Crowdsourced Context-Modeling as Key to Future Smart Spaces

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

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

***Abstract*—Managing smart spaces with software requires the acquisition and processing of context information about a space. To be usable for exchanging information, a context representation has to be structured with a context model. Existing context-modeling techniques usually require experts and lack support for collaborative distributed creation, which prevents a crowdsourced development in a distributed collaborative way by non-experts. To facilitate context modeling, this paper presents a hybrid meta model that combines features from key-value, markup, object oriented, and ontology based context-modeling approaches. An architecture is introduced that allows the dynamic collaborative extension and crowdsourced convergence of context models.**

I. Introduction

The term Internet of Things (IoT) describes the technical realization of ubiquitous computing [1] from a device-centric point of view. A physical environment that is enriched with *smart devices* is called *smart space* in this paper. *Smart devices* are embedded systems that are remotely controllable over a network interface and that contain sensors and actuators to interface their physical environment. *Smart devices* can have diverse purposes, as ubiquitous computing has diverse applications.

The execution of services that implement pervasive computing scenarios is called *Smart Space Orchestration*. The term *context* describes all information that is relevant for a service to fulfill its orchestration goal. *Context* is represented as so-called *virtual objects*. A *context model* defines the structure of a *virtual object*. A *meta model* defines the language that can be used to create *context models*.

Ubiquitous computing is still not reality in 2014 [2]. While its mobile computing part [3] is very successful [4], [5], the pervasive computing part still struggles [6], [2], [7], [8]. That the success of mobile computing came earlier is not surprising as its technology to connect devices, and to offer remote (cloud-) resources to smart devices for realizing complex orchestration tasks [9] is needed for realizing pervasive computing [10].

Many *smart devices* are available off-the-shelf in 2014. Often they are heterogeneous in the offered communication protocols [11], [2], [6], [12]. Available smart devices are developed for different domains, by different vendors, and have different targeted installation environments. A *smart device* that has the function of a fire alarm will probably expose the state of its sensors in a different way than a device with the same sensors in another functional domain. A battery powered *smart device* with a radio interface is likely to use a different communication protocol than a *smart device* that has a powerline based energy supply and communication interface.

978-1-4799-0913-1/14/$31.00

Pervasive computing scenarios typically bridge multiple formerly separate application domains such as office automation and building control. To do so, software is needed. The orchestration of diverse entities via software is called *software orchestration*. *Software orchestration* is complex in 2014 as interfacing distributed heterogeneous devices introduces complexity.

Middleware can help to overcome distribution complexity [13]. Context-provisioning middleware provides functionality to manage *context*. As pervasive computing is inherently context-aware, context-provisioning middleware facilitates the creation of *Smart Space Orchestration* services [8].

Context-provisioning middleware allows implementing services that use *context* as interface for communication. Services act as *context producer* and *context consumer* and interact over their exchanged *context*. A system that allows service interaction over *context* was presented in [14]. It is a context-provisioning middleware with the name Virtual State Layer (VSL, Sec. IV). The VSL is used as base for this work though any context-provisioning middleware that supports *context models* could be used.

When using *context* for information exchange between services, a fundamental problem is the standardization of *context models*. If each entity such as different lamps has a different context model, *portability* cannot be realized. The term *portability* is used to describe the independence of a service implementation from specific device interfaces. Introducing abstract interfaces typically provides it. In case each device brings its own abstract interface, heterogeneity of the device layer is only lifted on a higher level of abstraction. Therefore standardization of abstract interfaces is needed.

It is desired that *abstract interfaces* for a certain type of functionality, e.g. lamp, are standardized. As described in Sec. IV, the VSL uses *context* as abstract interface to services. The standardization of *abstract service interfaces* is therefore identical to the standardization of *context models*.

A problem with established standardization processes is that they take time, and they do not scale with the diversity of pervasive computing scenarios and devices [15]. Therefore this work proposes a novel mechanism for collaborative user-based, crowdsourced standardization of *context models*. It consists of:

- a simple-to-use dynamically-extensible *meta model*

that is based on hierarchically structured typed tuples, multi-inheritance, and composition (Sec. V-A)
- a crowdsourced *context model* creation and convergence mechanism (Sec. VI).

After the related work (Sec. II) requirements on context-modeling for future *smart spaces* are analyzed. Then the VSL is briefly introduced for understanding the practical usability of the proposed approach (Sec. IV). Next, the dynamically extensible *context modeling* approach is motivated with state of the art context-modeling techniques and presented in Sec. V. The VSL *context model* is the basis for the proposed crowdsourced *context modeling*, and *convergence* that are introduced in Sec. VI. Finally the concept is evaluated with a user study in Sec. VI-C.

II. Related Work

Defining *context models* for managed elements is a well known task from network management [16], [17]. A typical way to find common *context models* (e.g. [16]) for network elements is standardization or de-facto standardization by major vendors introducing their *context models* and other vendors adapting them. A fundamental difference between network elements and *smart devices* is that the diversity in functionality and vendors is much higher for *smart devices*. This makes applying the mechanisms used for standardizing models for network elements for standardizing *context models* of *smart devices* difficult.

The proposal of this paper is a collaborative creation and sharing of a Smart Space *context models* over a *Context Model Repository* (CMR). Using a central CMR automatically realizes an ontology for Smart Spaces as it contains the *context models* that are used to describe the entities of *smart spaces* (Sec. V-A). The closest related work is about collaborative editing and convergence of ontologies which fulfills the collaboration aspect but requires experts and does not support dynamic extensibility (see Sec. V).

In [18], the authors present different concepts to create context models, including a collaborative one. This collaborative approach requires manual arbitration, leading to scalability problems. The presented collaboration methods could well be used as supportive information for context model builders in our approach.

In [19], the authors propose consensus-building mechanisms for collaborative ontology building. Their approach is based on offline discussions and does therefore neither scale nor work efficiently with worldwide distributed participants. The presented supporting methodologies for finding a consensus is nevertheless interesting as guidelines for the manual model rating in our rating approach (Sec. VI-B).

The authors of [20] present a an entirely online-based process for ontology building and convergence that is implemented on an extended wiki. Like the previous approaches, the creation of an ontology needs technical experts, and the convergence process is human-based.

III. Requirements

Smartphones are a driving force behind the success of mobile computing [4], [5]. The App Store with its App distribution mechanism allows deploying software to smartphones [21]. This realizes *software orchestration* in the smartphone environment as the use of a smartphone can be changed by software without changing the hardware [22]. The introduction of an App Store enabled distributed user-based, crowdsourced, development of software. Such a development is also desired for *software orchestrated smart spaces* [2].

A key difference between smartphones and *smart spaces* is that smartphones have a limited and well defined set of sensors and actuators that is supported by the operating system. The limited amount allows to fix the Application Programming Interfaces (API) for smartphone peripherals. They can be supported by the operating system (OS) [23], [24].

The approach of implementing *smart device* adaptation functionality in each orchestration service does not scale with the diversity of pervasive computing scenarios and the diversity of *smart devices* [2], [25]. There is consensus [14], [26], [27] that the adaptation functionality between proprietary device protocols and standard interface must be implemented in specific software adaptation services, and should not be part of the *smart space* operating system kernel [28]. Different groups are currently working on solutions for an operating system for *smart spaces* that allows to interface heterogeneous *smart devices* via standardized interfaces [14], [26], [27].

A problem that emerges now is that different abstract interfaces for different *smart devices* with similar functionality prevent portability [27]. The VSL context-provisioning middleware that is used as base for this work uses *context models* as abstract interfaces between services. The diversity of *smart devices* requires a large amount of *context models* to be created for the VSL.

The proposal of this paper is to establish a process for distributed collaborative, crowdsourced development of *context models*. This leads to the requirements:
(1) A *context model* for smart space orchestration must be *simple* enough to create for non-experts;
(2) Methods for *crowdsourced development* of *context models* are required.

With crowdsourced *context model* creation the problem arises that different developers may create concurring models for the same *smart device* functionality. This is not desired as supporting diverse interfaces makes orchestration services that realize pervasive computing scenarios complex.

Existing standardization approaches are likely not to be sufficient (see Sec. II) for the huge amount of devices that need interfaces to be usable in future *smart space orchestration* workflows. Additionally the typical duration of standardization processes could be a decisive drawback for the emergence of *smart spaces* in the real world as it delays the creation of *smart space* services which in turn prevents customers from experiencing *smart spaces* and investing in the technology [29], [30].

To implement *portability* an equivalent to standardization is needed:
(3) Methods for *converging context models* are needed that scale with the required amount of *context models* for *Smart Space Orchestration*.

IV. The Virtual State Layer (VSL) Context-Provisioning Middleware

To implement pervasive computing work flows, services have to acquire *context* about the past, current, or future state of the environment they orchestrate. Context acquisition contains multiple sub tasks including contacting smart devices, retrieving information, interpreting device-specific results, and possibly storing them for later reuse as past context [8], [27].

Without additional support, context acquisition is a complex task for service developers. Middleware facilitates the access to distributed systems by making communication and coordination transparent for developers [13]. Context-aware middleware additionally provides context management functionality including context storage, discovery, and distribution [8]. To support developers, the context-aware middleware *Virtual State Layer* (VSL) is used as base for this work [14].

The VSL enables *software orchestrated smart spaces*. The VSL has a fixed set of API functions. They allow manipulating context in different ways such as subscribing for change notifications [14]. All VSL services use the same interface to access each other's *context*. The accessed *context* represents the interaction points of a service (see Sec. V-A).

To enable portability, VSL *context* is structured by instantiating VSL *context models* that are available in a central CMR. This allows service developers to look for possibly available *context* (e.g. an instance of the *context model* `lamp`), and search for it in a VSL instance.

As the API of the VSL is the interface for inter-service communication, all services are inherently API compatible fostering modular design, and reuse, resulting in a novel kind of context-based Service Oriented Architecture (SoA) [31].

Smart devices are managed using so-called *adaptation services*. They implement bidirectional adaptation between *smart devices* and their context model instances in the VSL [32]. So-called *orchestration services* realize pervasive computing scenarios by interfacing *smart devices* via their *context models* in the VSL. The VSL context is the virtual representation of the part of the physical world that can be orchestrated via software (see Sec. V-D).

The VSL is designed to facilitate and structure the creation of *smart space services*. It targets crowdsourced development of services for *smart spaces* [14] similar to how it happens the open source community [33] or the App economy [34] in 2014.

As described, the VSL *context models* (see Sec. V-A) are the abstract interfaces of services and as such an integral part of the service development process with the VSL support. Their creation and their convergence are the topics of this paper.

V. Existing Meta Models

Existing *meta models* for context modeling are assessed according to the following desired properties. *Simplicity-to-use* is important to enable non experts to create *context models* as described in Sec. IV. *Expressiveness* is important as *context models* define the information that can be represented in a VSL *smart space*. It is characterized by the semantic concepts a *meta model* supports such as dependencies between *context models*. *Fast processing* is important as future *software orchestrated smart spaces* are likely to contain a big amount of *context*.

Dynamic extensibility with new *context models* is required to integrate new *smart devices* at run-time. Support for *collaboration* is needed for realizing portability by sharing and collaboratively creating *context models*. *Context models* should support validation to enhance the dependability and the security of a *software orchestrated smart space*.

Different candidate concepts for representing the *context* of *smart spaces* are shown in table I. Following the shortcomings of the assessed approaches are discussed [35], [8]. Then the VSL context model is introduced. It realizes a hybrid of the analyzed concepts.

Property	Key-Value Pairs	Markup Scheme	Object Oriented	Ontology Based	VSL Context Model
Simplicity-to-use	++	++	++	+	++
Expressiveness	--	-	+	++	++
Fast processing	++	++	+	--	++
Dynamic Extensibility	--		--	-	++
Collaboration Support	--	-	-	-	++
Validation Support	--	+	+	++	++

TABLE I. Comparison of different context models.

Key-Value Pairs are very simple-to-use (++) which suits the goal to enable crowdsourced *context modeling* well but they lack expressiveness, as they are unstructured (--). Key-value model instances are *self-contained*. Their values can be read without resolving dependencies to other key-value pairs, allowing fast processing (++).

Key-value pairs do not have a formal structure that can be shared over a CMR. They realize an implicit, non extensible (--) *context model*. Collaborative editing is not supported (--). As key-value tuples do not follow predefined structures, validating them with generic algorithms is typically not possible [8] (--).

Markup Scheme-based *context models* provide a hierarchical structure that is expressed with markup tags such as XML. They are simple-to-use (++). Per se they are not expressive (-). Markup models are typically self-contained, allowing fast processing (++).

If a mechanism to distribute markup *context models* is established they can be updated dynamically ( ). The simple structure and the self-containment allow collaborative editing but a markup scheme per se does not support sharing (-). The markup language typically allows syntactic and partly semantic (e.g. boundaries) automatic validation (+) [8], [35].

Object Oriented context models are simple-to-use (++). Inheritance can easily be understood by humans [36]. The expressiveness is higher than in the former models as the inheritance represents dependency relationships (+). Object oriented *context models* can include dependencies that have to be resolved for access, making them slower to process (+).

Extending existing object oriented *context models* (--), and creating new ones collaboratively are difficult (-) as objects

typically "live" in a single environment and cannot easily be accessed remotely. Relationships and inherited properties can typically be validated [8].

Ontology Based context models are designed to model the reality. The use of concepts that are close to human reality simplifies their use. Though the conceptual creation of ontologies is simple creating ontology *context models* typically requires domain and ontology experts, as the used representations are complicated (+) [20].

Ontologies typically express several relationships between entities in their *context models*. While this brings more expressiveness (++) it makes processing them more complex as more dependencies have to be resolved to access context (–).

As described in the related work section (Sec. II), the collaborative creation of ontologies is an open research topic. Typically ontologies are defined before use and are not dynamically extended (-) or collaboratively created (-) [20], [8], [35]. With their high expressiveness and the use of markup for representation, ontologies enable syntactic and semantic validation of aspects of their *context models* (++).

A. The VSL Meta Model

Based on the evaluation in Sec. V, the VSL uses a hybrid *meta model* consisting of hierarchically structured *typed* key-value pairs (tuples) with additional management metadata: $\langle address, value, type, versioninfo, accessrights \rangle$.

The tuples are logically structured by using hierarchical addresses. The resulting logical hierarchy forms a tree of tuples as shown in Fig. 2. Nodes of the context tree are addressed in a similar way to files in a filesystem tree: parent/child/child. The hierarchic structure allows to express containment, e.g. /car/wheels/leftFrontWheel.

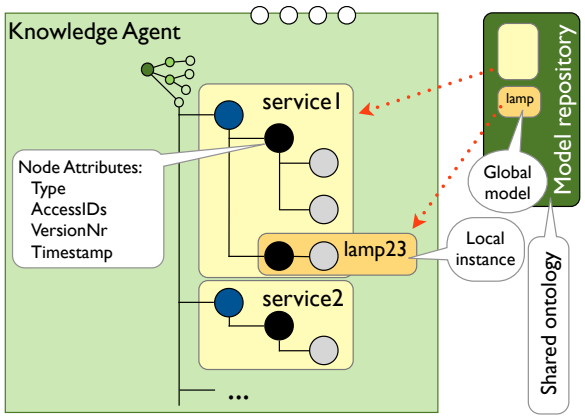

Fig. 1. Instantiation of a VSL Model from the global Central Model Repository (CMR) when a service instance is started on a node in a local VSL site.

VSL *context models* are represented in XML markup as shown in the listing in Sec. VI-C. This enables syntax validation.

To share context models, a global Central Model Repository (CMR) is introduced. It stores the XML representations of VSL *context models*. When a service is started in a local VSL *smart space*, the VSL *context model* that is associated with the service is loaded from the CMR and instantiated in the local VSL. See Fig. 1.

The correspondence between a service and its Model is expressed by the model identifier (*modelID*) that is part of the metadata of each service (see (0) in Fig. 2). Only context that has a *context model* in the CMR can be instantiated. As a result, a VSL sites always contains a subset of the Models that are defined in the CMR. A VSL *context model* instance can have any name as illustrated with the name "fooBar" for the instance of the "service2" *context model* on the left in Fig. 2.

Context models in the CMR are identified by their name called *modelID*. As described above, each VSL *context node* has a type. With the CMR, the VSL implements a dynamically extensible type system. Each *context model* in the CMR is a VSL type that can be used for composition and subtyping when creating new *context models*. The CMR only stores and provides valid models. The validity is checked on uploading new models.

The VSL knows three basic data types with restrictions. A *number* represents a number and has a lower and upper bound for the value as restriction. A *text* represents a text with a restricting regular expression. A *list* represents a list of VSL *context nodes*. It is restricted by allowed types for entries, and a minimum and a maximum amount of entries.

Subtyping extends an existing type. An example is defining a boolean as `<boolean type="/basic/number" lowerBound="0" upperBound="1" />`. When creating a subtype of an existing type the restrictions can only be narrowed. This can automatically be validated. See (2) in Fig. 2.

By creating new *context models* which contain nodes that instantiate other types, identified by their *modelID*, implements composition. The listing in Sec. VI-C shows an example. The printed VSL *context model* is composed of different nodes that are of different other types that are in turn defined by *context models* in the CMR. See (1) in Fig. 2.

The types that are defined in the CMR have two roles in the VSL:
(1) *typing*: they identify the included basic data type, e.g. `/ilab/led`.
(2) *naming*: they provide semantic names, e.g. `ledGreen`,

Identifier (2) in Fig. 2 shows the typing process: the type "number" is inherited to "byte". Typing is known from programming languages.

An example for naming is shown in Fig. 2 at the composition of the `service1` model with the `temperature` type in the CMR. The name given to the node is `tempOut` as shown in the *context model* instance on the left in Fig. 2. Naming adds semantics to the node. As the original types remain in the type field (multi inheriance), the *data type* is preserved.

The basic data types can be derived using the inheritance mechanism combined with narrowing the restrictions, or extending the inherited type with additional nodes. Narrowing restrictions is shown at the example of the derived type "byte" that inherits from the basic type "number" and restricts its range to [0..255] in Fig. 2 (see 2). Adding additional subnodes

is illustrated at the example of the derived type "temperature". It contains a number that is extended with one additional context node. This additional node must again be derived from a basic data type such as "text" for carrying a temperature unit identifier for instance[1]. As shown in Fig. 2, derived data types always contain all inherited type identifiers (modelIDs). This allows services to interface all subtypes independently (subtype polymorphism).

The basic data types can be composed, creating hierarchical structures of named nodes. All nodes carrying data have a basic data type as root (rightmost entry) of their derivation chain.

The VSL features multi-inheritance. This means that a *context node* can inherit from multiple existing types/ *context models*. This is attractive as the *context models* are used as abstract service interfaces in the VSL. A lamp that is dimmable could have a VSL *context model*, `<dimmableLamp type="lamp,specialLamp" />`. This realizes subtype polymorphism. An instance of the *context model* can be accessed as `dimmableLamp`, `lamp`, and `specialLamp`.

The multi-inheritance in combination with the rooting of all data nodes in few basic data types allows basic semantic validation (see Sec. V-C). At the same time, it allows services to access the same context data on different semantic levels. Different semantic levels mean that services that do not have knowledge about derived data types such as "temperature" in their code can still access the "number" semantics.

The described information model can be represented using different data models. XML is used as data model since a *markup scheme* is a simple-to-understand context-modeling approach (Sec. V). The following listing shows the XML definition of the inherited and restricted byte as shown in Fig. 2 which is extended with a "/derived/boolean" node that is initialized as $\perp$. Values are overwritten in the order they are included. The Model is stored as file "/derived/byteWithBoolean.xml" in the Model Repository as listed:

/derived/byteWithBoolean.xml

```
<byteWithBoolean type="/basic/number"
                 lowerBound="0"
                 upperBound="255">
  <isOn type="/derived/boolean">FALSE</isOn>
</byteWithBoolean>
```

The introduction of the CMR results in a central storage for all *context models*. The VSL *context models* define which *context* can be represented in a VSL space. The *context models* in the CMR can be dynamically extended. Therefore the VSL CMR is a dynamically-extensible, crowdsourced context model repository for *smart spaces*. At the same time it defines all data types that can be used in the VSL.

B. Organization of Model names

Each VSL Model has a unique name that cannot be reused (see Sec. VI-B). The name follows a hierarchical naming scheme, creating namespaces such as "/home/kitchen/-cooker1". Version number suffixes can be used to update existing Models, e.g. "/home/kitchen/cooker2". Such an update would typically extend some fields and derive from "/home/kitchen/cooker1" to remain backwards compatible with services interfacing "/home/kitchen/cooker1" as described in Sec. V-A.

C. Validation

The VSL *meta model* enables the validation of all context data nodes based on their restrictions.

The VSL implements so-called validators that check if a value matches the restrictions of a *context node* such as that a "number" value is a natural number and within the given range. *Context nodes* that can have values must be derived from one of the basic data types (Sec. V-A). This rooting enables the use of the basic data type validators to validate all VSL *context nodes*. This applies to values in *context model* instances in a VSL site, and to values in *context models* that are submitted to the CMR.

The CMR automatically checks if newly submitted *context models* are well defined. Well defined means here that all types used are defined in the CMR already, and that restrictions are only narrowed when deriving a data type. If a *context model* is not well defined it is rejected with an error.

The automatic validation ensures the integrity of all *context models* in the CMR. The validation of values in a VSL instance ensures basic validity of *context* that is represented in the VSL. Automated validation helps to enhance the dependability and the security of *software orchestrated smart spaces*. An accidental or intended violation of the specified structures of the *context model* is prevented.

D. Additional Semantic Providers

The *primary context* used by the VSL to identify *context models* is the VSL type. It represents containment relations between nodes on a fundamental (multi-inheritance, basic data types) and a higher semantic (hierarchical composition, inheritance) level.

The aim of an ontology is to model not only containment but diverse relationships of smart space entities that can "*facilitate the creation of a common and shared understanding*" [19]. The presented VSL *context model* can simply be extended to express more semantic relations. The reason for the extensibility is the underlying concept of only representing properties with the *context model*. So far the properties were mainly related to state that is created by services. But properties can also be used to express additional secondary contexts as described in the following example.

Location relationships are relevant in *smart spaces*. The spatial dimensions of rooms play an important role for building systems such as heating or lighting for instance. All service Models inherit from the Model "/system/service". To add location information to all services –which includes adding location information to all *smart devices* as they are connected to the VSL via services–, their basis Model can be extended with a location subtree. This subtree contains 3D geo-coordinates that give each service a unique location in the world.

[1]In the real temperature VSL Model the unit is selected from an "enumeration" that is a composition of a list containing the possible selections, and a number for the index of the currently selected value.

To make the additional information semantically usable for services, the VSL is extended with an additional search provider for location. This service periodically caches all location information from the service contexts in the VSL for fast search. Additionally it allows storing spatial objects such as rooms. The described extension enables services to query for location; e.g. "get /search/location/[living room]/[temperature]" returns the instance addresses of all nodes of type "temperature" that are within the area identified by "living room".

Similarly diverse relationships can be represented. This extends the VSL *context models* in the CMR to a crowdsourced dynamically-extensible *ontology* for smart spaces.

E. Discussion

The VSL context model consists of key-value pairs with hierarchical addresses for each node as keys. Therefore it is simple-to-use (++). See table I.

The VSL *context models* represent subtyping and composition relationships. Additional semantics can be added by extending the VSL with additional search providers. This realizes high expressiveness (++).

As the VSL *meta model* consists of self-contained tuples, it can be processed fast (++). The CMR allows to add *context models* at run-time. Via the described mechanism, the VSL instantiates new *context models* automatically from the CMR. This implements dynamic extensibility (++).

As described in Sec. VI, everyone can add Models to the CMR. The possibility to validate the markup of newly submitted models (see Sec. V-C) makes sure that the Models in the CMR are consistent. This results in good support for collaborative editing (++).

As described in Sec. V-C, VSL Models can be validated. The use of a markup scheme allows syntax validation. The rooting of all data types with values in the basic data types enables semantic validation (++).

All properties together foster a collaborative context model creation by non-experts.

VI. Crowdsourced Context-modeling

The VSL uses *context models* as abstract interfaces for services (Sec. V-A). To provide interface portability for future smart space services, a convergence of interfaces is desired. Following, a convergence mechanism for *context models* is introduced. Via the dual use of the *context models* it realizes a convergence of abstract service (and device) interfaces.

Fig. 2 shows context-modeling related components of the VSL architecture. On the top right, the *Smart Space Store* is shown. It stores service executables that end-users can deploy into their smart spaces using VSL mechanisms similar to how Apps get deployed to smartphones from an App Store. The *Smart Space Store* contains the CMR. In addition it stores metadata to services such as the relationships between services and their corresponding *context models*. As last relevant item the *Smart Space Store* stores and provides statistics about services and *context models* (Sec. VI-B).

On the left a VSL *smart space* is shown. The context-aware VSL middleware acts as context store and context broker for services (see Sec. IV).

The bottom right shows additional *smart spaces* that work like the one shown on the left, and users. Both are important for the crowdsourced convergence that is introduced in Sec. VI-B.

Users have two roles in the shown scenario. They use services in their VSL *smart space*, and they can participate in the development of services for *smart spaces*. The creation of *context models* is a fundamental task when developing a VSL service as it defines the interface (Sec. IV).

A. Crowdsourcing

The VSL *meta model* supports crowdsourced development. It is simple-to-use, it allows a collaborative creation of *context models*, and it provides automated mechanisms for validation (Sec. V-A).

The CMR in the *Smart Space Store* is open to the public. Everyone can contribute new *context models* in the representation they are shown in the listings in this paper. The CMR automatically validates new models for using a unique name, and for being valid. The automatic validity check of the CMR (Sec. V-A) ensures the integrity of all models that are published.

After acceptance, all *context models* in the CMR can directly be used by services as described in Sec. V-A.

B. Convergence

Allowing everyone to submit models is likely to result in heterogeneity. As described in the introduction, portability is needed to realize *software orchestration* in future *smart spaces*. *Portability* needs standardization to limit the heterogeneity, and to make VSL services interoperable.

Without a coordination mechanism it is likely that each developer creates the model that fits best to her purpose. As an example it can be expected that many different models for a lamp emerge. This is problematic as it re-introduces the major problem, the context-aware VSL middleware was introduced for: providing homogeneous abstractions on top of the heterogeneous smart devices of a space to facilitate the service development, and to make services portable. To converge context models it is desirable that one of the VSL *context models* for a lamp turns out to be the best, and all other lamps (e.g. dimmableLamp) derive from that lamp model.

This paper proposes a crowdsourced convergence mechanism that is based on public statistics. To create the statistics, the *Smart Space Store* collects different data and applies different metrics to rate the popularity of *context models*.

The *Smart Space Store* contains all available VSL services. Each service uses a *context model* as abstract interface. As shown on the top right of Fig. 2 the store contains a list of services, and the *context models* they use. This list can be evaluated identify which *context models* are used by which services. If it turns out, that a model "*lamp42*" is used by most services, it seems this is a popular *context model* for a lamp.

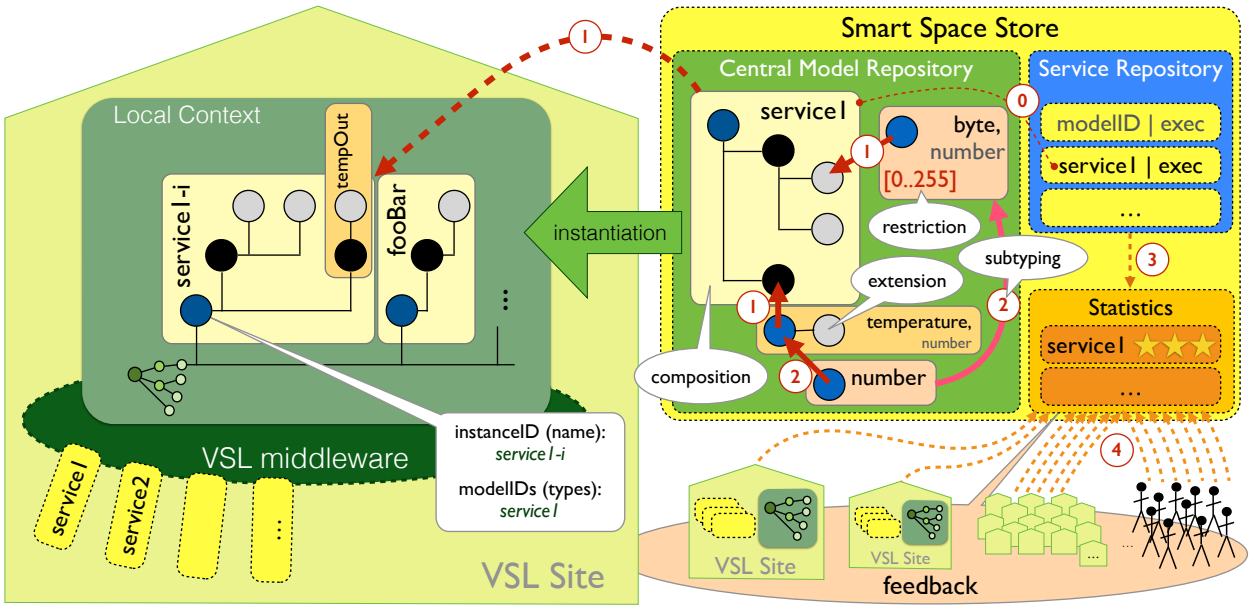

Fig. 2. System view on the crowdsourcing architecture, including the global Smart Space Store hosting the VSL context model, different smart spaces and users.

The *Smart Space Store* knows the amount of downloads per service. This can be used to refine the previous metric. A higher amount of downloads of services using a certain model results in higher popularity of a Model. Fig. 2 shows both described factors as (3).

As third input, the VSL sites and the *Smart Space Store* interact. A VSL *smart space* reports errors and usage statistics to the *Smart Space Store* when the user agrees. Both are added to the ratings of the *context models*.

Finally users can rate services, and developers can rate *context models* based on their usefulness or their beauty (e.g. naming, brevity). The automated feedback collection and the user ratings are shown as (4) in Fig. 2.

All ratings and statistical evaluations are published. This is expected to generate a feedback loop for developers as follows. The popularity of a *context model* expresses its use in available services. In case of a device driver, using a popular Model increases the interoperability of new *smart devices* with existing services. Therefore developers that want to develop a driver service for a *smart device* are likely to choose a *context model* that is very popular as abstract interface to their adaptation service.

Orchestration service developers are likely to act similar. Knowing that "`lamp42`" is the most popular *context model* for *smart devices* that have lamp functionality, searching and controlling lamps in a VSL site, using this *context model* is likely to have most effect.

The existing App economy for smartphones shows that a subset of the described mechanisms realizes App convergence [37]. Additional incentives such as revenue –popular services bring more revenue– can be added to the described scheme. Based on the experiences with the App economy the described mechanisms are likely successful in implementing a crowd-sourced *context model* convergence.

C. Evaluation

A critical factor for the proposed crowdsourced creation of *context models* is the usability of the *meta model*. Two user studies with six and eight participants were done to assess the usability. The participants were computer science students that did not know the presented concepts before the study. The VSL *context model* creation task was embedded in a larger exercise about using the VSL for creating *smart space* services [2]. Given some example *context models*, all students were immediately able to create complex *context models* for their previously built Arduino-based smart devices.

The following *context model* was created by a student team:

`/ilab/smartDevice.xml`

```
<smartDevice>
    <temperature type="/ilab/temperature">
    </temperature>
    <button type="/ilab/button"></button>
    <ledGreen type="/ilab/led"></ledGreen>
    <ledOnBoard type="/ilab/led"></ledOnBoard>
    <lightSensor type="/ilab/lightSensor">
    </lightSensor>
    <timer0 type="/ilab/triggerTimer"></timer0>
</smartDevice>
```

The listing gives an impression how much the described use of *context models* as types in combination with the multi-inheritance simplify the creation of *context models*. It becomes a modular task, hiding the complexity of the resulting model from the developer. The expanded version of the above context model is complex.

The VSL and the CMR are implemented and running[3]. The *Service Store* that provides the statistics and allows the ratings (see Sec. VI-B) is currently getting implemented.

[2] http://pahl.de/download/dissertation/ds2os.lab/
[3] https://cmr.ds2os.org/

Though it is not sure, that the proposed convergence mechanisms will have the desired effects, looking at statistics from the existing App stores for smartphones leads to the conclusion that the proposed metrics will return significant trends.

VII. Conclusion

The VSL middleware (Sec. IV) structures and facilitates the creation of software services that implement pervasive computing scenarios [29]. The use of *context models* as descriptive, simple-to-use interfaces for services pushes the complexity of smart space orchestration in the background. Simplification is required for pervasive computing to become reality [2], [1].

This paper identified advantages and limitations of different existing context modeling approaches (Sec. V). The simple-to-use dynamically-extensible VSL *meta model* was introduced. It modularizes the *context model* creation. This lowers the complexity significantly as shown in the example in the evaluation. The use of the CMR fosters the reuse of *context models*. Via the described multi-inheritance mechanism, subtype polymorphism is realized. Both foster the necessary portability of services.

To channel the diversity of management interfaces standardization is needed. It was discussed that existing standardization mechanisms for interfaces ar not suitable for the diversity of smart spaces. Consequently an architecture for crowdsourced *context modeling* was introduced (Sec. VI).

The App economy shows that user developed content is a good way to identify user demand. The principle of enabling (skilled) users to develop software that follows their interests instead of letting (few) companies decide for end-users which functionality they should want [30] is promising. Therefore the VSL is designed to enable end-users to support the *smart devices* they want and to create the *orchestration services* that realize the pervasive computing scenarios they like to have.

We believe that the presented crowdsourced *context modeling* approach is a fundamental –so far missing– building block for making pervasive computing reality outside research labs.

References

[1] M. Weiser, "The Computer for the 21st Century," *Scientific American*, Sep. 1991.

[2] G. D. Abowd, "What next, ubicomp?: celebrating an intellectual disappearing act," in *UbiComp '12*, 2012.

[3] K. Lyytinen and Y. Yoo, "Issues and Challenges in Ubiquitous Computing ," *Communications of the ACM*, 2002.

[4] I. T. U. ITU, "ICT Facts and Figures," Tech. Rep., 2013.

[5] C. Milanesi, L. Tay, R. Cozza, R. Atwal, T. H. Nguyen, T. Tsai, A. Zimmermann, and C. K. Lu, "Forecast: Devices by Operating System and User Type, Worldwide, 2010-2017," Tech. Rep., Jun. 2013.

[6] A. J. B. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon, "Home Automation in the Wild: Challenges and Opportunities," in *CHI'11*, 2011.

[7] D. J. Cook and S. K. Das, "Pervasive computing at scale: Transforming the state of the art," *Pervasive and Mobile Computing*, 2012.

[8] M. Knappmeyer, S. L. Kiani, E. S. Reetz, N. Baker, and R. Tonjes, "Survey of Context Provisioning Middleware," *IEEE Communications Surveys & Tutorials*, 2013.

[9] M. Conti, S. K. Das, C. Bisdikian, M. Kumar, L. M. Ni, A. Passarella, G. Roussos, G. Tröster, G. Tsudik, and F. Zambonelli, "Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber–physical convergence," *Pervasive and Mobile Computing*, 2012.

[10] D. Saha and A. Mukherjee, "Pervasive computing: a paradigm for the 21st century," *Computer*, 2003.

[11] G. Bell and P. Dourish, "Yesterday's tomorrows: notes on ubiquitous computing's dominant vision," *Personal and ubiquitous computing*, 2007.

[12] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newmann, "Communication Systems for Building Automation and Control," *Proceedings of the IEEE*, 2005.

[13] P. A. Bernstein, "Middleware: a model for distributed system services," *Communications of the ACM*, 1996.

[14] M.-O. Pahl and G. Carle, "The Missing Layer - Virtualizing Smart Spaces," in *MUCS 2013*, 2013.

[15] ISO, International Organization for Standardization and IEC, International Electrotechnical Commission, "ISO/IEC Directives, Part 1," ISO/IEC, 2012.

[16] J. Schonwalder, A. Pras, and J.-P. Martin-Flatin, "On the future of Internet management technologies," *IEEE Communications Magazine*, 2003.

[17] J. Schonwalder, "Protocol-Independent Data Modeling: Lessons Learned from the SMIng Project," *IEEE Communications Magazine*, 2008.

[18] C. W. Holsapple and K. D. Joshi, "A collaborative approach to ontology design," *Communications of the ACM*, 2002.

[19] S. Karapiperis and D. Apostolou, "Consensus building in collaborative ontology engineering processes," *Journal of Universal Knowledge Management*, 2006.

[20] V. Ludovici, F. Smith, and F. Taglino, "Collaborative ontology building in virtual innovation factories," in *Collaboration Technologies and Systems (CTS)*, 2013.

[21] K. Kimbler, "App store strategies for service providers," in *Intelligence in Next Generation Networks (ICIN), 2010 14th International Conference on*, 2010.

[22] M. H. Goadrich and M. P. Rogers, "Smart Smartphone Development: iOS versus Android ," in *42nd ACM technical symposium*, 2011.

[23] Apple Incorporated. iOS Developer Library. [Online]. Available: https://developer.apple.com/library/ios/navigation/

[24] Google Inc. Android Developer Reference. [Online]. Available: http://developer.android.com/reference/packages.html

[25] J.-P. Vasseur and A. Dunkels, *Interconnecting Smart Objects with IP*. Morgan Kaufmann, 2010.

[26] C. Dixon, R. Mahajan, S. Agarwal, A. J. Brush, B. Lee, S. Saroiu, and P. Bahl, "An operating system for the home," in *NSDI'12*, 2012.

[27] A. Krioukov, G. Fierro, N. Kitaev, and D. Culler, "Building application stack (BAS)," in *BuildSys '12*, 2012.

[28] M. Accetta, R. Baron, W. Bolosky, D. Golub, and R. Rashid, "Mach: A New Kernel Foundation For UNIX Development ," 1986.

[29] M.-O. Pahl and G. Carle, "Taking Smart Space Users Into the Development Loop," in *HomeSys 2013*, 2013.

[30] P. Dourish and S. Mainwaring, "Ubicomp's Colonial Impulse," *UbiComp '12*, 2012.

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

[32] Z. Movahedi, M. Ayari, R. Langar, and G. Pujolle, "A Survey of Autonomic Network Architectures and Evaluation Criteria," *IEEE Communications Surveys & Tutorials*, 2012.

[33] J. Lerner and J. Tirole, "Some Simple Economics of Open Source," *The Journal of Industrial Economics*, 2003.

[34] A. P, Matos, Christina, A. C, Vanessa, Michael, and Stijn, "Developer Economics 2012," Tech. Rep., 2012.

[35] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, 2010.

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

[37] H. Kim and I. Kim, "The Success Factors for App Store-Like Platform Businesses from the Perspective of Third-Party Developers: An Empirical Study Based on A Dual Model Framework," in *Proceedings of the Pacific Asia Conference on Information Systems 2010*, 2010.