Distributed Smart Space Orchestration

Marc-Oliver Pahl
Technische Universität München
Email: pahl@net.in.tum.de

Georg Carle
Technische Universität München
Email: carle@net.in.tum.de

Gudrun Klinker
Technische Universität München
Email: klinker@in.tum.de

***Abstract*—Many networked devices that can interface their physical environments are available off-the-shelf or can be built in 2016. A comprehensive management of those Smart Devices is required to unlock the existing potential. However, the amount and heterogeneity of the devices make their management difficult. A suitable abstraction is missing. This paper identifies requirements on managing Smart Devices from diverse research fields, assesses relevant existing work, proposes a new management middleware design, and evaluates it quantitatively and qualitatively. The presented novel middleware architecture could become an enabler for a software maker culture.**

I. Introduction

In our daily lives, connected sensors and actuators are omnipresent. They detect presence or open doors while we walk through physical spaces. Connecting sensors and actuators to embedded computing systems with a network interface –enabling remote access– forms so called *Smart Devices*. The physical spaces that contain connected Smart Devices are called *Smart Space*. For implementing changing scenarios within the same Smart Space, it becomes attractive to manage Smart Devices remotely via software. In today's physical spaces functionality is often hard wired. In a software managed Smart Space, functionality could be changed more easily by running different programs. Shifting functionality from hardware into exchangeable software increases the flexibility of a system. Such a shift was the basis of computing like we know it today [1]. For describing the management of Smart Devices via software, my thesis[1] defines the term *Smart Space Orchestration* (S2O) [2].

Smart Space Orchestration is still not reality outside research laboratories in 2016. The concept was first envisioned at Xerox PARC in 1991. Weiser [11] identified three main challenges towards an implementation in the real world [11]: *networking* support is available today [12]. Smart Device *hardware* is available off-the-shelf today, or can be created using platforms such as the Arduino [13]. A so-called *hardware maker culture* emerged.

The remaining open challenge that prevents S2O from spreading into the real world concerns the *software* [6]. A suitable tooling remains missing (Sec. III) [2], [14]. My thesis [2] analyzes requirements on a suitable abstraction for enabling an S2O *software maker culture* (Sec. II). It proposes a novel middleware-based programming methodology (Sec. IV). It shows how the new abstraction can be used to implement relevant components for a Smart Space App economy [6] (Sec. V). It evaluates the middleware prototype quantitatively and the usability of the novel programming methodology qualitatively (Sec. VI).

[1]The thesis is published in [2]. Some of the thesis results are currently under peer review. Others were already published in [3]–[9]. In addition an eLearning course was developed and published during the thesis [10]

978-1-5090-0223-8/16/$31.00 © 2016 IEEE

A. Research Questions

The overall research question of the thesis [2] is, *how to design a programming abstraction that can enable S2O in the real world*. This question can be concretized:

1) Which is a suitable virtual representation of S2O data?
2) Which is a suitable programming abstraction for services interfacing Smart Devices?
3) Which is a suitable programming abstraction for other services (e.g. orchestration Apps)?
4) Which other components are required for a software maker culture for Smart Spaces.

B. Contributions

Major contributions of the presented research are:

- A novel middleware architecture that structures and facilitates the implementation of management in distributed, heterogeneous, dynamically changing environments. The middleware is called Virtual State Layer (VSL).
- An easy-to-use, dynamically-extensible, self-managing ontology for modeling Smart Space data [4].
- A novel data-centric service coupling concept [3].
- A security-by-design architecture for future Smart Spaces.
- A hierarchical management architecture for distributed services within future Smart Spaces.
- A requirements analysis considering a wide range of research fields.
- A broad survey of existing middleware for S2O.

C. Publications

The thesis is published in [2]. The information model and the crowdsourced data modeling are published in [4]. The methodology for creating of a Service Oriented Architecture (SOA) using data-centric services coupling is published in [3]. Components towards an App economy for Smart Spaces are presented in [6]. Future application scenarios for the technology are described in [5]. An overview on the Virtual State Layer (VSL) middleware is given in [7]. The general concept was first introduced in [8]. Some security considerations are published in [9].

The software produced during this thesis is available open source [15]. It is under continuous development. A new release is planned for summer 2016. It significantly improves the performance (Sec. VI). The software is continuously tested in the education of Computer Science students (CS) at Technische Universität München (TUM) [10].

The relevance of S2O on technological, economic, and societal aspects requires including it into the curriculum at universities. Consequently, an S2O eLearning course was developed as part of the thesis [10]. Since summer term 2013 it is running at TUM with around 20 CS students each semester.

D. Relevance of the Work

The thesis proposes a novel management framework architecture for dynamically changing, distributed, and heterogeneous environments – the VSL. Diverse management scenarios can be mapped to a subset of the corresponding requirements. As a result, this research can be applied for diverse management scenarios. Applications from network management (NM) include managing services for Network Function Virtualization (NFV), algorithms in Software Defined Networking (SDN), and the IoT [16]. Beyond the self-purpose of service and device management, the research results are applicable in domains such as Industry 4.0 or Smart Buildings. Concrete examples are managing production lines to dynamically adapt for each product, saving energy in energy-intense systems, or enabling new applications such as Ambient Assisted Living (AAL) [2].

The presented research provides a novel data-centric management framework that could become the base for a software maker culture for Smart Spaces [2]. By establishing a data-centric Service Oriented Architecture (SOA), the VSL enables a new way of developing software. It allows a new engineering practice based on modularization, reuse, and agile development [2], [3]. This role requires the VSL to provide desired properties *by-design* such as security, scalability, performance, dependability, and usability. Though the prototype presented in [2] shows the feasibility of all described concepts, the research and development towards a better understanding of its challenges and opportunities is ongoing[2].

The proposed management approach targets enabling a real world implementation of S2O. S2O has strong implications not only on technology but also on society [2], [14], [17]: similar to changing the functionality of a smartphone by installing Apps today, it may become possible to change the functionality of a *Smart Space* by installing Smart Space Apps in the future [2], [6]. As a long term vision, future Smart Spaces may behave like a holodeck in Star Trek, transforming into diverse functional spaces such as a living room, or a kitchen only by running a different App. The computing paradigm behind S2O, Pervasive Computing, has the potential to become the third computing revolution after the introduction of Mainframes (1960s) and the introduction of Personal Computers (1980s), fundamentally changing our lives [14]. The thesis fosters weaving computing "into the fabric of everyday life" [11].

II. Requirements Analysis

Smart Space Orchestration (S2O) falls into different research areas. The overall domain is Ubiquitous Computing research [11]; more concrete Pervasive Computing [18]. There are several relevant sub domains: The Internet of Things (IoT) [16] focuses on the technical aspects of connecting Smart Devices. Cyber Physical Systems (CPS) focus on managing Smart Devices via software. Ambient Intelligence (AMI) focuses on applications interacting with humans. In a spectrum between hardware (HW) and software (SW), S2O is situated between CPS and AMI: HW — IoT — CPS — S2O — AMI — SW. It connects CPS and provides the base for AMI [2].

The analysis in [2] reflects the broad range of affected domains. It considers hardware, protocol bridging [19], standardization [20], distributed computing, human psychology, Artificial Intelligence (AI), autonomous computing [21], software architecture, and semantic modeling [22]. Major analysis results are summarized as requirements here.

Bottom-up, the analysis starts with Smart Devices (see (1) in Fig. 1). Their management interfaces are heterogeneous for diverse reasons [2]. Policies could eliminate some reasons. However, technical differences result in heterogeneous communication protocols that must be overcome in software (R1). Overcoming heterogeneity on any layer of abstraction requires standardization. Traditional standardization methods such as ISO standardization [20] are unsuitable for the speed of innovation and the amount of stakeholders in S2O scenarios [2], [4]. S2O requires new standardization methods (R2).
The bottom-up analysis concludes with identifying middleware as a suitable concept to overcome the heterogeneity of distributed systems as found in future Smart Spaces [2], [23]. While a middleware can cover many aspects of heterogeneity, it cannot provide access transparency to the diverse Smart Devices. Access transparency is required (R3) for enabling portable S2O Apps.
A major problem of existing middleware for S2O (Sec. III) is that it provides domain-specific functionality. Such functionality limits the possible applications that can run on top of a middleware. This limitation can be problematic as it restricts the creativity of developers [17]. More severe, it prevents the implementation of unsupported S2O scenarios. A suitable abstraction for S2O must not have domain-specific functionality built-in (R4).
However, domain-specific functionality significantly facilitates the implementation of workflows for the corresponding domain. Therefore, the dynamic adding of domain-specific functionality should be supported (R4a).
A fundamental problem of existing S2O middleware is the missing support for modes of referential and temporal coupling. The variety of S2O use cases requires all service coupling modes to be supported by a suitable middleware (R5).

Top-down, the analysis starts with human psychology. A descriptive abstraction is more intuitive to use than a procedural one (R6) [24], [25].
In AI, the human cognition is modeled using the Believe-Desire-Intention (BDI) model [26]. Simple implementations of the BDI model are Event-Condition-Action (ECA) rules. ECA rules are intuitive to create for humans [24], [25]. But the expressiveness of ECA rules is limited. They can only process existing data, resulting in reactive behavior. A desired active behavior would require function calls that can not be made in ECA rules [27]. These limitations make ECA rules unsuitable for implementing S2O workflows. However, for intuitive usability a suitable abstraction for S2O should be as simple to use as ECA rules are (R7).
Autonomous computing helps making complex systems manageable [28]. A typical implementation is a Monitor-Analyze-Plan-Execute (MAPE) cycle that can be extended with state (knowledge; MAPE-K) [21]. A suitable abstraction for S2O should provide and support autonomous functionality (R8).
Decomposing complex tasks into simpler modules is intuitive [24], [25]. A software architecture that supports modularization is a Service Oriented Architecture (SOA) [29]. A suitable abstraction for S2O should foster service-oriented software development (R9).
The common denominator of S2O Apps is storing and ex-

[2]http://s2o.net.in.tum.de/, http://www.ds2os.org/

changing data. The virtual representation of real world data is called context [22]. A suitable abstraction for S2O must support the handling of context (R10).

In a concluding case study [2], the smartphone App economy is analyzed. It shows what is required to enable a software maker culture for a new field of technology: suitable abstractions in the form of 1) runtime environments, 2) Operating Systems (OS), and 3) programming frameworks that hide the heterogeneity of the underlying systems [2].
1) A suitable system for providing execution portability (see R3) exists with Java. 2) An OS for a distributed system is called middleware [23]. Therefore a suitable abstraction for S2O is a middleware (R11). 3) Regarding framework support, Smart Devices within a Smart Space are significantly more heterogeneous and dynamic than the peripherals in a smartphone [2]. Therefore an S2O middleware must be dynamically extensible with device drivers (R12).
Concerning non-functional requirements, it is required that a suitable middleware framework for S2O is secure, resilient, and performant (R13).

III. Related Work

In the relevant domains (Sec. II), diverse systems have been developed. The communities that work for the longest time on S2O are the Ubiquitous Computing (UbiComp) and the Pervasive Computing (PerCom) community. Both focus on applications [2], [30]. On the other side of the spectrum is the Network Management (NM: NOMS, IM) community. It has a long experience in systematic management system design.
Currently, the Ubiquitous Computing community has the problem that it lacks research in systematic foundations for S2O. The NM community on the other hand only opens slowly towards new application domains such as the IoT[3]. This work follows a novel approach by combining the usability and flexibility of PerCom with the systematic foundations of NM.

Existing UbiComp/ PerCom middleware is typically tailored for its corresponding research project [2]. Many middleware designs emerged since 1991. Several peer-reviewed surveys including [31]–[35] are used to select a representative sample. The selected solutions can be classified into middleware that focuses on service management [36]–[40], and middleware that focuses on data management [41]–[45].

The requirements from Sec. II are used to summarize the analysis results. The parenthesis contain the corresponding requirement identifier R0 and the assessment $\in \{+, o, -\}$. Most solutions follow a vertical design. Device drivers are part of the middleware. This is unsuitable for dynamically integrating the diverse S2O Smart Devices (R1-). No solution uses explicit management data models as known from NM. This makes standardization impossible (R2-). Using context as interface between services provides access transparency (R3+). Being built for specific scenarios, most middleware supports only domain-specific workflows and inhibits others (R4-). This leads to *middleware silos* [2]: Apps that run within one middleware cannot interact with Apps (and Smart Devices) connected to another middleware. Middleware silos are a major inhibitor for S2O today. Existing middleware provides good solutions for its specific domains. But different middleware is not interoperable [2]. No assessed solution supports all combinations of referential and temporal couplings [3] (R5-). This prevents the use of existing middleware as general enabler for S2O.

Context based interfaces are typically declarative (R6+). Only one solution [44] considers simplicity (R7-). With their strong encapsulation of services behind context, many solutions support Autonomous Computing paradigms (R9+). Many of the solutions are based on context processing (R10+). All solutions but [38], [44] are implemented as a middleware (R11+). Dynamically integrating new devices is typically not possible (R12-). Security is widely not considered in the assessed solutions. Three architectures are used: 1) peer-to-peer, 2) centralized, and 3) distributed with centralized components such as directories. Depending on those architectures, the expected resilience and the performance vary (R13o).

On the other end of the spectrum, a typical NM methodology consists of agents that offer descriptive interfaces to management data on managed devices [46]. The abstract interfaces are the declarative data models [47] (R1+, R3+, R6+, R10+). Those models are typically standardized but the process and its mechanisms are not suitable for the high amount and heterogeneity of emerging Smart Devices and their manufacturers [4] (R2-). Typically NM frameworks do not include support for specific functionality (R4+), making them suitable as a general enabler. However, typically not all four coupling modes [2], [3] are supported (R5-). ECA rules can be applied but their limitations are not overcome (R7o). Management agents are typically working autonomously (R8+). Service-orientation is typically not supported as scenarios are implemented vertically: managers access agents to implement certain management goals (R9-). Management frameworks fulfill parts of the properties of a middleware (R11o). The extensibility with drivers is hidden behind the agents' abstract interfaces (R12+). The non-functional properties vary between different implementations (R13o).

IV. The Virtual State Layer (VSL)

The Virtual State Layer (VSL) middleware (R11) is designed using the requirements from Sec. II [2]. The VSL is the core of the Distributed Smart Space Orchestration System (DS2OS) framework [15]. DS2OS provides additional functionality such as a directory for data models, a service management framework, and an App store.

The VSL is a self-organizing unstructured peer-to-peer system that manages context data (R10). An autonomous peer is called Knowledge Agent (KA) (R8). A VSL overlay is shown as layer (3) in Fig. 1. From a NM perspective, the VSL extracts the agent functionality from the managed devices into a separate layer (R3, R6). The VSL is fully self-managing, enabling the desired security-by-design, resilience, and performance properties (R13) [2].

Layer (2a) shows services synchronizing state between Smart Devices (1) and their virtual representation in the VSL (3). Layer (2b) shows services implementing other functionality such as reasoning, orchestration or user interfaces [2]. Each service (2a, 2b) connects to one KA for storing data

[3]There was a special track "On the Management of the Internet of Things" organized by Jürgen Schönwälder, Anuj Sehgal, and Mauro Tortonesi at NOMS 2014.

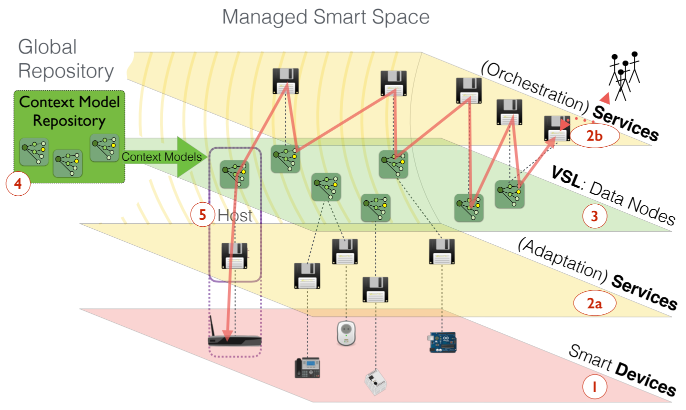

Fig. 1: VSL architecture.

and for retrieving data from other services (R6, R10). The Context Model Repository (CMR; 4) is the directory of VSL data models.

A. Crowdsourced Data Modeling

The VSL implements the blackboard communication pattern. Unlike a classic tuple space, it uses explicit data models with unique identifiers for tuple discovery.

Each service connects to one KA at start-up time for getting access to its own state and the states of any other service connected to the VSL overlay. All services such as Smart Device-connecting Apps get the same support by the VSL. This unifies the implementation of diverse services, facilitating the service development (R1, R12). The VSL fosters a separation of service logic (App) and data (KA) [48]. This separation facilitates the development of services.

As identified in Sec. II (R2, R3) and applied by the NM community, structuring management data by introducing models enables service portability. The context models of the VSL are stored in the global CMR directory (4).
The CMR contains a dynamically extensible semantic type system. The directory stores tree data structures that are identified by unique *model identifiers* (modelID) (R3) [2], [4]. Context models can inherit from other context models by referring to their modelID as VSL data type. Combined with an easy-to-understand XML notation [4], inheritance significantly facilitates the model creation (Sec. VI). By providing statistics about the use of certain context models, the CMR implements a crowdsourced standardization (R2). This standardization is the base for providing access transparency and portability via the VSL (R3).

For each service exactly one context model is instantiated in its KA. Services communicate by accessing each others data in the VSL. As a result, context models serve as abstract interface for their services (see Fig. 1) [3]. Having all data stored and served in the VSL makes data independent of service implementations enhancing the resilience of the Smart Space computing back-end (R13).

For discovering specific service data in the VSL, the VSL type identifier (modelIDs) from the CMR are used. All KAs autonomously synchronize their directories about where which context data is available. The incrementally replicated directories increase the resilience and performance of the KA overlay (R13). The resulting local type-searches are fast.
Only the type-search is built into the VSL (R4). The special VSL service coupling (Sec. IV-B) enables adding additional semantic search-providers at run-time (R4a). Such semantic search-providers enable a novel kind of dynamically extensible ontology [22]. This extensible ontology is a novel answer to the need for describing the dynamically changing management parameters of the IoT [2], [4].

B. Direct Coupling over Descriptive Interfaces

The VSL allows Apps to couple asynchronously and synchronously (R9). It implements all four modes of referential and temporal couplings (R4) [3]. Apps couple asynchronously by querying data stored in the KA overlay. For the synchronous coupling, there are two modes. The first mode is via subscriptions on regular VSL data nodes. A service can subscribe any data node it is authorized to (R13). On changes, the KA sends a notification. The line between the layers (2b) and (3) in Fig. 1 illustrates such notifications between different services. The second mode for direct coupling is via so called *Virtual Nodes*. Instead of serving the data request itself, a KA triggers a previously registered Virtual Node callback into a service. Whether the KA (regular node) or a service (Virtual Node) is serving a request is transparent for the requesting service (R3).

The VSL allows access to sub nodes of Virtual Nodes. The emerging context nodes are called *Virtual Subtrees*. They allow passing parameters by accessing virtual VSL data [2], [3]. Virtual Subtrees enable a transparent extension of the VSL functionality at run time (R4a). Virtual Subtrees overcome the restrictions of ECA rules (R7). This enables the implementation of complex workflows with simple ECA rules.
Virtual nodes enable a direct coupling of services via a descriptive data structure. As a result, the VSL supports service-oriented software development (R9). Complex functionality can be decomposed into better manageable, reusable modules. Decomposition and reuse facilitate the implementation of complex S2O workflows.

C. The VSL as Named Data Network

The VSL implements a self-organizing Named Data Network (NDN) [49]. It is therefore also an interesting architecture for investigating properties of a possible future Internet architecture. The VSL proves that NDN principles are beneficial for implementing management functionality.

V. Applications

In [2], it is shown how the described VSL programming abstraction (Sec. IV) enables and significantly facilitates the implementation of site-local service management. It is shown how the infrastructure for an App economy for Smart Spaces could be designed around the Smart Space App store (Smart Space Store; S2Store) to meet properties such as security-by-design (R13).

Reference implementations for relevant S2O service classes are presented. The examples include a generic user interface and an autonomous Smart Device adaptation service. The latter acts as device driver (R1, R8, R12; (1) and (2) in figure 1) [2]. The applications show the power and suitability of the VSL as an enabler for real world S2O.

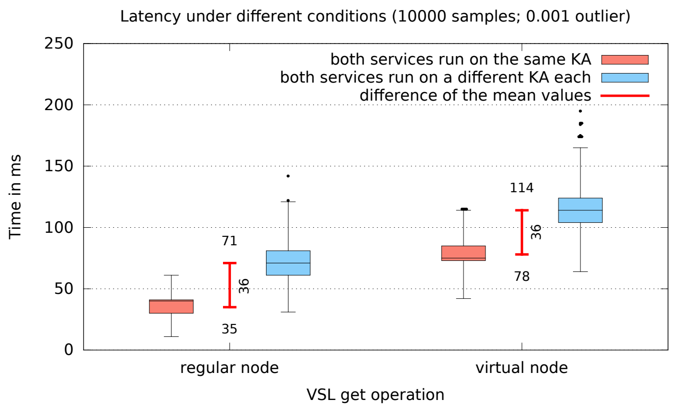

Fig. 2: Performance of the VSL prototype.

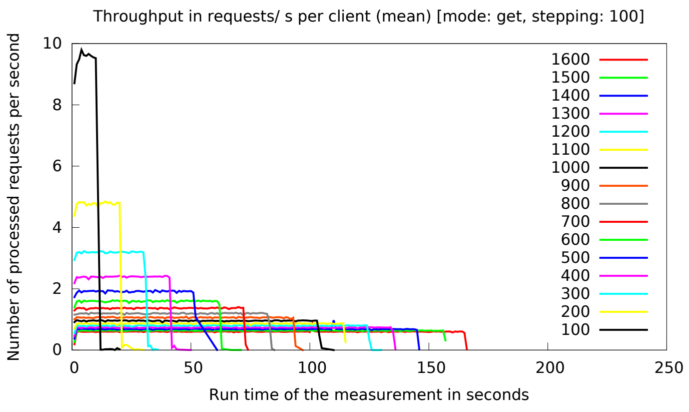

Fig. 3: Scalability of the VSL prototype.

VI. Evaluation

The VSL prototype implementation has been evaluated quantitatively and qualitatively. As described in Sec. I-C, an S2O eLearning course was created. In this course students continuously assess the usability of the VSL and reproduce the following measurement results. The results shown here are the original results from early 2014 [2].
All measurement results were obtained within a testbed of 36 Intel i5 computers with SSD and 4GB of main memory. All computers were connected via GBit Ethernet. Each conducted performance and scalability experiment involves one or multiple measuring services and a measurement probe service.

The measurement results in Fig. 2 were obtained by executing a *get* request on another service's VSL node 10000 times. The latencies for regular node and Virtual Node accesses are shown for both services running on the same KA (left), and both services running on two distributed KAs (right).
The lower candle of each box plot marks the 0.25 quantile. The line in the box is the median (0.5 quantile). The upper bound of the box is the 0.75 quantile. 0.1% of the samples are considered outliers and plotted as dots. Between the two boxplots the mean values and their difference are shown.
The plot shows little variation in the samples. The mean delay for a *get* operation on a regular node is 35ms. Accessing a remote KA adds about 36ms of delay. In the current implementation all KAs are interconnected. One hop is thereby the maximum observable coupling delay (see Fig. 1).
The implementation of Virtual Node callbacks currently uses a single socket and several lookups. This explains why the measured delay is more than double of that of a regular VSL node access. The measured values for the *set* operation are very similar to the shown ones of the *get* operation [2].
The delays are relevant for knowing how many services can be coupled over the VSL within a certain time. A delay up to 160ms is considered not noticeable by humans [50]. The authors of [51] found out that up to 500ms are okay for a telesurgery, which can be considered as a time critical application. Consequently, several services can be coupled over the VSL while preserving an interactive experience. Many S2O use cases are not highly time critical. Correspondingly the amount of services that can be coupled in a VSL SOA increases within the required time boundaries.

Fig. 3 shows the throughput per second per service over time. A single remote service was accessed via *get* on a regular node from 100 up to 1600 distributed services in parallel (see legend). In each experiment round, all participating services sent 100 consecutive requests in parallel. The plotted mean shows that the VSL KAs scale well. All requests got answered and the time of the entire experiment increased linearly with the amount of parallel requests. For up to 400 parallel requests the delay for a single remote service coupling is <500ms (interactive). The measured values for the *set* operation are close to the shown ones of the *get* operation [2]. The access to Virtual Nodes is slower as can be seen in Fig. 2.

In addition to the system measurements, a user study was conducted. The survey shows that the CS students at TUM were able to adapt to the VSL programming abstraction quickly and liked working with it [2]. In addition to the survey, the time for implementing given S2O scenarios was measured. All participants implemented a given complex workflow in less than 20h. This value is good compared to an experiment conducted in [45] where a comparably complex tasks took experienced programmers about 120h.

VII. Conclusion

This paper introduced the VSL middleware design. The VSL combines the systematic design from NM with the dynamic adaptation required for managing the diversity and dynamism found in the PerCom domain. Answering the research questions from Sec. I-A: 1) An easy-to-use, dynamically extensible virtual representation for S2O data was introduced (Sec. IV-A). 2), 3) A data-centric programming abstraction for S2O was introduced (Sec. IV). It supports all service categories equally. This was demonstrated with different applications in Sec. V and Sec. VI. 4) The service management framework and the S2Store illustrate how a software maker culture for Smart Spaces could technically be implemented (Sec. V).

It may take some time until holodecks become reality (Sec. I-D). In the meantime the presented research can enable the implementation of novel applications such as production lines that individually adapt for each product, automated energy saving, or Ambient Assisted Living (AAL).
Enabling a software maker culture is an important step towards real world Smart Spaces [52]. New possibilities bring new challenges (Sec. I-D). Challenges worth being considered can be found in science fiction [53]. A societal challenges that is closer to today's reality is technology constraining the freedom of people to implement workflows their preferred way [17].

References

[1] J. von Neumann, “First draft of a report on the EDVAC,” *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993.

[2] M.-O. Pahl, “Distributed Smart Space Orchestration,” Ph.D. dissertation, Technische Universität München, München, 2014. [Online]. Available: http://pahl.de/download/publications/dissertationPahl2014.pdf

[3] ——, “Data-Centric Service-Oriented Management of Things,” in *IM 2015*, Ottawa, Canada, 2015, pp. 484–490.

[4] M.-O. Pahl and G. Carle, “Crowdsourced Context-Modeling as Key to Future Smart Spaces,” in *NOMS 2014*, 2014, pp. 1–8.

[5] M.-O. Pahl, H. Niedermayer, H. Kinkelin, and G. Carle, “Enabling Sustainable Smart Neighborhoods,” in *SustainIT*, Palermo, Italy, 2013.

[6] M.-O. Pahl and G. Carle, “Taking Smart Space Users into the Development Loop: An Architecture for Community Based Software Development for Smart Spaces,” in *UbiComp*. New York, NY, USA: ACM, 2013, pp. 793–800.

[7] ——, “The Missing Layer - Virtualizing Smart Spaces,” in *PerCom 2013 adjunct*, San Diego, USA, 2013, pp. 139–144.

[8] M.-O. Pahl, C. Niedermeier, M. Schuster, A. Müller, and G. Carle, “Knowledge-based middleware for future home networks,” in *Wireless days*. Paris, France: IEEE, 2009.

[9] H. Niedermayer, R. Holz, M.-O. Pahl, and G. Carle, “On using home networks and cloud computing for a future internet of things,” in *FIS*. Springer, 2009.

[10] M.-O. Pahl. (2014) Smart Space Orchestration Course Material. [Online]. Available: http://pahl.de/download/dissertation/ds2os.lab/

[11] M. Weiser, “The Computer for the 21st Century,” *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[12] I. T. U. ITU, “Global ICT developments, 2001-2015,” Tech. Rep., 2015.

[13] M. Banzi, “Getting Started with Arduino, Ill edition,” *Getting Started with Arduino, Ist edition*, Oct. 2008.

[14] G. D. Abowd, “What next, ubicomp?: celebrating an intellectual disappearing act,” in *UbiComp*. ACM, 2012.

[15] System, Distributed Smart Space Orchestration. http://www.ds2os.org/. [Online]. Available: http://www.ds2os.org/

[16] I. T. U. ITU, “ITU Report ”The Internet of Things”,” Geneva, Tech. Rep. 7, 2005.

[17] P. Dourish and S. Mainwaring, “Ubicomp’s Colonial Impulse,” *UbiComp*, 2012.

[18] K. Lyytinen and Y. Yoo, “Issues and Challenges in Ubiquitous Computing ,” *Communications of the ACM*, vol. 45, no. 12, 2002.

[19] H. Zimmermann, “OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection,” *Communications, IEEE Transactions on*, vol. 28, no. 4, pp. 425–432, 1980.

[20] ISO and IEC, “ISO/IEC Directives, Part 1,” ISO/IEC, 2012.

[21] J. O. Kephart and D. M. C. Chess, “The vision of autonomic computing,” *Computer*, vol. 36, no. 1, 2003.

[22] U. Aßmann, S. Zschaler, and G. Wagner, “Ontologies, Meta-models, and the Model-Driven Paradigm,” in *Ontologies for Software Engineering and Technology*, C. Calero, F. Ruiz, and M. Piattini, Eds. Berlin Heidelberg: Springer, 2006, pp. 249–273.

[23] P. A. Bernstein, “Middleware: a model for distributed system services,” *Communications of the ACM*, vol. 39, no. 2, pp. 86–98, 1996.

[24] P. N. Johnson-Laird, *The Computer and the Mind: An Introduction to Cognitive Science*. London: Fontana Press, 1993.

[25] F. Détienne, “Software Design - Cognitive Aspects,” 2002.

[26] M. P. Georgeff, B. Pell, M. E. Pollack, M. Tambe, and M. Wooldridge, “The Belief-Desire-Intention Model of Agency,” in *ATAL 98*. Springer-Verlag, Jul. 1998.

[27] M. J. Wooldridge, “An Introduction to MultiAgent Systems,” 2009.

[28] Z. Movahedi, M. Ayari, R. Langar, and G. Pujolle, “A Survey of Autonomic Network Architectures and Evaluation Criteria,” *IEEE Communications Surveys & Tutorials*, vol. 14, no. 2, pp. 464–490, 2012.

[29] T. Erl, *Service-Oriented Architecture*, ser. Concepts, Technology, and Design. Prentice Hall, 2005.

[30] Y. Liu, J. Goncalves, D. Ferreira, S. Hosio, and V. Kostakos, “Identity crisis of ubicomp?: mapping 15 years of the field’s development and paradigm change,” in *UbiComp Adjunct*. ACM, Sep. 2014.

[31] K. Henricksen, J. Indulska, and T. McFadden, “Middleware for Distributed Context-Aware Systems ,” *DOA*, 2005.

[32] C. Endres, A. Butz, and A. MacWilliams, “A survey of software infrastructures and frameworks for ubiquitous computing,” *Mobile Information Systems*, 2005.

[33] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini, “A survey of context data distribution for mobile ubiquitous systems,” *Computing Surveys*, vol. 44, no. 4, 2012.

[34] V. Raychoudhury, J. Cao, M. Kumar, and D. Zhang, “Middleware for pervasive computing: A survey,” *Pervasive and Mobile Computing*, 2012.

[35] M. Knappmeyer, S. L. Kiani, E. S. Reetz, N. Baker, and R. Tonjes, “Survey of Context Provisioning Middleware,” *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 3, pp. 1492–1519, 2013.

[36] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, “Gaia,” *ACM SIGMOBILE CCR*, vol. 6, no. 4, pp. 65–67, 2002.

[37] J. P. Sousa and D. Garlan, “The Aura Software Architecture: an Infrastructure for Ubiquitous Computing,” 2003.

[38] C. Dixon, R. Mahajan, S. Agarwal, A. J. B. Brush, B. Lee, S. Saroiu, and P. Bahl, “An operating system for the home,” in *NSDI*, 2012.

[39] C.-F. Sørensen, M. Wu, T. Sivaharan, G. S. Blair, P. Okanda, A. Friday, and H. Duran-Limon, “A context-aware middleware for applications in mobile Ad Hoc environments,” in *MPAC*. ACM, 2004.

[40] S. Dawson-Haggerty, A. Krioukov, J. Taneja, S. Karandikar, G. Fierro, N. Kitaev, and D. Culler, “BOSS: Building Operating System Services,” *NSDI*, 2013.

[41] A. K. Dey, D. Salber, M. Futakawa, and G. D. Abowd, “An Architecture to Support Context-Aware Applications,” *UIST*, 1999.

[42] T. Gu, H. K. Pung, and D. Q. Zhang, “A service-oriented middleware for building context-aware services,” *Journal of Network and Computer Applications*, vol. 28, no. 1, pp. 1–18, 2005.

[43] J. E. Bardram, “The Java Context Awareness Framework (JCAF) – A Service Infrastructure and Programming Framework for Context-Aware Applications,” in *Pervasive Computing*. Berlin, Heidelberg: Springer, 2005, pp. 98–115.

[44] B. Guo, D. Zhang, and M. Imai, “Toward a cooperative programming framework for context-aware applications,” *Personal and ubiquitous computing*, vol. 15, no. 3, pp. 221–233, 2010.

[45] R. Grimm, “One.world: Experiences with a Pervasive Computing Architecture ,” *Pervasive Computing*, 2004.

[46] J. Schonwalder, “Protocol-Independent Data Modeling: Lessons Learned from the SMIng Project,” *IEEE Communications Magazine*, vol. 46, no. 5, pp. 148–153, 2008.

[47] A. Pras and J. Schoenwaelder, “On the Difference between Information Models and Data Models,” IETF, Tech. Rep., 2003.

[48] R. Grimm, J. Davis, B. Hendrickson, E. Lemar, A. Macbeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall, “Systems Directions for Pervasive Computing,” in *Workshop on Hot Topics in Operating Systems*, 2001, pp. 147–151.

[49] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, “Named data networking,” *SIGCOMM CCR*, vol. 44, no. 3, 2014.

[50] B. Shneiderman and C. Plaisant, *Designing the User Interface*, 4th ed. Pearson Education, 2005.

[51] M. D. Fabrlzio, B. R. Lee, D. Y. Chan, D. Stoianovici, T. W. Jarrett, C. Yang, and L. R. Kavoussi, “Effect of Time Delay on Surgical Performance During Telesurgical Manipulation,” *Journal of Endourology*, vol. 14, no. 2, pp. 133–138, 2000.

[52] Gartner. (2015) Gartner Says 6.4 Billion Connected ”Things” Will Be in Use in 2016, Up 30 Percent From 2015. [Online]. Available: http://www.gartner.com/newsroom/id/3165317

[53] S. Kubrick. (1968) 2001: A Space Odyssey.