspects: intelligibility (i.e., users understanding application behavior) and control (i.e., users changing/controlling context processing). The authors implemented a default Situation to support the traceability/debugging of applications. It allows us to visualize the state (e.g., a context value, parameter, rule) of other Situation components and change context values and parameters at runtime to debug.

Persona, presented by Kawsar et al. [51], is a toolkit to generate user interfaces for context-aware middleware. Afterward, end-users can customize and control applications through preference documents, i.e., interaction parametrization in a document-centric collaboration among system components. Persona provides APIs, plugins, and an interface engine to enable software developers to build applications. Developers represent possible user interactions, and the system generates (when deploying) user preferences/controls in an XML document. Then, end-users interact with the tool by selecting their preferences.

4.2. Model-Based Specification

An extended programming abstraction technique is model-based [8], where the application specification uses a model, i.e., pre-defined concepts with rules (e.g., a model for actions and temporal constraints). End-users can follow the model to build applications, creating a workflow connecting concepts as input—process—output. Then, the system uses the model to interpret the defined actions, generating an interactive application.

IVO (Integrated Virtual Operator), presented by Realinho et al. [57], is a framework to build and deploy applications by associating context conditions with activity workflows. It includes three tools: IVO Outlook, IVO Builder, and a mobile client.

- IVO Outlook allows defining workflows based on temporary and proximity conditions through an Outlook Calendar and Outlook Contacts add-ons, respectively [79].
- IVO Builder is a visual UI to create workflows of activities based on location context. It allows for manipulating geographical representations to create areas of interest through polygon or circle. IVO Builder can import geo-referenced areas from Wikipedia and Wikimapia and then facilitates map control (e.g., zoom, map views), including scripting for advanced users. A user associates the areas of interest with related information (e.g., website link) and then creates workflows in the area. Afterward, the user defines actions to be executed in both "on enter" and "on exit," e.g., change phone profile, update Facebook status. Available actions include filling forms and quizzes created by the IVO Builder tool.
- The mobile client provides the workflow functionalities, activating actions when the conditions match. It includes a UI that is either invisible (if not required), or a visual interface for user interaction, e.g., maps, augmented reality browser, forms, quizzes.

4.3. Simple Flow of Components

Diverse approaches simplify models proposing a simple flow of components (e.g., commands, actions, language-based). End-users create applications in easy steps by connecting components in sequence or parallel, e.g., pipeline [80]. The development tool includes techniques for flow validation, such as a list of compatible components and icons or colors for compatible connections/actions.

S-APIs (Semantic APIs), presented by Palviainen et al. [63], is a command-based programming abstraction for building smart space applications. End-users connect commands, forming sequences and branches, and the system provides composition input/output validation. End-users can build *command-oriented* and *goal-oriented* applications using UI editors that facilitate the use of S-APIs. *Command-oriented* programming provides two visual approaches: graph-based editor for desktops (Smart Modeler) and list-based editor for mobiles (RDFScript Creator). In graph-based development, a user attaches and connects command/branches, creating the application sequence. The editor provides type compatibility between input/output and checks inputs configuration. In list-based development, a user creates a command sequence by adding/removing commands, defining inputs, and connecting the command to the output branch of the previous command. *Goal-oriented* programming is performed in the RDFScript creator, enabling end-users to define goals achieved by executing a sequence of commands, e.g., if location changes, and I arrive home; then, turn on the TV.

MicroApps, presented by Cuccurullo et al. [55], is a tool to create applications in mobile phones through a jigsaw-based visual editor. The tool integrates pre-existing applications managing services (e.g., camera, contact, mail), making them available, exposing their list of actions (e.g., camera.take, camera.preview). Icons represent actions for dragging and dropping in a composition area. End-users can connect actions in sequential or parallel and define preconditions (triggers), inputs/outputs of actions, and implicit loops, i.e., implicit objects generated based on outputs. The visual editor helps through the composition process. It highlights action icons by compatibility, disables not compatible columns, and represents inputs/outputs with shapes and colors (e.g., a red bullet means text string, a cyan bullet means contact data, a triangle means an available parameter for composition). The visual editor also includes aids such as color descriptions when a user long-press an action icon, allowance to associate parameters in parallel actions and undo operation to remove the last action or association [81].

4.4. Visual Language-Based Flows

Visual language-based flows augment the interaction for connecting components in sequence or parallel. End-users create flows through graphical components rather than using only commands or actions. The development tool includes an enhanced visual representation of each component (e.g., a block representing a programming construct). It also enhances visual aids for flow validation, such as shapes for compatible components, connections, or actions.

MNFL (the Monitoring and Notification Flow Language), presented by Edgcomb [58], is a graphical flow language to build monitoring applications, i.e., an application that uses data from sensors and cameras to detect and notify situations of interest. The semantics include building blocks to represent sensing (e.g., audio/video, number, button), computing (e.g., opposite, combine, math), and actuating (e.g., send a message, display number, beeper). The language lets us connect building blocks with inputs (at left for one variable or top otherwise) or outputs (at right). Inputs/outputs types of graphical representation include both shapes for Booleans (triangle) and numbers (trapezoid); and icons for pictures and sounds. The language also includes blocks representing data duration types, i.e., snapshot, stream, and snippet types, to represent instant, continuous, and portion-time. Diverse colors facilitate the development/execution, e.g., a red triangle means "no" (i.e., Boolean false), green when yes, and yellow when it is an error/not present. The language implements other visual supports, such as showing the current number when the cursor is over a trapezoid and an "i" icon to open the block's configuration window.

Puzzle, presented by Danado et al. [66], is a framework to build, edit, and run end-user applications on mobile devices. End-users combine visual blocks in a jigsaw-based visual editor through drag and drop. The editor supports the developer with a color help system, hints, and options [82]. A new set of interactions includes deleting by dragging a jigsaw into a trash can, intuitive taps for configuration, and further helps and simplified menus [66]. A jigsaw piece includes an input (inner circular shape on the left) and an output (outer circular shape on the right) to receive and send information. Two pieces can connect when a piece's output color matches with the next piece's input color.

4.5. Assembling Pieces

Other visual programming languages release the restrictions of a sequential connection (flow). An approach is assembling pieces (e.g., blocks, tiles) [80]: specify actions/rules by joining visual components (pieces) that represent programing language constructs or sensors/actuators, e.g., block-based [83], jigsaw [29,32]. In assembling pieces, the components themselves provide the connection validation, generally through shape compatibility.

VisualRDK [48] is a high-level visual language targeted to non-computer novices. A development environment presents context by querying sensors or connecting (as plugins) to a context server. The language allows building programs with simple logic in two classes of applications: Automation and communication. *Automation applications* are contextual-reaction programs such as office automation and spatial information systems. *Communication applications* include users triggering actions based on pre-defined roles, e.g., an application for support lectures or meetings. A developer builds an application by dragging and dropping components (a piece of hardware or software) and context (location, situations, nearby persons, and available components). The final application can generate two implementations from one source: prototype and debug versions. *The prototype version* is a distributed version based on the application, and the *Debug version* is a centralized interpreter executable where all devices are remote-controlled, simplifying debugging.

Rodríguez et al. [59] proposed a UI for prototyping smart space applications based on the semantic Web. The UI presents diverse graph-based data (ontologies and datasets) representing smart spaces' context. End-users compose applications by dragging and joining building blocks representing an RDF graph structure. The blocks include nodes (i.e., class, data property value), arcs (i.e., data property, object property), statements/operators (i.e., loops, and, or). The UI includes visual support such as color to distinguish literals from ontology classes and visualize the graphical structure, operators, and statements. Even though more blocks (e.g., computing, service, semantic web components) allow building richer applications, learning the meaning of the blocks is required.

4.6. Simplified Meaning of Pieces

Other approaches simplify the meaning of visual programing by assembling pieces. The simplification is achieved using natural language, ontologies, or domain-specific languages, targeting non-technical users. The development tool includes an abstraction of complex programming constructs, providing an instinctive interaction to connect compatible components.

CAMP, presented by Truong et al. [45], is a tool that introduces a magnetic poetry metaphor for programming smart home environments. Users combine flexible magnets that contain a printed word, creating "poems" or statements, e.g., "dinner happens between 7:00 and 9:00 P.M. in the dining room." A vocabulary set contains color-coded categories belonging to who (e.g., I, everyone), what (e.g., picture, video), where (e.g., kitchen, everywhere), when (e.g., always, morning), and general (e.g., remember, save). A UI assists users in the composition to highlight matching words with the first introduced letter and select, drag, and re-order words. Users can extend the vocabulary by combining words, creating a new magnet.

Preuveneers et al. [56] present a framework to configure home environments. A composition tool enables end-users to create activities, i.e., building blocks representing sensors/actuators (e.g., a required light setting). A composition manager facilitates the composition, gathers available services, and presents available blocks to compose following an ontology model. The ontology includes a BlockItem class representing a device and a Mapping class to relate values to blocks, e.g., slider-mapping to pick a value between min and max values. The ontology is used to adapt the composition tool to end-users.

Dobby, introduced by Park et al. [69], is a system to support programming by selection (PBS)—a method for creating end-user context-aware services by making a series of selections to define rules. The development environment includes a visual language to facilitate using the PBS method. End-users create applications by the selection, supported by a visual language to connect building blocks. The UI shows blocks structured like a file system, facilitates the interaction with a double click for zooming areas, presents available attributes/values, and uses the language to support the composition.

4.7. Languages to Specify Rules

Other approaches also involve languages to specify rules. Examples are ontology-based languages, domain-specific languages, and natural languages. The development tool reduces the complexity of the language for developers, and the target application can have a robust implementation (based on the language).

iPlumber, presented by Guo et al. [52], exploits the meta-design method [12] to create, share, and reuse context-aware applications. The system provides tools to create application templates (or meta-applications) based on an ontology rule language. A web sharing center allows for sharing the applications. End-users can check recommendations (e.g., user ratings, number of users) to decide which application to use. Afterward, end-users become application co-designers, performing configurations to meet their needs, personalizing the application. End-users also can simulate sensing values to test rules or test applications with real sensing values. Guo et al. also introduced OPEN [54] to exploit the cooperation among developers, end-users, and between developers and end-users. They extend the ontology rule language to involve a broad category of users. They propose three programming modes/complexities to create/modify applications: incremental mode (high-level users who create/manage rules), composition mode (middle-level users who modify rules), and parameterization mode (low-level users who set parameters).

Mayer et al. [67] introduce a domain-specific model and language to facilitate the development of applications to control smart things—i.e., things that interact and communicate with each other and with physical and virtual entities. The authors proposed abstracting sensing and actuation primitives. Then, the primitives' implementation is independent, e.g., graphical widgets, physical interactors, speech/gesture commands. The authors reify the abstraction as a language that implements an interactive component representing a physical or virtual component not reasonably sub-divisible (e.g., light switch). Interactive components communicate with interactors—i.e., a device or software used to interact, e.g., graphical interactors such as gauges, click wheels, knobs. The language represents interactive model components for storing, querying, and composing with a UI.

SITE, presented by Häkkilä et al. [71], is a smart home solution to specify, monitor, and control IoT smart objects (SOs). The UI was developed using the LabVIEW [84] programming environment. The UI lists all available SOs, and the user selects the required ones. Then, the UI presents SOs information and the available options, i.e., visualize (monitor) real-time sensor measurement, configure rules, and run/stop controller to process gathered data applying fuzzy rules and generating actuation commands. The UI also allows scripting for advanced users.

EasyContext, presented by Ponciano et al. [77], is a visual tool to generate rules for context-aware mobile applications. The authors propose an abstraction of the Google Awareness API [85] and a Web-based tool that helps developers to create the rules. The abstraction is based on context-free grammar to define conditions and actions. Based on the grammar, the tool then generates a configuration file to be integrated into mobile devices.

In general, languages are targeted to technical people, requiring learning the language concepts. Other approaches propose targeting non-technical people with natural language. Coutaz et al. introduced SPOK [65], a development environment for smart home context-aware applications. Non-technical users develop applications in clients with a simplified language. End-users can compose applications using words from the language with a smart keyboard, which updates the available words based on the current entry and home state. Metaxas et al. introduced CoRE (context-range editor) [72] which includes a constrained natural language text editor to edit logical expressions. Noura et al. introduced GroWTH [75] with a natural language abstraction for web, voice, and mobile applications.

4.8. Select/Demonstrate Rules

Select/demonstrate rules [40] is a well-known approach for programming where end-users define condition $\rightarrow$ action rules by selecting conditions/actions/services/commands from UI elements such as lists, menus, dialogs, widgets. Some of the previous approaches adopted the select rules method to augment the usability, e.g., Persona [51], OPEN [54], S-APIs [63], SITE [71], and EasyContext [77]. Persona allows end-users to select generated preferences from an options list (e.g., yes/no). OPEN extends its ontology language for non-technical users proposing a UI for modifying and parameterizing rules. S-APIs restrict the creation of flows (called goal-oriented applications) to a sequence of commands that define rules, e.g., if location changes, and I arrive home, then turn on the TV. SITE proposes a UI (based on LabVIEW) to abstract its language for defining fuzzy rules. The EasyContext Web tool presents actions and contextual clauses in cards (card metaphor). The cards are divided into categories (conditions, actions), and the cards' content is interactive, presenting dynamic values according to user inputs. In the following, we present other approaches that adopt the condition $\rightarrow$ action rule pattern for a simplified specification of applications.

Context Studio, presented by Korpipää et al. [86], part of a context-aware framework [47], is a tool to program applications in small-screen mobile devices. Context Studio is a customization component to define rules based on input context (e.g., location, environment sound, RFID-based commands) and actions (e.g., set profile Silent, call a contact). According to a vocabulary hierarchy, it adopts the folder-file UI metaphor: context and actions as folders and subfolders. Then, a user navigates through the hierarchy and selects triggers and actions to configure rules. The UI allows us to create, edit, copy, activate/deactivate and delete rules and include visual icons representing rule states and options/menus to facilitate the navigation.

Atomate, presented by Van Kleek et al. [53], is a tool for defining context-aware behaviors (rules) for web-based personal information management. The tool integrates web feeds to generate/update a world model that enables context-reactive automation based on rules. The world model includes information about the physical world (e.g., people, places, events), digital items (e.g., messages, articles, tweets, e-mails), and system information (i.e., rules, predicates, data sources). End-users can configure rules via direct manipulation in a UI based on a global address book metaphor representing the world model. The UI presents info by type, e.g., people, places, and events. Then, it allows us to select the desired type and define rules. The UI also allows us to explore the world model, allowing us to assign/edit properties and merge redundant items.

Context Cloud, presented by Martín et al. [60], is a platform to build context-aware applications for end-users. The platform provides a web front-end to facilitate programming, supported by a proposed methodology to guide situations identification and parameterization. The web front-end allows specifying providers, context models, mappings, areas (polygons on Google Maps), and rules. It consists of widgets and dialogs to define rules with priority, operations (e.g., equals, greater than), and a calendar option to define date ranges. Martín et al. also improve Context Cloud UI to target domain-expert development [68]. The tool allows virtual geographical areas creation (indoors or outdoors) and proximity definition. The tool also includes widgets to specify date/time and a WYSIWYG editor for locales (Spanish, French, and English) notifications.

Sensor service binder (SSB), presented by Nakamura et al. [62], is a visual UI to orchestrate context-aware services in-home network systems (HNS). The aim is to facilitate end-users building applications based on Web technology implemented in their HNS service-oriented framework. The UI parses the WSDL and XML representing sensor/appliance services and enables end-users to interact in a user-friendly form through buttons, lists, and text boxes. Similarly, Corcella et al. introduced TARE [73], which defines all context as category, subcategory, and entity. Then, end-users specify rules selecting widgets with colors for context categories.

VS-CaSP, introduced by Su et al. [64], is a system that facilitates non-technical users authoring 3D environments for context-aware applications. The architecture consists of three modules: Context manager, 3D authoring, and 3D simulation and hardware integration. *Context manager* manages context resources such as 3D objects, models, scenes, and scenarios (context for rules and 3D models). *3D authoring* facilitates building application scenarios, including a 3D editor and UI tools to specify rules with conditions (data from sensors) and actions. *3D simulation and hardware integration* enable debugging and simulation of applications using virtual and real sensors.

4.9. Illustration of Rules

Illustration of rules is another approach—end-users depict/sketch rules, facilitating the definition of conditions and actions. The key point is to overcome the complexity of the user interface elements, such as showing/hiding dialogs distracting end-users. The development tool includes frames, shapes, or templates to guide the composition of applications, simplifying the overall development process for a situation.

iCAP, introduced by Sohn et al. [44], is a visual tool for prototyping context-aware applications. The tool allows for defining the input/action context, compose rules, and testing. Using a drag-and-drop technique, the composition process allows for creating situations/actions by sketching inputs and actions in a rule-based specification. The testing allows checking rules in a run mode. In this mode, users can change input values and evaluate the behavior of applications.

ActivityDesigner, introduced by Li et al. [49], is a tool for activity-based prototyping of ubiquitous applications. The tool includes an activity language for the description of activity characteristics. The tool also includes activity abstractions to extend the expressiveness, i.e., Scenes representing the activity's actions and Themes representing everyday aspects (e.g., eating healthy). Then, end-users prototype applications using storyboarding to combine Themes and Scenes in a GUI. Then, end-users define transition conditions using the activity language. The tool allows simulating and testing individual activities and multiple prototypes simultaneously, as well as running tests. In running tests, an Activity Server controls applications and coordinates sensors modules and the Designer monitor. The Designer also allows for presenting and exporting statistical data.

4.10. Demonstrate Actions

Demonstrate actions [8,40] refers to approaches to specifying actions by only showing the system what to do through an interface, e.g., programming by example/demonstration/rehearsal/teaching. The development tool includes functionalities to "record" a demonstration (e.g., sequence of events or actions), and then the application will perform accordingly.

Dey et al. introduced a CAPpella [41], programming by demonstration approach for building context-aware applications. The approach enables end-users to specify features by annotating relevant demonstration portions to program desired behaviors. A recording system captures data from sensors (e.g., video, microphone, RFID, phone in use) and actions from actuators (e.g., login/out, send e-mail, turn on/off a light). The recorded data are presented in an UI as a stream of information, including three parts: a video/audio player, events detected, and actions performed during the recorded session. A user can annotate the data in the UI, setting a start and end time of events, representing desired behaviors. The system then uses the annotations (of repeated recordings) as a machine learning system's training data. Then, the system recognizes "programmed" situations with live data and performs the specified actions.

Li et al. introduced the Topiary tool [46] to prototype applications based on a location of entities (i.e., people, places, or things), e.g., a location-based tour guide. The tool includes a visual language to represent places of interest, linking people to the places. Topiary can show status (i.e., display location), find (i.e., recognize specific or nearest entity), active maps (i.e., update entity locations), trigger action (i.e., activate when in/near) and wayfinding (i.e., reach location). Topiary's scope is twofold: prototype scenarios using a storyboarding interaction for representing location contexts, and running the storyboards on mobile devices, testing the prototype by updating locations. The scenarios are captured using programming by demonstration with a resizable recording window covering the target screen. The demonstrations allow dragging entities to define spatial relationships (i.e., in, out, enters, exits), a definition of proximity regions (i.e., near), and inclusion and exclusion of entities. After being defined, the tool allows for generalizing scenarios (e.g., replacing "a person" with "any person") and editing scenarios through visual functionalities. Finally, the tool facilitates linking scenarios to create storyboards with explicit links (i.e., end-users followed actions) and implicit links (i.e., automatic actions defined in scenarios).

Lee et al. introduced GALLAG Strip [61], a mobile tool for building context-aware applications based on real-world tangibility—an approach that follows the programming by demonstration development style. In this approach, end-users interact with sensors and objects to create an application through physical demonstration rather than software models and representations. A server maintains a registry of sensors, and the tool presents available sensors for building applications using a comic strip metaphor. A user starts a recording mode to create a new application, and the tool waits for user actions that trigger events. The demonstration screen shows it as an action frame when existing an event (physically triggered, such as turning on the TV). The user stops the recording mode after the required events, and the tool changes to an edit mode. In edit mode, the user reviews the sequence, and the tool allows for adding response frames (e.g., an alarm message). The tool then allows starting the recording mode to add events (e.g., turn off TV) and complete the application. The tool also allows for starting the recording mode again and completing the application with additional events (e.g., turn off TV + turn off speakers). Finally, end-users can edit the final application sequence (i.e., application strip), adding time-date frames for constraining executions. The tool incorporates visual support for programming such as colors (blue for actions, orange for responses, and green for time-date frames), icons (default and taken by the mobile's camera), and scroll functions. The programming model follows a linear condition (if-then) read from left to right and top to bottom.

Li et al. introduced EPIDOSITE [74], a mobile tool to create apps by demonstration. The tool integrates IoT devices (smart home) and infrared blasters and context accessed by smartphone apps. Smartphone widgets allow for recording and editing actions (i.e., a sequence of smartphone iterations). The tool allows for generalizing the demonstration to create new apps.

Fischer et al. introduced VASH [76], a web tool to demonstrate applications through web browsing actions and voice commands. The tool records web elements and user interaction (mouse, keyboard events). Then, the tool converts the interaction to executable operations based on an internal language. Finally, the tool uses the language to generalize the demonstration.

4.11. *Augmented/Advanced Interaction*

Specification of actions through an augmented/advanced interaction [40] refers to approaches applying other UI interactions such as virtual/augmented reality, physical/tangible proximity/interfaces [87], and voice.

Blackwell et al. introduced AutoHAN [43], an architecture that includes the Media Cubes programming language to program home appliances accessible to users that can operate a remote control. The language allows direct manipulation of cubes (physical blocks), arguing that it is more important than pictorial metaphors for end-user programming. Each cube has a single button, a single LED (status), and communication components for the home network, appliances, and cubes, enabling pairing cube and appliances. The home network integrates devices in the home living space (e.g., TV), outside it (e.g., cable TV decoder in the attic), and virtual devices (e.g., software components). The language allows a direct behavior associated with a concrete representation, including action effects, e.g., a cube "play/pause function" associated with a CD player and a VCR. A program then follows a composition of the abstractions represented in the media cubes, automating home processes or enhancing situations. The AutoHAN language's potential lies in the 2D adjacency of cubes rather than a 1D arrangement of textual programming languages.

Vianello et al. introduced T4Tags 2.0 [70], a toolkit for end-user programming of tangible tokens attached to home objects/appliances. The tokens, which embed sensing technologies, are used to create smart behaviors (called recipes). The behaviors are configured based on a trigger-condition-action syntax in a mobile-oriented web application. A trigger is an event that activates a recipe, such as a physical action/state or a date/time event. Conditions are constraints for triggered events using and/or operators. An action is an activity to perform, e.g., send, share, buy. It contains an action object, e.g., e-mail, file, item for the actions send, share, and buy, respectively. Actions also involve an action provider, which is the action's service, e.g., e-mail provider, Twitter, Facebook, and Amazon. The application shows available options regarding the selected action, e.g., for "send e-mail," it presents fields to introduce "to," "subject," and "message".

Wang et al. presented CAPturAR [78], an augmented reality authoring tool for IoT context-aware applications. It includes a visual programming interface to create rules, defining human actions corresponding to daily activities. The interface records scenarios and then presents virtual replicas of objects and a human avatar. A user can demonstrate actions supported by UI functionalities for timeline navigation object drawing (e.g., connecting object functions and desire actions). Then, the tool generates detection models with motion data, which the user can also edit. Users can add or edit necessary contextual information such as time, location, objects, state (e.g., light on/off). Finally, users can test the functions on a simulated IoT.

5. What Is behind the Tool? Implementation Approaches and Challenges

5.1. Rule-Based Programming

Rule-based is the most common method to specify context-aware applications. The basic rule expression form is the "IF condition Then action" adopted by iCap [44], Context Cloud [60], GALLAG Strip [61], and VS-CaSP [64]. Martín et al. [68] improve the implementation of Context Cloud, incorporating a rule engine to infer and iBeacons and algorithms to provide proximity functionalities. Situation API [50] enables decomposing context rules into references, parameters, and listeners. A reference abstracts input context (e.g., location, light level). The parameters are relevant parameters processed by references (e.g., bedroom light is on). The listeners receive notification of occurrences such as invoked actions or modified parameters (e.g., turn off the lights).

Events complement rule-based expressions, such as T4Tags 2.0 [70], which define trigger–condition–action (WHEN) rules. TARE [73] extends the rule expression to include IF-THEN, WHEN, AND, OR. Similarly, IVO [57] activity workflows model events. The architecture includes a rule-based system and an event-driven workflow engine. The rule system allows the definition of alternative actions (i.e., IF-THEN-ELSE), considering standard operations (e.g., greater than, equals, not, AND, OR). The workflow engine is implemented as a finite state machine (FSM) to execute workflows in mobile clients. A mobile client also includes the perceived environment (sensing) and process conditions (register/unregister context values in the workflow engine). The application definitions are expressed in IVOML—an XML-based language to develop mobile context-aware applications [88].

SSB [62] allows for context registration and subscription of HNS (Home Network System) services. End-users can search and aggregate context, as well as verify, reuse, and refine existing services. The context registration allows defining conditions (logical expressions) for sensor services. Afterward, the context subscription allows binding registered contexts to appliance operations. In the end, end-users can create service rules for the HNS.

5.2. Using Ontologies for Programming

Ontologies further complement rule-based expressions. Context Studio [47] uses an ontology [89] to define the trigger context and actions. The ontology and rule-based model manage the context [90], enabling a hierarchical representation to customize applications. Preuveneers et al. [56] adopt an ontology for modeling environments. The ontology includes a model of people, service devices, sensor devices, and location. The framework integrates context from devices and infers high-level context for the composition of applications. In the Rodríguez et al. [59] approach, an application represents semantic IF-THEN rules. Subgraphs (i.e., graph inside IF or THEN clauses) can be connected using logical operators. Subgraphs in THEN clauses can also include loops.

iPlumber [52] adopts a domain ontology (e.g., an ontology for smart homes). An application template consists of rules (action templates) that consider the context (hardware and software input/outputs). The context is integrated using context wrappers and maintained into an individualized ontology server, extended from the domain ontology. The individualized ontology provides an abstract interface for applications, relating rules, and facilitating context access and reasoning. OPEN [54] (iPlumber following proposal) introduced three levels of ontologies to support developer/end-user cooperation: (1) domain ontology, e.g., an ontology for smart spaces, (2) upper ontology abstracting high-level concepts of context spaces, e.g., place, sensor, and (3) individualized ontology for applications.

5.3. Using a Reduced Language or Pattern for Programming

The rule-based systems are combined with simplified languages to increase expressiveness. Atomate [53] UI is based on the Controlled Natural Language Interface, i.e., UI with simplified grammars to reduce ambiguity/complexity. Furthermore, the tool includes a feeder component to retrieve a list of all data sources and polls periodically. At startup, it registers callbacks for data sources that provide asynchronous call-backs to push data. A rule manager notices feeder updates and starts a brute-force evaluation of rules, checking antecedent clauses (trigger) against the current world model state.

Similarly, CAMP [45] includes a domain-specific vocabulary subset of natural language. The tool also includes a custom dictionary to reword and decompose statements, applying diverse mechanisms for restructuring to standard phrases, e.g., "beginning at 3:00 p.m. for 2 h" becomes "between 3:00 p.m. and 5:00 p.m." CoRE [72] includes a constrained natural language limited in vocabulary and grammar. Complementary to the basic logic operation (and, or, not), it includes 'unless' to express exception, negations over disjunctions (neither a nor b), and an association of logical expressions.

S-APIs [63] are implemented by software developers using semantic components and a language for composition compatibility. S-APIs are supported by an interoperability platform to describe and manage cross-smart spaces, e.g., information interchange between a smartphone and a building. S-APIs provide commands to perform functionalities (e.g., PublishLocation) and event-based commands (e.g., IdentifyLocation). A command involves inputs, outputs, and execution, branches (e.g., display_failure, information_displayed for the command DisplayInformation).

SPOK [65] integrates context (i.e., sensor/actuators and Web services) through a middleware developed by software experts. The middleware manages the context and connects to SPOK, supporting an interchange between the context and applications. The middleware exposes its API through a simplified language (i.e., a pseudo-natural concrete syntax). End-users define applications, i.e., abstract syntax tree following the middleware API. Applications contain a set of ECA rules interpreted in parallel, including WHEN (events), IF-THEN, and WHILE statements. Thus, the middleware can interpret applications and provide the state of the smart home to clients, including triggered events.

Dobby [69] programming by selection (PBS) follows seven design patterns for context-aware services. The proposed patterns extend the condition → action pattern to action services interchanging messages, sending multimedia streams, or synchronizing services. The Dobby system support service creation and execution through a middleware (service platform) and a development environment. The middleware enables registering service objects (i.e., device/software as service) in a service space, checking rules, and executing actions. The development environment (UI) shows service objects as blocks for selection to build applications.

5.4. Language-Based Programming

Advanced languages enable an augmented and specific level of expressiveness. AutoHAN [43] programming language Media Cubes include ontological and linguistic abstraction. Ontological abstractions are tokens that represent "natural categories" such as remote-control operations, appliance functions, network capabilities, and user skills. Linguistic abstractions are physical situations represented as linguistic terms. The physical cubes are made of wood with sensors and transducers, microprocessors, and batteries. VASH [76] includes a grammar to define variables and create control constructs: functions calls, conditionals, and iterations. End-users define (in the UI) a sequence of events, which are then translated to the internal programming language.

Similarly, EasyContext [77] defines a Backus-Naur form grammar to create rules, integrating contextual information using the Google Awareness API. A Web UI presents conditions and actions as an abstraction of the Awareness API features. Then, the rules are exported to mobile devices.

VisualRDK [48] integrates context as an abstract component to manage the heterogeneity of environments. Applications contain context and components connected with a set of events and commands. The events allow the starting behavior of an application for subsequent processing. Commands are pre-defined instructions for managing context, e.g., set, find, and for processing, e.g., wait, emit (send out a signal), split (execute threads in parallel), and call (invoking functions).

MicroApps [55] visual language manages the UI composition process. The application programming is based on a data-flow Direct Acyclic Graph of services using a sequential, fork, and join compositions. The tool includes a running engine that interprets applications, generating a UI. In the running UI, each action is presented, waiting for user inputs when necessary (e.g., press button "Take" for the action camera.take). Subsequently, the authors presented a methodology to enable end-user development and an improved MicroApps application environment [91]. The application environment includes a repository (MicroAppStore) to share MicroApps and compose new applications based on existing ones. Similarly, MNFL [58] visual language manages the UI composition process. The expressiveness is given mainly by the blocks for computing (e.g., opposite, combine, math).

Puzzle [66] framework includes a jigsaw language that guides the sequential composition of an application. The framework is web-based, comprising a front-end and a mobile client. The front-end links an application repository and a piece repository. The piece repository stores standard pieces (e.g., quotes jigsaw piece), pieces representing external Web Services (e.g., post on Facebook), and interactive physical objects (e.g., turn off the lights). The mobile client includes an HTML viewer to build/edit/run applications and a native module to access the mobile functionalities (e.g., send a text message).

Mayer et al. [67] domain-specific model/language for smart things defines a taxonomy and hierarchy for abstracting sensing and actuation primitives. The *taxonomy* includes abstraction for sensors (i.e., get data, get value, get proportion), stateless actuators (i.e., trigger, goto), and stateful actuators (i.e., set, set value, level, set intensity, switch mode, switch, position, move). The *hierarchy* involves an organization for interactions. Get data, trigger, and set are the root of the hierarchy. Similarly, SITE [71] includes a domain-specific language representing a smart object (SO) context. The language, i.e., Simple Control Language (SCL), facilitates the specification of SOs control rules through context-free grammar. SOs represents transducers for objects, and the system supports a network for connecting the transducers. A user builds and deploys SOs setting a smart environment. Afterward, SITE enables the manipulation of SOs, registering a name and IP address, defining control rules, and visualizing sensor information. The control rules follow three modes: form-based, editor-based, and advanced for using an SCL graphical editor, SCL text editor, and fuzzy logic (without SCL), respectively. Then, a Fuzzy Generator interprets SCL expressions to generate fuzzy logic elements (i.e., membership functions and rules).

5.5. Artificial Intelligence and Other Approaches

Reviewed approaches also applied other methods for implementation. For instance, a CAPpella [41] involves annotations introduced by end-users to train a machine learning engine based on a Dynamic Bayesian Network (DBN). It allows for hiding the creation of behavior models and simplifying the training and testing of the DBN. GrOWTH [75] defines goals with an ontology to describe the set of actions required to achieve the goals. Then, apply AI planning to achieve the desired goal.

Topiary [46] implement algorithms for spatial relations (e.g., out, enters) and proximity (e.g., near). Similarly, ActivityDesigner [49] tool's model extends the activity theory (Leontiev, 1978 [92]), where the activity is decomposed hierarchically into actions and operations. The model adds the Scene as an action with a situation, e.g., for the action running—situation 1: with friends in a gym, and situation 2: alone in a park. The model also includes Themes inside a Scene. Themes are everyday aspects of people, e.g., eating healthy, having fun, and staying physically active. The model is the basis for the Activity Query Language (AQL), which facilitates activity specification of time, duration, confidence, and occurrences of interest using a list of filters. The tool allows scripting to implement enhanced behaviors. The implementation includes a part-of-speech tagging tool to analyze the scene's textual data and an XML-based query engine (for AQL) to support defining rules.

Persona [51] customization depends on a small cache of past preference. The preference changes (usage events) use a finite state machine implementation. Among others, the authors consider preferences for an artifact (e.g., use wall-mounted display rather than other display), action (e.g., enable/disable displaying weather information), interaction modality (e.g., use handwriting/voice for input, display/sound for output), and contextual events such as location, time, external (e.g., select the silent profile in a meeting room).

CAPturAR [78] processes augmented reality (AR) data from an AR platform with multiple cameras. It applies diverse machine learning algorithms to detect human activity and infer body positions. The authors also applied video and image processing algorithms to detect and infer real object interactions such as cup, kettle, can, and proximity to appliances (e.g., lamps).

5.6. Discussion and Challenges

The challenges for building a new application are variable and include challenges of the application domain (e.g., integrate a health device context or service). A primary aspect is providing more functionalities in a domain, managing different inputs, and data format. Adaptation and different ways to act (required actions) are also important. Therefore, EUD tools must consider user profiles and preferences to keep an enriched context for context awareness. Furthermore, EUD tools must include traditional methods to manage applications, e.g., update/upgrade, share, reuse, and deploy. In addition, they must include methods to control applications, e.g., monitor resources/situations, audit performances, failures, etc. A key challenge for building, manage and controlling applications is to provide context-aware EUD without requiring advanced computer technical skills (i.e., non-technical users).

The first category of EUD adopted a rule-based application development, with 22 out of 37 papers (Figure 3). They adopt ontologies, algorithms, and simplified languages to increase the expressiveness for creating rules. Language-based is the next category of EUD implementation method with ten out of 37 papers. These papers define the programming constructs adopting ontologies, vocabularies, or defining their language model (e.g., commands, composition logic). Other papers adopted other implementation approaches such as machine learning (two out of 37), AI planning (one out of 37), location algorithms (two out of 37), and finite state machine (two out of 37). In the following, we present the overall challenges obtained after our review.

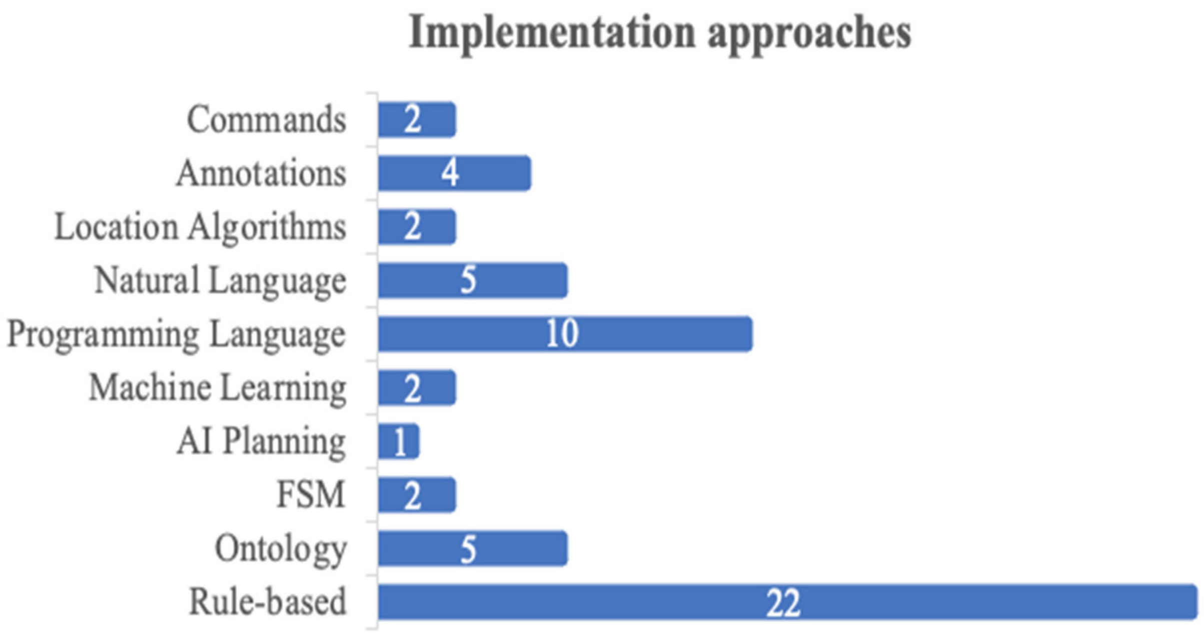

Figure 3. Implementation approaches.

5.6.1. Context-Aware End-User Development

The presented approaches facilitate the specification of context-aware applications, which involves (1) integration of required context, e.g., reading from sensors, (2) how to evaluate context values, e.g., two similar contexts, check event, and (3) definition of desired actions (i.e., reaction, proaction) that the system must perform when evaluating matches. Furthermore, rule-based, natural language, programming by demonstration, and trigger-action (based on events) are EUD approaches suitable for the IoT [11]. However, the reviewed approaches reveal challenges to target non-technical users. For example, using a model for tailoring applications requires end-users to learn the model's components. Combining metaphors for improved UI composition [80] and introducing artificial intelligence and machine learning is complicated for non-technical users. Likewise, we found technicalities for programming with an augmented UI interaction, e.g., tangible, wearable, and embodied interaction. This aspect is relevant because the final product (end-user program) uses things through a tangible/embodied [87] or wearable [93] interaction. Non-technical users should learn technical aspects such as linking smart tags and wearables (e.g., rings [94]).

Mobile end-user tools use available device resources that users can access for building applications (e.g., available services and actions, mobile sensors). We found eight out of 37 projects that enable EUD on mobile devices. From those projects, seven out of eight also run on mobile devices. In addition to those seven projects, three out of 37 projects run on a mobile device, but the EUD is on a PC/Web. However, in mobile EUD, it is challenging to include functionalities and improve the expressiveness because of the limitation in resources and screen dimensions. For example, eliciting space/geographical relations and enabling complex composition blocks (e.g., computing math) is still performed in a PC.

5.6.2. Specification of Complex Situations

End-user development proposes empowering non-software experts with building tools, adopting metaphors, and programming methods that increase usability. Most of the analyzed tools limit their expressivity to rules or simple languages/artifacts, simplifying the system to increase usability. For example, a sequential flow programming drawback is the restriction to build complex applications. Similarly, the illustration of rules improves usability but constrains the development of complex applications either by requiring knowing advanced elements (e.g., activity language for scripting), or by pre-defined elements for composing rules. For example, Dey et al. [95] presented the iCAP tool, including 371 building rules. The authors defined the rules based on a study of 20 subjects and their activities. With this study, the authors defined the categories of input/action context. They concluded the need to implement a sophisticated Boolean logic in the iCAP tool and manage dynamic rules to infer activities with ambiguous context.

Furthermore, to achieve the ubiquity characteristic of pervasive systems, it is necessary to facilitate the specification of complex situations, e.g., build applications to maintain conditions in diverse environments. A complex situation requires advanced mechanisms (i.e., mathematical calculations, algorithms), contrary to simpler input/action rules. An approach to deal with complex situations is to target the EUD tool for professional developers [48]. Likewise, augmenting the expressiveness of the context-aware platform can expand the situations that end-users can program. In this sense, one approach considers an enriched context to create a semantic layer with which semantic operations enable expressing/interpreting diverse situations [19,96,97]. However, it is still challenging for EUD tools to keep usability, targeting non-technical users while expanding the expressiveness for complex situations.

5.6.3. End-User Development in Industry

Adopting available development artifacts such as APIs and tools enhances end-user development due to the pre-existing functionalities. We found research efforts adopting available artifacts, combining them to promote end-users as active actors in software adaptation to build applications tailored to their needs. For example, EasyContext [77] abstracts the Google Awareness API [85] to create rules for Android apps. SITE [71] uses the LabVIEW [84] visual programming environment to create fuzzy rules for smart homes.

Industry tools also promote end-user development, empowering industry operators. They rely on the tools which provide a versatile Human Machine Interface (HMI) to monitor and control machines and processes. Industry operators customize UIs with high-quality graphics (2D, 3D) for representations of industry machine and processes, alarms, real-time trends, simulation, messaging (e.g., for errors), scripting language, and machine integration (i.e., access to the industrial network and machines) [98,99]. For example, Normanyo et al. [99] adopted Siemens Simatic WinCC to design an automated boiler plant. The authors described the industrial HMI design covering UI factors and risk factors such as (a) having a realistic view of an industrial plant, (b) reducing hardware (by replacing many push buttons and selector switches, and lights), and (c) replacing humans in tasks done in dangerous environments.

Similarly, Industry 4.0/5.0 is introducing additional interface capabilities. The new HMI generation for the industry is enhanced with an intuitive/autonomic UI and process prediction, adopting advances in technology such as virtual/augmented reality, collaborative robots, and artificial intelligence/machine learning [100,101]. End-user development and meta-design (i.e., design for designers) will play an important role in empowering industrial operators to capitalize on new technologies [102].

5.6.4. Development Methodology and End-User Software Engineering (EUSE)

End-user development (EUD) subsumes end-user programming (EUP) and target programming tasks with easy-to-use and easy-to-understand tools [8] for end-users. Complementary, end-user software engineering (EUSE) also focuses on the development methodology. EUSE involves systematic and disciplined activities that address software quality issues to improve application development for non-technical users. These activities are indistinguishable for the users and, consequently, secondary to the software's goal [7].

Adopting EUSE for context-aware application development is also a challenge. We found two out of 37 papers describing a development methodology. Context Cloud [60] methodology involves analysis (i.e., identify and parameterize situations), configuration (i.e., configure/implement context data model and providers), development (i.e., implement behavior), validation, and maintenance. MicroApps [91] methodology involves an incremental and iterative development process through service management of MicroApps components, e.g., finding compatible services. However, providing a EUSE methodology is not enough: the functional correctness and software quality still lies on the end-user and how the tools can support the end-user development.

6. Conclusions

End-user programming (EUP), end-user development (EUD), and end-user software engineering (EUSE) have been adopted to facilitate context-aware application development. This paper presented the result of our literature review of context-aware application development tools applying EUP, EUD, and EUSE. We selected 37 papers (out of 968 documents found) for the review to analyze the development technique (i.e., programming interaction and composition metaphor) and implementation approaches (i.e., what is behind the tools?).

The tools target non-technical and technical users, supporting them in the building process. The tools include techniques to facilitate the integration of context and application development, specifying conditions, actions, activities, and sharing resources in environments. We found 11 techniques, including the abstraction of traditional programming artifacts (e.g., APIs), visual programming (e.g., composing UI blocks), programming by demonstration (e.g., recording interactions), and augmented interactions (e.g., using tangible tokens).

Our review found that the most common implementation approach behind the tools is rule-based programming (e.g., selecting conditions and required actions in the UI will create the target rule). Ontologies and simplified natural language are used to increase the expressiveness of rules. Language-based programming is also applied to manage the diversity of elements for building applications (e.g., heterogeneous devices, context types, visual block types). Other mechanisms include finite state machines, algorithms, and machine learning for advanced management of the context.

Even though end-users can use well-known language constructs (e.g., if, when, then, and) to build applications, empowering end-users also implies simplifying constructs using abstractions (e.g., icons, blocks, commands, model). Thereby, EUD is a promising approach to achieve end-user intentions, and EUD implementations are also evolving to mobile devices. However, the end-user development of complex context-aware applications is still challenging while applying advanced algorithms to improve reasoning and learning.

Author Contributions: Conceptualization, V.P. and B.A.; methodology, V.P. and B.A.; software, V.P.; validation, V.P. and B.A.; formal analysis, V.P.; investigation, V.P.; resources, V.P.; data curation, V.P.; writing—original draft preparation, V.P.; writing—review and editing, V.P. and B.A.; visualization, V.P.; supervision, B.A.; project administration, B.A.; funding acquisition, B.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by SENESCYT, Ecuador and Université de Sherbrooke, Canada.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Perera, C.; Zaslavsky, A. Context aware computing for the internet of things: A survey. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 414–454. [CrossRef]
2. Saha, D.; Mukherjee, A. Pervasive computing: A paradigm for the 21st century. *IEEE Comput.* **2003**, *36*, 25–31. [CrossRef]
3. Satyanarayanan, M. Pervasive Computing: Vision and Challenges. *Pers. Commun. IEEE* **2001**, *8*, 10–17. [CrossRef]
4. Burnett, M.M.; Myers, B. A Future of End-user Software Engineering: Beyond the Silos. In Proceedings of the on Future of Software Engineering, Hyderabad, India, 31 May–7 June 2014; ACM: New York, NY, USA, 2014; Volume 2014, pp. 201–211.
5. Burnett, M.M.; Scaffidi, C. End-user development. In *The Encyclopedia of Human-Computer Interaction*, 2nd ed.; The Interaction Design Foundation: Aarhus, Denmark, 2012.
6. Paternò, F. End User Development: Survey of an Emerging Field for Empowering People. *ISRN Softw. Eng.* **2013**, *2013*, 532659. [CrossRef]
7. Ko, A.J.; Myers, B.; Rosson, M.B.; Rothermel, G.; Shaw, M.; Wiedenbeck, S.; Abraham, R.; Beckwith, L.; Blackwell, A.; Burnett, M.; et al. The State of the Art in End-user Software Engineering. *ACM Comput. Surv.* **2011**, *43*, 1–44. [CrossRef]
8. Lieberman, H.; Paternò, F.; Klann, M.; Wulf, V. End-User Development: An Emerging Paradigm. In *End User Development*; Lieberman, H., Paternò, F., Wulf, V., Eds.; Springer: Dordrecht, The Netherlands, 2006; pp. 1–8.
9. Barricelli, B.R.; Cassano, F.; Fogli, D.; Piccinno, A. End-user development, end-user programming and end-user software engineering: A systematic mapping study. *J. Syst. Softw.* **2019**, *149*, 101–137. [CrossRef]
10. Tetteroo, D.; Markopoulos, P. A review of research methods in end user development. In Proceedings of the International Symposium on End User Development, Madrid, Spain, 26–29 May 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 58–75.
11. Paternò, F.; Santoro, C. A design space for end user development in the time of the internet of things. In *New Perspectives in End-User Development*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 43–59.
12. Fischer, G.; Giaccardi, E.; Ye, Y.; Sutcliffe, A.G.; Mehandjiev, N. Meta-design: A manifesto for end-user development. *Commun. ACM* **2004**, *47*, 33–37. [CrossRef]
13. Ardito, C.; Costabile, M.F.; Desolda, G.; Matera, M. A Three-Layer Meta-Design Model for Addressing Domain-Specific Customizations. In *New Perspectives in End-User Development*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 99–120.
14. Burnett, M.; Cook, C.; Rothermel, G. End-user software engineering. *Commun. ACM* **2004**, *47*, 53–58. [CrossRef]
15. Kulesza, T.; Burnett, M.; Stumpf, S.; Wong, W.; Das, S.; Groce, A.; Shinsel, A.; Bice, F.; Mcintosh, K. Where are my intelligent assistant's mistakes? A systematic testing approach. In Proceedings of the International Symposium on End User Development, Torre Canne, Italy, 7–10 June 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 171–186.
16. Roy, P. *ContextAA: Plateforme Sensible au Contexte Pour Aborder le Problème de L'espace Intelligent Ouvert*; Université de Shrerbrooke: Sherbrooke, QC, Canada, 2019.
17. Pham, Q.V.; Fang, F.; Ha, V.N.; Piran, M.J.; Le, M.; Le, L.B.; Hwang, W.J.; Ding, Z. A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art. *IEEE Access* **2020**, *8*, 116974–117017. [CrossRef]
18. Dey, A.K. Understanding and Using Context. *Pers. Ubiquitous Comput.* **2001**, *5*, 4–7. [CrossRef]

19. Ponce, V.; Roy, P.; Abdulrazak, B. Dynamic domain model for micro context-aware adaptation of applications. In Proceedings of the 13th IEEE International Conference on Ubiquitous Intelligence and Computing, Toulouse, France, 18–21 July 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 98–105.
20. Abdulrazak, B.; Roy, P.; Gouin-Vallerand, C.; Giroux, S.; Belala, Y. Macro and micro context-awareness for autonomic pervasive computing. In Proceedings of the 12th International Conference on Information Integration and Web-Based Applications & Services, Paris, France, 8–10 November 2010. [CrossRef]
21. Schilit, B.; Theimer, M. Disseminating Active Map Information to Mobile Hosts. *Netw. IEEE* **1994**, *8*, 22–32. [CrossRef]
22. Abowd, G.D.; Dey, A.K.; Brown, P.J.; Davies, N.; Smith, M.; Steggles, P. Towards a better understanding of context and context-awareness. In *International Symposium on Handheld and Ubiquitous Computing*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 304–307.
23. Brønsted, J.; Hansen, K.M.; Ingstrup, M. Service Composition Issues in Pervasive Computing. *IEEE Pervasive Comput.* **2010**, *9*, 62–70. [CrossRef]
24. Holloway, S.; Julien, C. The case for end-user programming of ubiquitous computing environments. In Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research, FoSER 2010, Santa Fe, NM, USA, 7–8 November 2010; ACM: New York, NY, USA, 2010; pp. 167–171.
25. Zhang, D.; Adipat, B.; Mowafi, Y. User-centered context-aware mobile applications-the next generation of personal mobile computing. *Commun. Assoc. Inf. Syst.* **2009**, *24*, 27–46. [CrossRef]
26. Butz, A. User Interfaces and HCI for Ambient Intelligence and Smart Environments. In *Handbook of Ambient Intelligence and Smart Environments*; Nakashima, H., Aghajan, H., Augusto, J.C., Eds.; Springer: New York, NY, USA, 2010; pp. 535–558, ISBN1 978-0-387-93807-3. (Print); ISBN2 978-0-387-93808-0. (Online).
27. Ponce, V.; Deschamps, J.P.; Giroux, L.P.; Salehi, F.; Abdulrazak, B. QueFaire: Context-Aware in-person social activity recommendation system for active aging. In Proceedings of the Inclusive Smart Cities and e-Health, ICOST 2015, Geneva, Switzerland, 10–12 June 2015; Springer: Cham, Switzerland, 2015; pp. 64–75.
28. Apache Cordova. Available online: https://cordova.apache.org (accessed on 10 December 2021).
29. Wolber, D.; Abelson, H.; Friedman, M. Democratizing Computing with App Inventor. *GetMobile Mob. Comput. Commun.* **2015**, *18*, 53–58. [CrossRef]
30. Modkit. Available online: https://www.modkit.com (accessed on 10 December 2021).
31. Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Millner, A.; Rosenbaum, E.; Silver, J.; Silverman, B.; et al. Scratch: Programming for All. *Commun. ACM* **2009**, *52*, 60–67. [CrossRef]
32. Maloney, J.; Resnick, M.; Rusk, N. The Scratch programming language and environment. *ACM Trans. Comput. Educ.* **2010**, *10*, 1–15. [CrossRef]
33. Arduino. Available online: https://www.arduino.cc (accessed on 10 December 2021).
34. Slany, W. A mobile visual programming system for Android smartphones and tablets. In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, Innsbruck, Austria, 30 September–4 October 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 265–266.
35. Ur, B.; McManus, E.; Pak Yong Ho, M.; Littman, M.L. Practical trigger-action programming in the smart home. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, New York, NY, USA, 26 April–1 May 2014; ACM: New York, NY, USA, 2014; pp. 803–812.
36. Lucci, G.; Paterno, F.; Paternò, F. Understanding End-User Development of Context-Dependent Applications in Smartphones. In Proceedings of the Human-Centred Software Engineering, HCSE 2014, Paderborn, Germany, 16–18 September 2014; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8742, pp. 182–198.
37. Amazon Alexa. Available online: https://developer.amazon.com/en-US/alexa (accessed on 10 December 2021).
38. SmartThings. Available online: https://www.smartthings.com (accessed on 10 December 2021).
39. Workflow for iOS. Available online: https://my.workflow.is (accessed on 10 December 2021).
40. Kelleher, C.; Pausch, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmer. *ACM Comput. Surv.* **2005**, *37*, 83–137. [CrossRef]
41. Dey, A.K.; Hamid, R.; Beckmann, C.; Li, I.; Hsu, D. A CAPpella: Programming by demonstration of context-aware applications. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Vienna, Austria, 24–29 April 2004; ACM: New York, NY, USA, 2004; Volume 6, pp. 33–40.
42. Trullemans, S.; Van Holsbeeke, L.; Signer, B. The Context Modelling Toolkit: A Unified Multi-layered Context Modelling Approach. In *Proceedings of the ACM on Human-Computer Interaction*; ACM: New York, NY, USA, 2017; Volume 1, pp. 1–16.
43. Blackwell, A.F.; Hague, R. AutoHAN: An architecture for programming the home. In Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments, Stresa, Italy, 5–7 September 2001; IEEE: Piscataway, NJ, USA, 2001; pp. 150–157.
44. Sohn, T.; Sohn, T.; Dey, A.; Dey, A. iCAP: An informal tool for interactive prototyping of context-aware applications. In Proceedings of the CHI'03 Extended Abstracts on Human Factors in Computing Systems, Fort Lauderdale, FL, USA, 5–10 April 2003; ACM: New York, NY, USA, 2003; pp. 974–975.

45. Truong, K.N.; Huang, E.M.; Abowd, G.D. CAMP: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home. In Proceedings of the UbiComp 2004: Ubiquitous Computing, Nottingham, UK, 7–10 September 2004; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3205, pp. 143–160.
46. Li, Y.; Hong, J.; Landay, J. Topiary: A tool for prototyping location-enhanced applications. In Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology, Santa Fe, NM, USA, 24–27 October 2004; ACM: New York, NY, USA, 2004; Volume 6, pp. 217–226.
47. Korpipää, P.; Malm, E.J.; Rantakokko, T.; Kyllönen, V.; Kela, J.; Mäntyjärvi, J.; Häkkilä, J.; Känsälä, I. Customizing user interaction in smart phones. *IEEE Pervasive Comput.* **2006**, *5*, 82–90. [CrossRef]
48. Weis, T.; Knoll, M.; Ulbrich, A.; Mühl, G.; Brändie, A. Rapid prototyping for pervasive applications. *IEEE Pervasive Comput.* **2007**, *6*, 76–84. [CrossRef]
49. Li, Y.; Landay, J.A. Activity-based prototyping of ubicomp applications for long-lived, everyday human activities. In Proceedings of the ACM CHI 2008 Conference on Human Factors in Computing Systems, Florence, Italy, 5–10 April 2008; ACM: New York, NY, USA, 2008; Volume 1, pp. 1303–1312.
50. Dey, A.K.; Newberger, A. Support for Context-Aware Intelligibility and Control. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Boston, MA, USA, 4–9 April 2009; ACM: New York, NY, USA, 2009; pp. 859–868.
51. Kawsar, F.; Fujinami, K.; Nakajima, T.; Park, J.H.; Yeo, S.S. A portable toolkit for supporting end-user personalization and control in context-aware applications. *Multimed. Tools Appl.* **2010**, *47*, 409–432. [CrossRef]
52. Guo, B.; Zhang, D.; Imai, M. Enabling user-oriented management for ubiquitous computing: The meta-design approach. *Comput. Netw.* **2010**, *54*, 2840–2855. [CrossRef]
53. Van Kleek, M.; Moore, B.; Karger, D.; André, P.; Schraefel, M.C. Atomate it! end-user context-sensitive automation using heterogeneous information sources on the web. In Proceedings of the 19th International Conference on World Wide Web, Raleigh, NC, USA, 26–30 April 2010; ACM: New York, NY, USA, 2010; Volume 26, pp. 951–960.
54. Guo, B.; Zhang, D.; Imai, M. Toward a cooperative programming framework for context-aware applications. *Pers. Ubiquitous Comput.* **2011**, *15*, 221–233. [CrossRef]
55. Cuccurullo, S.; Francese, R.; Risi, M.; Tortora, G. MicroApps Development on Mobile Phones. In Proceedings of the End-User Development, IS-EUD 2011, Torre Canne, Italy, 7–10 June 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 289–294.
56. Preuveneers, D.; Berbers, Y. Intelligent widgets for intuitive interaction and coordination in smart home environments. In Proceedings of the 2012 Eighth International Conference on Intelligent Environments, Guanajuato, Mexico, 26–29 June 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 157–164.
57. Realinho, V.; Romão, T.; Dias, A.E. An event-driven workflow framework to develop context-aware mobile applications. In Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia—MUM '12, Ulm, Germany, 4–6 December 2012; ACM: New York, NY, USA, 2012; pp. 1–10.
58. Edgcomb, A.D.; Vahid, F. MNFL: The monitoring and notification flow language for assistive monitoring. In Proceedings of the 2nd ACM SIGHIT Symposium on International Health Informatics—IHI '12, Miami, FL, USA, 28–23 January 2012; ACM: New York, NY, USA, 2012; pp. 191–200.
59. Rodríguez, N.D.; Lilius, J.; Cuéllar, M.P.; Calvo-Flores, M.D. Extending semantic web tools for improving smart spaces interoperability and usability. In Proceedings of the Distributed Computing and Artificial Intelligence, Salamanca, Spain, 22–24 May 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 45–52.
60. Martín, D.; López-de-Ipiña, D.; Alzua-Sorzabal, A.; Lamsfus, C.L.; Torres-Manzanera, E. A methodology and a web platform for the collaborative development of context-aware systems. *Sensors* **2013**, *13*, 6032–6053. [CrossRef]
61. Lee, J.; Garduno, L.; Walker, E.; Burleson, W. A Tangible Programming Tool for Creation of Context-Aware Applications. In Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing, Zurich, Switzerland, 8–12 September 2013; ACM: New York, NY, USA, 2013; pp. 391–400.
62. Nakamura, M.; Matsuo, S.; Matsumoto, S. Supporting end-user development of context-aware services in home network system. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing 2012*; Springer: Berlin/Heidelberg, Germany, 2013; Volume 443, pp. 159–170. ISBN 9783642321719.
63. Palviainen, M.; Kuusijärvi, J.; Ovaska, E. A semi-automatic end-user programming approach for smart space application development. *Pervasive Mob. Comput.* **2014**, *12*, 17–36. [CrossRef]
64. Su, J.-M.; Huang, C.-F. An easy-to-use 3D visualization system for planning context-aware applications in smart buildings. *Comput. Stand. Interfaces* **2014**, *36*, 312–326. [CrossRef]
65. Coutaz, J.; Demeure, A.; Caffiau, S.; Crowley, J.L.; Demeure, A.; Crowley, J.L. Early lessons from the development of SPOK, an end-user development environment for smart homes. In Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing Adjunct Publication, Seattle, WA, USA, 13–17 September 2014; ACM: New York, NY, USA, 2014; pp. 895–902.
66. Danado, J.; Paterno, F.; Paternò, F. Puzzle: A mobile application development environment using a jigsaw metaphor. *J. Vis. Lang. Comput.* **2014**, *25*, 297–315. [CrossRef]
67. Mayer, S.; Tschofen, A.; Dey, A.K.; Mattern, F. User interfaces for smart things—A generative approach with semantic interaction descriptions. *ACM Trans. Comput. Interact.* **2014**, *21*, 1–25. [CrossRef]

68. Martín, D.; Lamsfus, C.; Alzua-Sorzabal, A. A cloud-based platform to develop context-aware mobile applications by domain experts. *Comput. Stand. Interfaces* **2016**, *44*, 177–184. [CrossRef]
69. Park, J.; Lee, K.H. Design patterns for context-aware services. *Multimed. Tools Appl.* **2015**, *74*, 2337–2358. [CrossRef]
70. Vianello, A.; Florack, Y.; Bellucci, A.; Jacucci, G. T4Tags 2.0: A Tangible System for Supporting Users' Needs in the Domestic Environment. In Proceedings of the TEI '16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction, Eindhoven, The Netherlands, 14–17 February 2016; ACM: New York, NY, USA, 2016; pp. 38–43.
71. Hafidh, B.; Al Osman, H.; Dong, H.; Arteaga-Falconi, J.; El Saddik, A. SITE: The Simple Internet of Things Enabler For Smart Homes. *IEEE Access* **2017**, *5*, 2034–2049. [CrossRef]
72. Metaxas, G.; Markopoulos, P. Natural contextual reasoning for end users. *ACM Trans. Comput. Interact.* **2017**, *24*, 1–36. [CrossRef]
73. Corcella, L.; Manca, M.; Paternò, F. Personalizing a student home behaviour. In *International Symposium on End User Development*; Springer: Cham, Switzerland, 2017; pp. 18–33.
74. Li, T.J.-J.; Li, Y.; Chen, F.; Myers, B.A. Programming IoT devices by demonstration using mobile apps. In *International Symposium on End User Development*; Springer: Cham, Switzerland, 2017; pp. 3–17.
75. Noura, M.; Heil, S.; Gaedke, M. GrOWTH: Goal-oriented end user development for web of things devices. In Proceedings of the Web Engineering—ICWE 2018, Cáceres, Spain, 5–8 June 2018; Springer: Cham, Switzerland, 2018; Volume 10845, pp. 358–365.
76. Fischer, M.H.; Campagna, G.; Choi, E.; Lam, M.S. Multi-Modal End-User Programming of Web-Based Virtual Assistant Skills. *arXiv* **2020**, arXiv:2008.13510.
77. Ponciano, T.; Tabosa, D.; Viana, W.; Duarte, P.; Carmo, R. A Generative Approach for Android Sensor-based Applications. In Proceedings of the Brazilian Symposium on Multimedia and the Web, São Luís, Brazil, 30 November–4 December 2020; ACM: New York, NY, USA, 2020; pp. 33–40.
78. Wang, T.; Qian, X.; He, F.; Hu, X.; Huo, K.; Cao, Y.; Ramani, K. CAPturAR: An augmented reality tool for authoring human-involved context-aware applications. In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology—UIST 2020, Virtual Event, USA, 20–23 October 2020; ACM: New York, NY, USA, 2020; pp. 328–341.
79. Realinho, V.; Dias, A.E.; Romão, T. Testing the usability of a platform for rapid development of mobile context-aware applications. In Proceedings of the Human-Computer Interaction—INTERACT 2011, Lisbon, Portugal, 5–9 September 2011; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6948, pp. 521–536.
80. Davidyuk, O.; Milara, I.S.; Gilman, E.; Riekki, J. An Overview of Interactive Application Composition Approaches. *Open Comput. Sci.* **2015**, *5*, 79–95. [CrossRef]
81. Cuccurullo, S.; Francese, R.; Risi, M.; Tortora, G. A visual approach supporting the development of MicroApps on mobile phones. In Proceedings of the 17th International Conference on Distributed Multimedia Systems, DMS 2011, Florence, Italy, 18–20 August 2011; Knowledge Systems Institute: Skokie, IL, USA, 2011; pp. 171–176.
82. Danado, J.; Paternò, F. Puzzle: A visual-based environment for end user development in touch-based mobile phones. In Proceedings of the Human-Centered Software Engineering, HCSE 2012, Toulouse, France, 29–31 October 2012; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7623, pp. 199–216.
83. Lee, J.; Burleson, W.; Walker, E.; Hekler, E.B.E.; Burleson, W.; Hekler, E.B.E.; Walker, E.; Hekler, E.B.E. Programming tool of context-aware applications for behavior change. In Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication, Seattle, WA, USA, 13–17 September 2014; ACM: New York, NY, USA, 2014; pp. 91–94.
84. LabVIEW. Available online: https://www.ni.com/en-ca/shop/labview.html (accessed on 10 December 2021).
85. Google Awareness API. Available online: https://developers.google.com/awareness (accessed on 10 December 2021).
86. Häkkilä, J.; Korpipää, P.; Ronkainen, S.; Tuomela, U.; Hakkila, J.; Korpipaa, P.; Ronkainen, S.; Tuomela, U. Interaction and end-user programming with a context-aware mobile application. In Proceedings of the Human-Computer Interaction-INTERACT 2005, Rome, Italy, 12–16 September 2005; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3585, pp. 927–937.
87. Tetteroo, D.; Soute, I.; Markopoulos, P. Five key challenges in end-user development for tangible and embodied interaction. In Proceedings of the 15th ACM on International Conference on Multimodal Interaction—ICMI '13, Sydney, Australia, 9–13 December 2013; ACM: New York, NY, USA, 2013; pp. 247–254.
88. Realinho, V.; Romão, T.; Dias, A.E. A language for the end-user development of mobile context-aware applications. *J. Wirel. Mob. Netw. Ubiquitous Comput. Dependable Appl.* **2020**, *11*, 54–80. [CrossRef]
89. Korpipää, P.; Häkkilä, J.; Kela, J.; Ronkainen, S.; Känsälä, I. Utilising context ontology in mobile device application personalisation. In Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia—MUM '04, College Park, MD, USA, 27–29 October 2004; ACM: New York, NY, USA, 2004; pp. 133–140.
90. Korpipää, P.; Malm, E.-J.J.E.E.-J.; Salminen, I.I.; Rantakokko, T.; Kyllönen, V.; Känsälä, I.I. Context management for end user development of context-aware applications. In Proceedings of the 6th International Conference on Mobile Data Management, MDM'05, Ayia Napa, Cyprus, 9–13 May 2015; ACM: New York, NY, USA, 2005; pp. 304–308.
91. Francese, R.; Risi, M.; Tortora, G.; Tucci, M. Visual Mobile Computing for Mobile End-Users. *IEEE Trans. Mob. Comput.* **2016**, *15*, 1033–1046. [CrossRef]
92. Leontyev, A.N. *Activity and Consciousness*; Progress Publishers: Moscow, Russia, 1977.

93. Rissanen, M.J.; Fernando, O.N.N.; Pang, N.; Iroshan, H.; Foo, S. Ubiquitous Shortcuts: Mnemonics by Just Taking Photos. In Proceedings of the CHI'13 Extended Abstracts on Human Factors in Computing Systems, Paris, France, 27 April–2 May 2013; ACM: New York, NY, USA, 2013; pp. 1641–1646.
94. Rissanen, M.J.; Vu, S.; Fernando, O.N.N.; Pang, N.; Foo, S. Subtle, Natural and Socially Acceptable Interaction Techniques for Ringterfaces—Finger-Ring Shaped User Interfaces. In Proceedings of the Distributed, Ambient, and Pervasive Interactions, DAPI 2013, Las Vegas, NV, USA, 21–26 July 2013; Springer: Berlin/Heidelberg, Germany, 2013; Volume 8028, pp. 52–61.
95. Dey, A.K.; Sohn, T.; Streng, S.; Kodama, J. iCAP: Interactive Prototyping of Context-Aware Applications. In *International Conference on Pervasive Computing, Dublin, Ireland, 7–10 May 2010*; Fishkin, K.P., Schiele, B., Nixon, P., Quigley, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 3968, pp. 254–271.
96. Roy, P.; Abdulrazak, B.; Belala, Y. Quantifying Semantic Proximity between Contexts. In Proceedings of the Smart Homes and Health Telematics, ICOST 2014, Denver, CO, USA, 25–27 June 2014; Springer: Cham, Switzerland, 2014; pp. 165–174.
97. Ponce, V.; Abdulrazak, B. Activity Model for Interactive Micro Context-Aware Well-Being Applications Based on ContextAA. In Proceedings of the Enhanced Quality of Life and Smart Living, ICOST 2017, Paris, France, 29–31 August 2017; Springer: Cham, Switzerland, 2017; pp. 99–111.
98. Qasim, I.; Anwar, M.W.; Azam, F.; Tufail, H.; Butt, W.H.; Zafar, M.N. A model-driven mobile HMI framework (MMHF) for industrial control systems. *IEEE Access* **2020**, *8*, 10827–10846. [CrossRef]
99. Normanyo, E.; Husinu, F.; Agyare, O.R. Developing a human machine interface (HMI) for industrial automated systems using siemens simatic WinCC flexible advanced software. *J. Emerg. Trends Comput. Inf. Sci.* **2014**, *5*, 134–144.
100. Papcun, P.; Kajáti, E.; Koziorek, J. Human machine interface in concept of industry 4.0. In Proceedings of the 2018 World Symposium on Digital Intelligence for Systems and Machines (DISA), Košice, Slovakia, 23–25 August 2018; pp. 289–296.
101. Longo, F.; Padovano, A.; Umbrello, S. Value-oriented and ethical technology engineering in industry 5.0: A human-centric perspective for the design of the factory of the future. *Appl. Sci.* **2020**, *10*, 4182. [CrossRef]
102. Fogli, D.; Piccinno, A. End-user development in industry 4.0: Challenges and opportunities. In Proceedings of the International Symposium on End User Development, Hatfield, UK, 10–12 July 2019; pp. 230–233.