A Brief History of Liquid Software

Cesare Pautasso
Software Institute – USI
Lugano, Switzerland
c.pautasso@ieee.org

***Abstract*—The concept of liquid software, i.e., software with flexible deployment, over the past two decades has appeared in the fields of edge computing, Internet of Things (IoT), Human-Computer Interaction, DevOps and Web engineering. In this paper, we survey, compare, and provide a comprehensive definition of liquid software by analyzing how the metaphor has been used in existing literature and identifying gaps and inconsistencies in the current vs. past understanding of the concept. Overall, liquid software can be seamlessly deployed and redeployed within a dynamic and distributed runtime environment in response to changes applied to the set of available devices and to the software itself. Liquid software has been introduced in the context of active networks and intelligent environments, it has been applied to describe the user interaction with multi and cross-device user interfaces, it has found a promising foundation in Web technology, continuous software delivery pipelines, as well as isomorphic software architectures running across the IoT, edge and Cloud continuum.**

***Index Terms*—Liquid Software, Software Deployment, Isomorphic Software Architecture, Continuum, Liquid User Experience, Multi-Device User Interface, Cross-Device User Interface**

I. Introduction

The liquid metaphor has been applied to software as it gets dynamically deployed and redeployed across multiple IoT, edge or endpoint devices [1]. It describes the seamless user experience of interacting with applications which can migrate between different devices following the user attention [2], [3] as well as take full advantage of all available devices [4], for example, in a collaboration scenario when multiple users are involved [5]. It refers to the ability of software to adapt to the capabilities of *a set* of devices [6], just like liquids adapt to the shape of their containers. It has also been recently used to compare the delivery of software updates with zero downtime to the "constant, unending flow of a river" [7].

In this paper we survey, compare and provide a comprehensive definition of liquid software. As many different authors over the past decades have interpreted and revisited the concept of liquid software in their own context – e.g., from edge computing to artificial intelligence, from cross-device user interfaces to task migration and offloading, from development frameworks and tools for continuous build and integration pipelines – the main contribution of this paper is a comprehensive definition of liquid software that synthesizes diverse interpretations based on a critical analysis of existing literature. This analysis helps in identifying the gaps and inconsistencies in the various interpretations of the liquid software metaphor and lays the groundwork for future research directions.

While countless publications have appeared in the past two decades proposing seamless solutions to the problem of flexible software deployment, we have selected to include only the ones which explicitly make use of the terms "liquid", "liquidity", or "fluid" as a metaphor to illustrate the behavior of the software deployed in a dynamic environment as it adapts to changes in the available devices or in the software itself. The metaphor has been mainly used to refer to two different aspect of liquids:

1) their ability to **flow** between different containers. The action of software deployment is often depicted as the latest version of the software being "poured" into a container [7] just like the migration of a running software application from one device to another [3]. In the latter, what is flowing is not only the software assets, but also its complete runtime execution state: liquid software features strong migration capabilities [8].

2) their ability to **adapt** to the shape of their container[1]. For example, user interfaces can adjust to fit on all available screens across all devices on which the liquid software is running [5]. Liquid big data analytics workflows can occupy all available devices across the IoT-Edge-Cloud continuum [9], [10].

In both cases, such adaptation and flow happen seamlessly, without user intervention [2] and minimal developer effort [11]. We classified each definition of liquid software we have collected in Table I, depending on which aspect (flow, adaptation, or both) has been emphasized. The dates refer to the year of publication of the earliest work on the topic.

The rest of this paper is structured as follows. We first present how existing works have introduced the liquid software metaphor (Section II), applied it to the user experience (Section III), to development and operation tools (Section IV) and to intelligent edge computing applications (Section V). We then synthesize the various interpretations of liquid software we have surveyed into a novel definition in Section VI, before we present related work in Section VII, draw some conclusions in Section VIII and sketch some future research directions in Section IX.

II. Towards Liquid Computing

A. Active Networks (1996)

One of the early references to the term liquid software was introduced in the context of active networking applications to refer to the "entire infrastructure for dynamically

[1]Unlike gases, liquids do not expand to fill up all available space

TABLE I
LIQUID SOFTWARE DEFINITIONS

Year	Term	Context	Definition	Liquidity
1996	Liquid Software [12], [13]	Active Networks	Low-level, comunication-oriented code that easily flows from machine to machine	Flow
1996	Liquid Load Balancing [14]	Load Balancing	Shifts in workload allocation are seen as liquid flow reaching a stable equilibrium in a hydrodynamic system	Flow and Adapt
2002	Liquid State Machine [15], [16]	Theoretical Computer Science	A generalization of a finite state machine to continuous time and continuous ("liquid") internal states	Flow
2003	Liquid Media [17]	Ubiquitous Computing	Seamless handover of streaming media	Flow
2005	Fluid Computing [18]	State Replication	The seamless transfer of an application's data and state between devices, possibly without user intervention.	Flow
2006	Fluid Architecture [19]	Intelligent Environments	Accommodate continuous user-induced structural changes without adversely affecting the system's behavior	Adapt
2008	Liquid Metal [20]	Programming Languages	Programming with a single high-level OO language that maps well to both CPUs and FPGAs.	Adapt
2011	Liquid Web Services [4]	Service-Oriented Computing	Provide elastic scalability to applications deployed on heterogeneous environments	Adapt
2014	Liquid Software Manifesto [2]	Multi-device User Interfaces	A multi-device user experience where software can seamlessly and effortlessly flow from one device to another	Flow and Adapt
2014	Liquid Computing [21]	Multi-device User Interfaces	Your activities, not just your data, flow from device to device	Flow
2015	Liquid Stream Processing [22]	Stream Processing Pipelines	Autonomously deal with deployment, parallelisation, migration and recovery of streaming operators	Flow and Adapt
2015	Liquid Privacy Spheres [23]	Privacy	Unclear boundaries, confusing settings make users unaware of personal information leaks	Flow
2015	Liquid Web Applications [3]	Multi-device User Interfaces	Benefit from all user-owned devices' computing, storage, and communication resources, while smoothly roaming across Web browsers following the user attention and usage context.	Flow and Adapt
2016	Liquid Web Component [24]	Web Technology	Web Component whose HTML/CSS/JS assets and dynamic state can be dynamically redeployed across different Web browsers	Flow and Adapt
2016	Liquid Context [25]	Context-Awareness	Seamless synchronization of contextual metadata for consistently personalized multi-device applications	Flow
2016	Liquid Model [26]	Model-driven Engineering	Model evolution reflecting runtime operations	Adapt
2018	Liquid Software Updates [7]	DevOps Pipelines	Practices to enable continuous updates and evolution of software systems without downtime or disruption to end users	Flow
2019	Liquid Web Worker [27]	Opportunistic Computing	Transparent offloading of Web workers to run on nearby devices	Flow
2020	Liquid Media Query [28]	Multi-device User Interface	Detect which devices, roles, users are present to declaratively control the placement of liquid Web components across a distributed user interface	Adapt
2020	Liquid Handover [29]	6G Networks	Seamless handover of tasks being shared between devices and edge nodes while devices move in the network.	Flow
2020	Liquid Mode [30]	Responsive User Interfaces	A breakthrough reading experience that enables a much easier way to read PDF documents on mobile	Adapt
2022	Liquid Functions [11]	Serverless Computing	Code offloading and placement depending on annotations, load metrics, data affinity, and expected capacities	Flow
2023	Liquid AI [31]	Machine Learning	Flexible deployment of continuously re-trained models along IoT-Edge-Cloud analysis pipelines	Flow

moving functionality throughout a network" [32]. This new paradigm was driven by the requirements of supporting mobile code [8] and the vision for seamless integration of local and remote code [12]. According to the authors, the key distinction between mobile code and liquid software lies in the ability to deploy each software component in the optimal location along a network communication path. The concept was demonstrated with a MPEG video streaming application (NetTV) implemented using the Joust framework, running over a highly-optimized Java virtual machine. The main challenges addressed by Joust [13] concerned ensuring efficient code portability to support running "low-level, comunication-oriented code that easily flows from machine to machine'.

While portability concerns have become less critical, the flexible deployment of continuously evolving software components across dynamic, distributed and diverse runtime environments still remains a challenging problem.

B. Load Balancing (1996)

By mimicking the behavior of liquids which reach a stable equilibrium level across communicating vessels, the liquid model for load balancing [14] used hydrodynamic systems as a vivid analogy to represent the process of distributing the workload across a network of computers with heterogeneous capacities. By repeatedly shifting superfluous load to direct neighbours (the approach prioritizes load sharing over load balancing) the load eventually equalizes itself: a global effect achieved asymptotically by iterating a strictly local mechanism.

C. Liquid State Machines (2002)

As a model of computations carried out by biological neural circuits, liquid state machines are designed to represent real-time and anytime computations on continuous input streams [15]. In this work the liquid metaphor refers to the lack of predefined structure in the internal states and the continuous nature of the input/output signals. For example, the liquid internal states "consisted of a randomly connected circuit of just 135 spiking neurons" [16] or even literally a bucket of water "harnessed to solve nonlinear pattern recognition problems" [33].

D. Liquid Media (2003)

The Liquid Computing System for Liquid Media from Motorola Labs was a Multi-Device Streaming Media Orchestration Framework [17]. The work identified the lack of "peripheral vision" of devices as a problem to support dynamic discovery of resources in pervasive computing applications. The proposed framework relies on a "liquid computing server" to keep track of the available devices and "select the best set of resources for rendering content". With it, users could migrate streaming media sessions across devices as their location changed [34].

While the technology addressed the seamless handover of multimedia streams between devices, we could not find an explicit depiction of the liquid metaphor in this work beyond using it to name key system components.

E. Fluid Computing (2005)

Fluid [18] is a middleware for replicating application data so that it can "flow, as a liquid, between devices". Transparent replication of application data and state is achieved in a decentralized, peer to peer network of mobile devices by dealing with intermittent connectivity issues. The Fluid middleware is designed to be seamlessly integrated within the Model-View-Controller architecture of mobile applications. As a consequence, users do not need to manage the physical location of their data, which is automatically synchronized between their devices. Developers do not have to redesign their applications, as long as they strictly follow the Model-View-Controller pattern using Models which implement a *liquefy* method. This is at the foundation of a weak and asynchronous approach to replication [35] based on operations which can be rolled back or rolled forward by the middleware. The synchronization protocol supports two modes of operation: "trickle" and "batch" [36]. Trickle replication occurs in real-time, propagating each update as it is generated (e.g., as the user enters keystrokes) as long as there is some connectivity. Batch mode replication refers instead to the process of exchanging all accumulated updates at once after regaining connectivity following a disconnection. During synchronization, the middleware can resolve syntax-level conflicts by reordering operations and will inform the application code about semantic-level conflicts.

In this work the liquid metaphor is used to represent the seamless (from the user perspective), implicit (without developers worrying about implementing it in their applications), and epidemic (between nearby devices without requiring a central Cloud-based master copy) transfer of application state across user devices, making it possible for users to pick up a work session on a different device without having to explicitly transfer it or request a synchronization action. This is the result of solving the state transfer problem at the model layer through a data replication protocol, under the assumption that a suitable view layer is already present and ready to display the updated model on each device.

F. Intelligent Environments (2006)

Intelligent environments are defined by their key ability of being resilient to changes in their structural and physical configurations initiated by users, without disrupting the overall system behavior. The "Fluid architectures" [19] of such environments need to be capable of accommodating continuous structural modifications while maintaining system stability and correct operation. They are defined by capturing how the structure of intelligent environments can evolve over time in terms of defining the sets of permissible architecture configurations and rules to anticipate which transitions are possible. Possible reconfiguration actions included not only adding, moving, replacing or removing individual components, but also changing the location in which the system is deployed, as well as the possibility to merge multiple systems into a single deployment.

The author used the fluid concept to illustrate the dynamic reconfiguration of the software and the corresponding flexibility of its architecture. No attempts are made to further develop the metaphor and apply it to the corresponding software deployed in an intelligent environment.

G. Liquid Metal (2008)

The Liquid Metal (LIME) extensions to the Java programming language use the liquid metaphor to refer to the automated translation of "large portions of a program in hardware via direct synthesis into a programmable or reconfigurable logic fabric such as Field Programmable Gate Array (FPGA)" [20]. By employing a unified programming language, the authors simplify the intricacies of domain crossing between CPU and FPGA. Additionally, it becomes possible to support a fluid movement (i.e., a seamless transfer) of computations between them and optimize the execution based on efficiency gains or resource availability constraints.

H. Liquid Services (2011)

Liquid services [37] are capable of 1) handling varying workloads and traffic patterns, 2) dynamically adjusting to different environmental conditions, such as changing resource availability. In other words, they feature elastic scalability by taking advantage of a wide variety of heterogeneous deployment environments, including both shared-nothing virtualized Cloud computing clusters and shared-memory multicore architectures.

Liquid services satisfy the SAFE qualities [4]: they are Scalable, Adaptive with respect to their environment, Flexible thus supporting heterogeneity, and Elastic with respect to their workload. The metaphor was introduced to highlight the flexible nature of the deployment, with the ability of the services themselves to fill up (i.e., fully use) all kinds of available container resources in response to changes in both environment and workload.

In the design process of liquid services, it is important to delay making design decisions related to service deployment until runtime. Achieving this is helped by the following design constraints [38]: 1) services should have a fine-granular structure, allowing for flexible deployment, replication, and dynamic migration. 2) connectors between services should be underspecified, so that the most efficient communication mechanism available at runtime can be employed. 3) Service interfaces should explicitly define the semantics of interactions, particularly regarding how these affect the state of services, to select appropriate replication mechanisms.

I. Liquid Stream Processing (2015)

Data stream processing pipelines provide a natural environment in which to embed the liquid software metaphor, as the data continuously flows through pipes and filters, as it springs from sources into sinks [39]. Liquid stream processing targets the automatic deployment of stream processing operators, which is no longer statically pre-determined along the pipeline. A liquid stream processing operator can dynamically change its location, moving closer to the source or closer to the sink depending on the constraints and the opportunities provided by its execution environment and the corresponding network conditions [22].

III. Liquid User Experience

While the early forays into liquid computing focused on the flexible deployment of headless software components across the network, the liquid metaphor was also adopted to describe a novel, activity-centric user experience, vs. the traditional device-centric user experience born with the personal computing (PC) age. In other words, "a world where both data and activities move around as needed" [21].

A. Liquid Software Manifesto (2014)

Motivated by the explosive growth in network-connected devices owned by individual users [5], [40] and inspired by Mark Weiser's vision of the disappearing computer [41], the liquid software manifesto launched a call for action to make multiple device ownership as "casual, fluid, and hassle-free as possible". The authors define liquid software as "a multi-device software experience that can seamlessly and effortlessly flow from one device to another" [2]. The focus is on the experience and the efficiency of the users, whereby users are empowered to "seamlessly roam and continue their activities on any available device".

Given its major impact in subsequent developments, in the following we report the main requirements[2] for liquid software enumerated by the manifesto [2]:

1) **Seamless roaming** – Users shall be able to effortlessly roam between all the computing devices that they have;

2) **Maintenance-free** – Such roaming shall be as casual, fluid and hassle-free as possible;

3) **Transparent Synchronization** – Applications and data shall be synchronized transparently between all user devices, insofar they are compatible with such applications and data;

4) **Strong Migration** – Such roaming shall include the transportation and synchronization of the full state of each application, so that users can seamlessly continue their previous activities on any device;

5) **Platform Independent** – Such roaming shall not be limited to devices from a single vendor ecosystem only; any compatible device with adequate resources from any vendor shall run liquid software;

6) **Privacy** – Users should remain in control of the liquidity of applications and their data, with the option to restrict access to specific functionality or data on specific devices.

B. Liquid Web Applications (2015)

The call of the Liquid Software Manifesto found resonance within the Web engineering community, leading to the vision of Liquid Web Applications [3]. Unlike existing proprietary efforts such as Apple Continuity [42], Samsung Flow, Microsoft Continuum – Web technology can satisfy the requirements for platform independence, with many popular Web applications

[2]Titles added by the author

featuring support for multi- and cross-device workflows in single user but also multi-user collaboration scenarios.

Liquid Web applications have the capability to fully utilize the computing, storage, and communication resources of all devices owned by the end user. Additionally, they are designed to seamlessly and dynamically migrate [43] from one device to another in real-time, in response to the user's attention and changing usage context. As opposed to treating the same user connecting from different devices as a security threat, liquid Web applications adapt their user interface to the *set of* devices currently connected. Responsive ones do so only for one device at a time [44] – with the exception of the "Liquid Mode" of the mobile Acrobat Reader app, a "revolutionary mobile reading experience powered by machine learning technology" [30] which reflows the layout of a PDF document to enhance its readability on the small screen of a *single* mobile phone or tablet device.

Table II summarizes the key properties defining the behavior of a liquid vs. solid (traditional) Web application [3]. While it has always been possible to migrate the state of a Web application by encoding it as a URL and sharing the URL across different Web browsers, more complex solutions based on the WebSockets and WebRTC protocols are needed to support multi-device collaboration scenarios which require real-time synchronization [45].

C. Liquid Software Design Space (2016)

The design space of Liquid Software has been mapped in [46] and further refined in [47]. We report the most important design issues and alternatives in Table III. They can be used to position and compare to which extent and under which constraint alternative technologies implement the liquid software concept. For example, migration and synchronization is rather easy to achieve when relying on a highly available centralized storage element (e.g., deployed in the Cloud) and devices simply attach or detach a view to display the same shared information. Building the same liquid experience on top of a mesh network [48] of mobile phones or any kind of decentralized architecture remains more challenging[3].

D. Liquid Context (2016)

The liquid metaphor was also applied to context-aware applications [25]. Here the context in which an application is running is kept seamlessly synchronized across all devices running the application. This way applications are consistently adapted to the user preference settings, their location, goals and ongoing activities, social relationships, as well as many other contextual aspects, no matter which device is running them. Given the large amounts of contextual metadata involved, developers can explicitly control which are the portions of the user virtual profile that should be kept synchronized.

[3]One of the most prominent efforts to re-decentralized the Web and get end-users back in control of their content and their devices – a result that could be easily achieved with liquid software – has been named the Social Linked Data (Solid) project [49], [50].

TABLE II
LIQUID VS. SOLID WEB APPLICATIONS [3]

Element	Liquid	Solid
Code Mobility [8]	Strong Migration	Weak Migration
State Synchronization	Migration and Cloning	Refresh
Adaptation	Dynamic	Restart required
Adaptation Target	Set of Devices	Single Device

TABLE III
LIQUID SOFTWARE DESIGN SPACE [46], [47]

Issue	Alternatives
User Experience	
Device Usage	Sequential, Parallel
User Interface Adaptation	Manual, Responsive, Complementary
Primitives	Migrate, Clone, Forward, Fork
Discovery and Pairing	Smartcard, WiFi, Bluetooth, Shared URL, QR Code, Geolocation, Contact List
Developer Experience	
Granularity	OS, VM, Container, Application, Component
State Identification	Implicit, Explicit
State Synchronization	Trickle, Batch
State Replication	Multi-primary, Primary/Replica
Topology	Centralized, Decentralized, Hybrid
Deployment	Pre-Installed, Cached, On-Demand
Deployment Source	Single Repository, Multiple Repository, Peer Repository

IV. TOOLS AND FRAMEWORKS

A. Elastic Database (2015)

The elastic database (EDB [51]) supports automatic multi-master synchronization between multiple mobile devices in the presence of intermittent network connectivity. It is positioned as a fundamental building block for liquid software, making it possible to keep application data and state synchronized across devices. To "support the illusion of truly liquid user interface behavior", the authors reiterate that "it must be possible to carry user interface state information from one device to another as efficiently as possible.".

This approach shares common requirements (e.g., partially offline operation, update-anywhere replication) with the Fluid computing middleware [36], although it allows for a centralized Cloud-based backup of the data to preserve it in case some (or even all) users devices are lost. Replication between user devices should still be possible however also in the absence of the centralized Cloud master copy.

B. Liquid.js (2016)

Developers of liquid Web applications need to control how to expose the liquid behavior of their cross-device Web applications to the users. The API of the Liquid.js framework [52] offers primitives to manage how different devices are discovered, paired and disconnected as well as the complete lifecycle of "liquid Web components" as they are deployed and instantiated within Web browsers running across multiple

devices. Concerning stateful components [24], it offers low-level mechanisms to describe which are the "liquid properties", whose state is migrated and if necessary kept synchronized across different devices. Other features include the offloading of "liquid Web workers" on paired devices [27] and the synchronization of "liquid storage", again, across all connected devices. "Liquid media queries" are used to detect the number of users/devices connected to the liquid Web application, or the role played by each device to control the deployment, dynamic migration and cloning of liquid Web components across multiple browsers [28]. A higher-level API to perform the migration, fork, or cloning of entire Web components and applications is also provided. Such operations may happen through a WebRTC peer to peer communication channel, thus demonstrating the possibility of developing a liquid Web component on one device and transferring it to another device without relying on a centralized Web server [53].

C. Liquid Model (2016)

A different interpretation of the liquid metaphor was introduced to describe models which "should not be isolated and frozen, but reusable and evolutionary" [26]. While design-time models have been used to prescribe how a software architecture should be built, they can also describe how a software system has actually been built [54]. While there is often a drift between the two, the goal of liquid models is to reconcile the 'to-be' with the 'as-is' version of the model. Liquid models do not simply abstract the underlying code, but they are obtained by mining multiple heterogeneous data streams monitoring the operational behaviour of a running software system so that they reflect "also the feedback after the release from the operation, to cover the complete lifecycle of a system".

D. Development and Operation Tools (2018)

In the context of Internet of Things applications, it has been challenging to keep software tied to embedded devices up-to-date [55] as the devices are installed away in the field and there is a risk of "bricking" them [56]. A recent interpretation of liquid software concerns its ability "to continuously update itself". Unlike traditional software suffering from downtimes and disruptions during manual or automated upgrades, liquid software is made up of "tiny pieces, like drops of water make up the ocean". Thanks to user expectations for non-disruptive updates driving the construction of more and more advanced DevOps build pipelines, the software flowing through them has become liquid. Devices will be "connected to software pipes that stream updates" into them. As a consequence, "software patches can instantly incorporated into running systems" [7]. Achieving this requires security mechanisms to avoid supply chain attacks, quality assurance processes to release software with high confidence, and a new kind of versioning metadata to transparently identify and track which "versionless" software has been automatically installed.

While the liquid user experience main target was the migration of running software across a dynamic set of user devices, here the main problem concerns changes to the software itself (Table IV) and not (only) to the environment in which it runs (Table II).

TABLE IV
LIQUID VS. SOLID SOFTWARE UPDATES [7]

Element	Liquid	Solid
Update Action	Automated	Manual
User Impact	Seamless	Disruptive
Restart	None	Required
Update Flow	Continuous	Discrete
Package Size	Small	Large
Change Delivery	Incremental	Entire Version Snapshot

E. Collaborative Development Tools - LiquiDADE (2018)

LiquiDADE [57] applies the liquid user experience concept to collaborative software development tools, whereby multiple users can edit together on the same code artifacts. The tool combines a live (or real-time [58]) collaborative editor with a WebRTC-based audio/video conferencing system and offers server-side compilation, deployment and testing of the code. It does not specifically facilitates or targets the development of liquid software.

V. INTELLIGENT EDGE COMPUTING

The development of intelligent edge computing applications is currently limited by divergent tools, methods, and practices targeting different kinds of devices and platforms across the IoT-Edge-Cloud continuum [59]. This creates significant friction in the ability of software to seamlessly flow between devices. As a consequence, decisions on where to locate intelligence in the network topology must be made at design time due to the difficulty of relocating computations without prior preparation [29].

The opportunity to overcome such limitations is represented by next-generation telecommunication networks, where liquid software has been mentioned in the context of 6G networks [60]. In this case, the metaphor is applied to both data and resources. "In the data liquid model, the flow of data is dynamic, and the data generated on devices" can be fed to the appropriate intelligent edge devices. Additionally, "virtualized communication and computation resources are also fluid, and can be allocated to different nodes with different requirements".

A. Isomorphic Software Architecture (2021)

The concept of isomorphic software architectures [10] has been introduced to refer to software which can be used to create applications that run on different types of devices/platforms, without requiring to maintain platform-specific variants. Literally, from a developer's perspective, isomorphic software components do not change shape as they are deployed across different types of containers. This can be achieved by using containers or runtime environments which abstract the heterogeneity of the underlying platforms, or programming languages which can be compiled targeting

multiple runtime platforms. Isomorphic software can thus be seen as a pre-requisite for liquid software, providing the necessary portability and making it "easier, faster, and potentially cheaper to develop ioT applications" whose components can be dynamically redeployed from the edge to the cloud and back.

This vision should be achieved with minimal developer effort, for example, by introducing simple annotations into the code, as proposed by the "Liquid Functions" framework [11], or by compiling to WebAssembly [61] to ensure portability.

B. WebAssembly (2021)

Flexible deployment of liquid software requires a suitable packaging and runtime container. In [61], the authors propose to use WebAssembly [62] modules as a technology platform for the deployment of liquid applications which is sufficiently lightweight to be also used for IoT devices. Originally introduced within Web Browsers, this technology is also compatible with Cloud deployments, making it suitable for running the same WASM code across the computing continuum. While this greatly simplifies code relocation, developers still need to explicitly manage the migration of the stateful modules, until more advanced serialization techniques become mainstream [43], [63].

C. Liquid Functions (2022)

The functions as a service (FaaS) concept from serverless computing [64] is revisited and applied to code which is meant to run across the entire IoT-Edge-Cloud continuum. Liquid functions "are abstracted to become useful for applications highly distributed across a topology of nodes that do not necessarily have full mutual visibility" [11]. Unlike the original FaaS concept – invoking functions exclusively at their deployment locations – liquid functions can "traverse a continuum as needed". This includes the ability of functions to be *embedded* in the caller for minimal invocation overhead, as well as for functions to be *pinned* to specific container nodes. No explicit packaging or pre-deployment is required as the framework makes all deployment, binding and routing decisions "until 100% correctness along with a peak performance are reached as irrevocable optimisation goals". While in the extreme case each invocation may result in executing the function on a different node, there are some fixed image deployment costs that need to be amortized across multiple calls. Likewise, it is not clear whether long-running liquid functions are mobile while they are being executed, or only across separate invocations.

D. Liquid AI (2023)

Another use case scenario for liquid software on the edge is motivated by the rise of edge intelligence which has led to the development of technologies that enable the scaling down of AI/ML components to small devices (e.g., TensorFlow Lite, AIfES, TinyML). This should be naturally followed by the seamless migration of AI/ML components between the Cloud and the Edge [1].

More in detail, considering machine learning models as a particular type of software, trained and validated by using specific algorithms on given curated input datasets, leads to the question of whether also such software can be susceptible of becoming liquid. This would imply that "liquid ML models" [31] can be dynamically deployed across different edge devices to make efficient use of all available resources, reduce the latency between the sensors fed into the classifiers as well as guarantee the privacy of the prompts engineered by their users, as well as be kept continuously up-to-date when the original training data is improved [65]. In other words, "If we want a world of AI, we need a world of data fluidity. Data needs to be free to move from system to system, from platform to platform, without transfer fees, egress or other nonsense" [66]. The developer experience of such liquid intelligent edge applications would be as seamless as the liquid user experience illustrated above, with no need for concern regarding the deployment and maintenance of the ML pipeline.

VI. Redefining Liquid Software

As we have seen, the liquid software metaphor has been introduced in many different contexts across the entire software stack: from network interconnects, load balancers, stream processing pipelines, continuous build, integration and delivery pipelines, programming languages, replicated databases, state machines, software functions, software components, software architectures, all the way to multi-device and cross-device user interfaces and Web technology. What do these uses of the term liquid have in common? And is it possible to synthesize a unique definition?

Liquid software provides a seamless user and developer experience. No effort is required to install, maintain, and keep liquid software synchronized and up-to-date across multiple devices. Just as users can swipe windows across different screen displays connected to the same computer, with liquid software they can perform the same spontaneous action across different devices. Likewise developers should not have to dedicate significant effort to obtain liquid software: adding some code annotations [11], injecting a new behavior into existing components [24], deploying isomorphic software in the right kind of container [10] – these simple steps should be sufficient.

Liquid software flows between containers while it runs and adapts to their shape. It fills them all up, taking full advantage of available resources for enhanced usability, convenience, scalability, reliability and performance. Such behavior is an intrinsic property of the software being liquid.

More precisely, as a software quality attribute [67], "liquidity" can be seen as a form of "flexibility" [68] combined with "deployability" [69]. Liquid software implies flexible deployment: deployment (and re-deployment) in the presence of changes. These changes can affect the environment (i.e., the set of available containers) in which the software is running. As a new container is added, the software immediately flows into it. As a container is removed, the software continues seamlessly running in the remaining containers. Changes can

also affect the software itself. As a new version of the software is released, it should immediately flow into all containers and seamlessly replace the previous version running there.

In this context, the same term "migration" can be applied both to the transfer of the execution state of the software from one container to another, but also to the transfer of the execution state of the software from its current version to the next one. The same primitive can thus accommodate changes that require the software – while it is running – to flow between containers or changes that make a new version of the software flow into the same containers replacing the previous version.

The viscosity of liquid software can be measured in presence of changes by observing the time and effort required to redeploy the software as a consequence of the changes applied to its runtime environment or to the software itself. Liquid software presents minimal effort and very small (i.e., sub-second) migration time. Solid software cannot be redeployed, nor it can be updated without resetting its execution state. In the extreme case, it cannot be changed and redeployed at all. This might be a useful property to guarantee stability (not all changes are necessarily improvements) and to prevent leaks of sensitive application state onto untrusted devices.

VII. Related Work

The liquid metaphor has also been applied beyond the software domain, for example in philosophical and sociological reflection to characterize the most significant aspects of the present reality: "a dimension in which the lasting gives way to the transient, the need to the desire, and the necessity to the utility" [70].

Another example is the LiquidPub [71] project, whose goal was to capture the "opportunities provided by the Web and open source, agile software development to develop concepts, models, metrics, and tools for an efficient (for people), effective (for science), and sustainable (for publishers and the community) way of creating, disseminating, evaluating, and consuming scientific knowledge".

The liquid metaphor has been also used to illustrate the lack of privacy guarantees. While a solid boundary does not leak personal information, a liquid boundary is loosely defined. This image has been used to represent how some users are unaware or confused about the implications on their personal information by the privacy policies and terms of service of many smartphone apps [23].

There are also other nature-inspired metaphors that have been applied to software, among others, in pervasive [72] and distributed [73] computing, self-organizing systems [74] or telecommunication technology [75]. Common goals for introducing such metaphors include achieving qualities such as flexibility, resilience, self-adaptation and self-organization, sustainability in the face of diversity and heterogeneity under the constraint of decentralization. Also the solid vs. liquid (vs. gas) state of matter could be considered as a nature-inspired metaphor from the physical domain, on a largely more coarse-grained level than considering "service components as sort of computational particles living in a world of [...] virtual computational force fields" [72].

VIII. Conclusion

In this paper we have tracked the history of the liquid software metaphor, from its inception in the context of active networks and load balancing (1996) until its latest appearance as liquid functions as a service (2022) and liquid AI (2023). Liquid software offers a high degree of flexibility and ease of deployment, allowing for quick updating of software versions and seamless adaptation to changing runtime environments. With minimum efforts required for updates and changes, liquid software provides a fitting depiction of desired qualities of intelligent edge applications where flexibility, responsiveness, resilience, smooth upgrades, user satisfaction and privacy preservation are critical factors.

IX. Outlook

While many existing efforts have focused on the liquid user experience, the developer experience of liquid software needs more attention. Delivering liquid software through continuous and non-disruptive updates requires on the one hand to set up a suitable build, quality assurance, release and delivery pipeline. On the other hand, the development process of liquid software could benefit from introducing live programming and coding techniques [76] featuring an increased level of liveness [77], [78].

As we have seen, the liquid metaphor has been applied to many software artifacts of different granularity (from liquid functions [11] to liquid components [24] and liquid models [26], from liquid storage [51], [52] to liquid stream processing [22] and liquid web services [4]). Many other architectural elements and programming language constructs could be liquefied. For example, liquid objects and classes, liquid virtual machines, liquid file systems, liquid algorithms and liquid data structures are still waiting to be discovered.

Given the ease with which liquid software can roam across different devices while carrying its runtime state and associated data assets, its deployment locations should be chosen with care. Simply put, liquid software requires secure and trusted containers to avoid uncontrolled leaks.

The liquid software metaphor can be further stretched by using it to distinguish different deployment locations, in particular concerning the ownership and control of the containers or devices in which the software is deployed. This way software running on trusted devices can be clearly distinguished from software running on untrusted devices. For example, liquid software runs on a set of personal user devices or across a company-wide sensor network. Liquid software changes state (it is no longer considered liquid) as it is deployed to run on a Cloud or Fog computing platform (where it behaves like a gas), given the increased latency and lack of transparency concerning the ownership and control of such rented and shared devices [79].

ACKNOWLEDGEMENTS

The author would like to convey deep gratitude to Daniele Bonetta, Andrea Gallidabino, and Masiar Babazadeh for their generous contributions to explore the topic of liquid software during their PhD. The author is also indebted to Tommi Mikkonen, Kari Systä, and Antero Taivalsaari for their unwavering belief in the importance of this topic. The author would like to extend sincere gratitude to Raveendra Vvs for his thought-provoking exchange.

REFERENCES

[1] A. Taivalsaari, T. Mikkonen, and C. Pautasso, *Towards Seamless IoT Device-Edge-Cloud Continuum*, ser. Communications in Computer and Information Science (CCIS). Springer, 2021, vol. 1508, pp. 82 – 98.

[2] A. Taivalsaari, T. Mikkonen, and K. Systä, "Liquid software manifesto: The era of multiple device ownership and its implications for software architecture," in *2014 IEEE 38th Annual Computer Software and Applications Conference*. IEEE, 2014, pp. 338–343.

[3] T. Mikkonen, K. Systä, and C. Pautasso, "Towards liquid web applications," in *15th International Conference on Web Engineering (ICWE 2015)*. Rotterdam, NL: Springer, June 2015, pp. 134–143.

[4] D. Bonetta and C. Pautasso, "An architectural style for liquid web services," in *9th Working IEEE/IFIP Conference on Software Architecture (WICSA 2011)*, Boulder, CO, USA, June 2011, pp. 232–241.

[5] F. Brudy, C. Holz, R. Rädle, C.-J. Wu, S. Houben, C. N. Klokmose, and N. Marquardt, "Cross-device taxonomy: Survey, opportunities and challenges of interactions spanning across multiple devices," in *Proceedings of the 2019 chi conference on human factors in computing systems*, 2019, pp. 1–28.

[6] M. Husmann, M. Spiegel, A. Murolo, and M. C. Norrie, "Ui testing cross-device applications," in *Proceedings of the 2016 ACM International Conference on Interactive Surfaces and Spaces*, 2016, pp. 179–188.

[7] F. Simon, Y. Landman, and B. Sadogursy, *Liquid Software - How to Achieve Trusted Continuous Updates in the DevOps World*. JFrog, 2018.

[8] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Transactions on software engineering*, vol. 24, no. 5, pp. 342–361, 1998.

[9] D. Balouek-Thomert, E. G. Renart, A. R. Zamani, A. Simonet, and M. Parashar, "Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows," *The International Journal of High Performance Computing Applications*, vol. 33, no. 6, pp. 1159–1174, 2019.

[10] T. Mikkonen, C. Pautasso, and A. Taivalsaari, "Isomorphic internet of things architectures with web technologies," *Computer*, vol. 54, no. 7, pp. 69–78, 2021.

[11] J. Spillner, "Self-balancing architectures based on liquid functions across computing continuums," in *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC'21)*, 2022.

[12] J. Hartman, U. Manber, L. Peterson, and T. Proebsting, "Liquid software: A new paradigm for networked systems," Technical Report 96, Tech. Rep., 1996.

[13] J. J. Hartman, P. A. Bigot, P. Bridges, B. Montz, R. Piltz, O. Spatscheck, T. A. Proebsting, L. L. Peterson, and A. Bavier, "Joust: A platform for liquid software," *Computer*, vol. 32, no. 4, pp. 50–56, 1999.

[14] D. HENRICH, "The liquid model load balancing method," *Parallel Algorithms and Applications*, vol. 8, no. 3-4, pp. 285–307, 1996.

[15] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[16] W. Maass, "Liquid computing," in *Computation and Logic in the Real World*, 2007, pp. 507–516.

[17] J. Mysore, V. Vasudevan, J. Almaula, and A. Haneef, "The liquid media system——a multidevice streaming media orchestration framework," in *Ubicomp 2003 Workshop: Multi-Device Interfaces for Ubiquitous Peripheral Interaction*, 2003.

[18] D. Bourges-Waldegg, Y. Duponchel, M. Graf, and M. Moser, "The fluid computing middleware: bringing application fluidity to the mobile internet," in *The 2005 Symposium on Applications and the Internet (SAINT 2005)*. IEEE, 2005, pp. 54–63.

[19] G. Kortuem, "Modelling and evaluating fluid software architectures for intelligent environments," in *Proc. 2nd IET International Conference on Intelligent Environments (IE 06)*. IET, 2006, pp. 201–210.

[20] S. S. Huang, A. Hormati, D. F. Bacon, and R. Rabbah, "Liquid metal: Object-oriented programming across the hardware/software boundary," in *Proc. of the 22nd European Conference on Object-Oriented Programming (ECOOP 2008)*. Springer, 2008, pp. 76–103.

[21] G. Gruman, "Welcome to the next tech revolution: Liquid computing," *InfoWorld*, 2014. [Online]. Available: https://www.infoworld.com/article/2608440/

[22] M. Babazadeh, A. Gallidabino, and C. Pautasso, "Decentralized stream processing over web-enabled devices," in *4th European Conference on Service-Oriented and Cloud Computing (ESOCC 2015)*, vol. 9306. Taormina, Italy: Springer, September 2015, pp. 3–18.

[23] A. Serrano Tellería and M. Oliveira, "Liquid spheres on smartphones: The personal information policies." *International journal of interactive mobile technologies*, vol. 9, no. 1, 2015.

[24] A. Gallidabino and C. Pautasso, "The liquid.js framework for migrating and cloning stateful web components across multiple devices," in *25th International World Wide Web conference (WWW 2016)*. Montreal, Canada: ACM, April 2016, pp. 183–186. [Online]. Available: http://www2016.net/proceedings/companion/p183.pdf

[25] J. Berrocal, J. Garcia-Alonso, C. Canal, and J. M. Murillo, "Liquid context: Migrating the users' context across devices," in *Proc. ICWE 2016 International Workshops, DUI, TELERISE, SoWeMine, and Liquid Web*. Springer, 2016, pp. 128–141.

[26] A. Mazak and M. Wimmer, "Towards liquid models: An evolutionary modeling approach," in *Proc. 18th Conference on Business Informatics (CBI)*, 2016, pp. 104–112.

[27] A. Gallidabino and C. Pautasso, "The LiquidWebWorker API for horizontal offloading of stateless computations," *Journal of Web Engineering*, vol. 17, pp. 405–448, March 2019.

[28] ——, "Multi-device complementary view adaptation with liquid media queries," *Journal of Web Engineering (JWE)*, vol. 18, 761–800 2020. [Online]. Available: https://journals.riverpublishers.com/index.php/JWE/article/view/1099

[29] E. Peltonen, M. Bennis, M. Capobianco, M. Debbah, A. Ding, F. Gil-Castiñeira, M. Jurmu, T. Karvonen, M. Kelanti, A. Kliks *et al.*, "6g white paper on edge intelligence," *arXiv preprint arXiv:2004.14850*, 2020.

[30] Acrobat, "Liquid mode." [Online]. Available: https://blog.adobe.com/en/publish/2020/09/23/adobe-unveils-ambitious-multi-year-vision-for-pdf-introduces-liquid-mode

[31] K. Systä, C. Pautasso, A. Taivalsaari, and T. Mikkonen, "Liquidai: Towards an isomorphic ai/ml system architecture for the cloud-edge continuum," in *Proceedings of the 23rd IEEE International Conference on Web Engineering (ICWE2023)*, 2023.

[32] J. Hartman, L. Peterson, A. Bavier, P. Bigot, P. Bridges, B. Montz, R. Piltz, T. Proebsting, and O. Spatscheck, "Experiences building a communication-oriented javaos," *Software: Practice and Experience*, vol. 30, no. 10, pp. 1107–1126, 2000.

[33] C. Fernando and S. Sojakka, "Pattern recognition in a bucket," in *Advances in Artificial Life*, W. Banzhaf, J. Ziegler, T. Christaller, P. Dittrich, and J. T. Kim, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 588–597.

[34] J. Mysore and V. Vasudevan, "A reconfigurable stream orchestration framework for mobile users," in *Proceedings Third International Conference on Mobile Data Management (MDM 2002)*, 2002, pp. 35–42.

[35] Y. Lin, B. Kemme, M. Patino-Martinez, and R. Jimenez-Peris, "Enhancing edge computing with database replication," in *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. IEEE, 2007, pp. 45–54.

[36] M. Graf, "Fluid computing," *ERCIM News*, no. 54, pp. 21–22, 2003.

[37] D. Bonetta and C. Pautasso, "Towards liquid service oriented architectures," in *Proc. of the 20th International Conference on World Wide Web (WWW 2011) PhD Symposium*, Hyderabad, India, 2011, pp. 337–342.

[38] D. Karastoyanova, M. Carro, D. Ivanovic, C. D. Napoli, M. Giordano, Z. Neméth, and C. Pautasso, "Research challenges on service technology foundations," in *Proc. ICSE Workshop on European Software Services and Systems Research - Results and Challenges (S-Cube)*, June 2012, pp. 27–33.

[39] M. Babazadeh, "Liquid stream processing on the web: a javascript framework," PhD, USI, Lugano, Switzerland, November 2017. [Online]. Available: https://doc.rero.ch/record/306887

[40] M. Levin, *Designing Multi-device Experiences: An Ecosystem Approach to User Experiences Across Devices*. O'Reilly, 2014.

[41] M. Weiser, "The computer for the 21st century," *Scientific american*, vol. 265, no. 3, pp. 94–105, 1991.

[42] G. Gruman, "Apple's Handoff: What works, and what doesn't," *InfoWorld*, Oct. 7, 2014. [Online]. Available: http://www.infoworld.com/article/2691101/ios/apples-handoff-what-works-and-what-doesnt.html

[43] Y.-H. Yoo and S.-M. Moon, "Snapshot-based migration of es6 javascript," in *Proc. 21st International Conference on Web Engineering (ICWE)*, 2021, pp. 404–419.

[44] E. Marcotte, *Responsive Web Design*. Editions Eyrolles, 2011.

[45] A. Gallidabino and C. Pautasso, "Maturity model for liquid web architectures," in *17th International Conference on Web Engineering (ICWE 2017)*, vol. 10360. Rome, Italy: Springer, June 2017, pp. 206–224.

[46] A. Gallidabino, C. Pautasso, V. Ilvonen, T. Mikkonen, K. Systä, J.-P. Voutilainen, and A. Taivalsaari, "On the architecture of liquid software: Technology alternatives and design space," in *13th Working IEEE/IFIP Conference on Software Architecture (WICSA 2016)*, Venice, Italy, April 2016.

[47] A. Gallidabino, C. Pautasso, T. Mikkonen, K. Systa, J.-P. Voutilainen, and A. Taivalsaari, "Architecting liquid software," *Journal of Web Engineering*, vol. 16, pp. 433–470, September 2017. [Online]. Available: http://www.rintonpress.com/journals/jweonline.html#v16n56

[48] P. N. Thai and H. Won-Joo, "Hierarchical routing in wireless mesh network," in *The 9th International Conference on Advanced Communication Technology*, vol. 2, 2007, pp. 1275–1280.

[49] A. V. Sambra, E. Mansour, S. Hawke, M. Zereba, N. Greco, A. Ghanem, D. Zagidulin, A. Aboulnaga, and T. Berners-Lee, "Solid: a platform for decentralized social applications based on linked data," *MIT CSAIL & Qatar Computing Research Institute, Tech. Rep.*, 2016.

[50] F. Parrillo and C. Tschudin, "Solid over the interplanetary file system," in *2021 IFIP Networking Conference (IFIP Networking)*, 2021, pp. 1–6.

[51] O. Koskimies, J. Wikman, T. Mikola, and A. Taivalsaari, "Edb: a multi-master database for liquid multi-device software," in *2015 2nd ACM International Conference on Mobile Software Engineering and Systems*. IEEE, 2015, pp. 125–128.

[52] A. Gallidabino and C. Pautasso, "The liquid user experience api," in *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 767–774.

[53] A. Gallidabino, "Liquid web applications - design and implementation of the decentralized cross-device web," PhD, USI, Lugano, June 2020. [Online]. Available: https://doc.rero.ch/record/328744

[54] R. N. Taylor, N. Medvidovic, and E. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009.

[55] J. Bauwens, P. Ruckebusch, S. Giannoulis, I. Moerman, and E. D. Poorter, "Over-the-air software updates in the internet of things: An overview of key principles," *IEEE Communications Magazine*, vol. 58, no. 2, pp. 35–41, 2020.

[56] F. J. A. Padilla, E. Baccelli, T. Eichinger, and K. Schleiser, "The future of iot software must be updated," in *IAB Workshop on Internet of Things Software Update (IoTSU)*, 2016.

[57] C. De Meo, N. Siena, L. Riccardi, F. Nocera, A. Parchitelli, M. Mongiello, E. Di Sciascio, and N. Mäkitalo, "Liquidade: a liquid-based distributed agile and adaptive development environment (DADE) multi-device tool," in *Proceedings of the 1st ACM SIGSOFT International Workshop on Ensemble-Based Software Engineering*, 2018, pp. 9–12.

[58] C. A. Ellis and S. J. Gibbs, "Concurrency control in groupware systems," in *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, 1989, pp. 399–407.

[59] J. Spillner, J. F. Borin, and L. F. Bittencourt, "Intent-based placement of microservices in computing continuums," in *Future Intent-Based Networking: On the QoS Robust and Energy Efficient Heterogeneous Software Defined Networks*. Springer, 2021, pp. 38–50.

[60] T. Yang, M. Qin, N. Cheng, W. Xu, and L. Zhao, "Liquid software-based edge intelligence for future 6g networks," *IEEE Network*, vol. 36, no. 1, pp. 69–75, 2022.

[61] N. Mäkitalo, T. Mikkonen, C. Pautasso, V. Bankowski, P. Daubaris, R. Mikkola, and O. Beletski, "WebAssembly modules as lightweight containers for liquid IoT applications," in *International Conference on Web Engineering*. Springer, 2021, pp. 328–336.

[62] A. Rossberg, B. L. Titzer, A. Haas, D. L. Schuff, D. Gohman, L. Wagner, A. Zakai, J. Bastien, and M. Holman, "Bringing the web up to speed with WebAssembly," *Communications of the ACM*, vol. 61, no. 12, pp. 107–115, 2018.

[63] J. Ménétrey, M. Pasin, P. Felber, and V. Schiavoni, "Webassembly as a common layer for the cloud-edge continuum," in *Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge (FRAME 2022)*, 2022, p. 3–8.

[64] J. Chapin and M. Roberts, *What is Serverless*. O'Reilly Media, Inc, 2017.

[65] E. Peltonen, I. Ahmad, A. Aral, M. Capobianco, A. Y. Ding, F. Gil-Castineira, E. Gilman, E. Harjula, M. Jurmu, T. Karvonen *et al.*, "The many faces of edge intelligence," *IEEE Access*, vol. 10, pp. 104 769–104 782, 2022.

[66] J. Graham-Cumming, "Batteries included: how ai will transform the who and how of programming." [Online]. Available: https://blog.cloudflare.com/ai-will-transform-programming/

[67] I. Ozkaya, L. Bass, R. S. Sangwan, and R. L. Nord, "Making practical use of quality attribute information," *IEEE Software*, vol. 25, no. 2, pp. 25–33, 2008.

[68] I. Nurdiani, J. Börstler, and S. A. Fricker, "Literature review of flexibility attributes: A flexibility framework for software developing organization," *Journal of software: Evolution and Process*, vol. 30, no. 9, p. e1937, 2018.

[69] S. Bellomo, N. Ernst, R. Nord, and R. Kazman, "Toward design decisions to enable deployability: Empirical study of three projects reaching for the continuous delivery holy grail," in *Proc. 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 702–707.

[70] Z. Bauman, *Liquid modernity*. John Wiley & Sons, 2013.

[71] N. Osman, C. Sierra, J. Sabater-Mir, J. Wakeling, J. Simon, G. Origgi, and R. Casati, "LiquidPublications and its technical and legal challenges," *Intelligent Multimedia: Managing Creative Works in A Digital World*, vol. 8, 2010.

[72] F. Zambonelli and M. Viroli, "A survey on nature-inspired metaphors for pervasive service ecosystems," *International Journal of Pervasive Computing and Communications*, vol. 7, no. 3, pp. 186–204, 2011.

[73] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor *et al.*, "Design patterns from biology for distributed computing," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 1, pp. 26–66, 2006.

[74] G. D. M. Serugendo, A. Karageorgos, O. F. Rana, and F. Zambonelli, *Engineering self-organising systems: Nature-inspired approaches to software engineering*. Springer, 2004, vol. 2977.

[75] P. Marrow, "Nature-inspired computing technology and applications," *BT Technology Journal*, vol. 18, no. 4, pp. 13–23, 2000.

[76] P. Rein, S. Ramson, J. Lincke, R. Hirschfeld, and T. Pape, "Exploratory and live, programming and coding: A literature study comparing perspectives on liveness," *The Art, Science, and Engineering of Programming*, vol. 3, no. 1, 2019.

[77] S. L. Tanimoto, "Viva: A visual language for image processing," *Journal of Visual Languages & Computing*, vol. 1, no. 2, pp. 127–139, 1990.

[78] D. Ingalls, T. Felgentreff, R. Hirschfeld, R. Krahn, J. Lincke, M. Röder, A. Taivalsaari, and T. Mikkonen, "A world of active objects for work and play: the first ten years of lively," in *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2016, pp. 238–249.

[79] A. Ometov, O. L. Molua, M. Komarov, and J. Nurmi, "A survey of security in cloud, edge, and fog computing," *Sensors*, vol. 22, no. 3, 2022.