Received December 13, 2013, accepted February 10, 2014, date of publication March 3, 2014, date of current version March 17, 2014.

Digital Object Identifier 10.1109/ACCESS.2014.2309396

Piecemeal Development of Intelligent Applications for Smart Spaces

EILA OVASKA AND JARKKO KUUSIJÄRVI
VTT Technical Research Centre of Finland, Oulu 90571, Finland

Corresponding author: J. Kuusijärvi (jarkko.kuusijarvi@vtt.fi)

This work was supported in part by the SOFIA/ARTEMIS Project and in part by the RECOCAPE/EU Project through the EU, Finnish Funding Agency for Technology and Innovation, and VTT Technical Research Centre of Finland.

ABSTRACT Software development is facing new challenges as a result of evolution toward integration and collaboration-based service engineering, which embody high degrees of dynamism both at design time and run-time. Short times-to-market require cost reduction by maximizing software reuse. Openness for new innovations presumes a flexible development platform and fast software engineering practices. User satisfaction assumes situation-based applications of high quality. The main contribution of this paper is the piecemeal service engineering (PSE) approach developed for and tested in application development for smart spaces. The intent of PSE is to maximize the reuse of existing knowledge of business and design practices and existing technical assets in the development of new smart-space applications. Business knowledge is mostly informal and domain-dependent, but architectural knowledge is generic, at least semiformal, and represented in principles, ontologies, patterns, and rules that together form a reusable architectural knowledge base for fast smart-space application development. The PSE facilitates the incremental development of intelligent applications by supporting abstraction, aggregation, and adaptability in smart-space development.

INDEX TERMS Service engineering, semantic, dynamic, architectural knowledge, reuse.

I. INTRODUCTION

Smart spaces are shared information search domains, accessible and understood by all the authorized applications that access them. Shared information is about things existing in the environment or about the environment itself. Thus, smart spaces provide information about physical environments, shared with inherently dynamic applications that are preferably developed by users themselves. Smart spaces merge two technology fields: the Internet and context-aware computing. The Internet provides generic communication facilities, while context-aware computing addresses application-domain specifics and the user's situation [1]. Thus, the Internet of Things (IoT) [2] is one of the main terms related to the development of smart spaces. The IoT focuses on uniquely identifiable objects that are virtually represented in the semantic web-like structure. The IoT first referred to physical objects, but thereafter the understanding of Things enlarged to also cover logical entities. Thus, Things can mean any entity that can have a unique identifier. Briefly, the IoT provides a smart way of connecting entities and sharing data among these entities. Another term related to smart spaces is Big Data [3] that refers to large-scale data collected from heterogeneous sources. For example, a smart-city application may need data from sensors embedded into the environment, local and/or global systems, the web and the behavior and preferences of citizens. The monitored data are analyzed with intelligent algorithms to solve the problem in hand and/or provide enriched services for citizens for their everyday lives. Thus, IoT is a prerequisite for collecting data, but is additionally required for mining, handling and providing the data for the user where and whenever needed. Therefore, new data management technologies are required for querying, filtering, analyzing, reasoning about and representing the semantic information in an accurate, understandable and personalized way for smart-space users. Moreover, these applications are likely to be created at run-time, when all factors that influence service creation are available. This type of predictive functionality requires advanced intelligence realized by self-management capabilities that are able to self-configure the system properties in a way that best matches the situation in hand [4]. In other words, the systems and their applications are to be dynamic in nature. Thus, the IoT makes the structure of networked systems dynamic, and semantic data enables interacting with the systems' internal states dynamically. The third factor that demands fast and flexible smart-space application development is crowds [5],

i.e., end-users and communities that participate in creating intelligent applications based on data collected from heterogeneous sources (e.g., the Internet, smart spaces, sensor networks, etc., that provide open data). A crowd is composed of ordinary people who have problems, and who possess basic skills to solve these problems by developing applications that meet their requirements if there are ready-made building blocks that make application development easy, fast and fascinating. There exists experimental evidence that documented patterns improve understanding of source code, enable professionals to trust the quality of source code and reduce the effort to analyze the source code [6]. However, there is still a lack of systematic approaches for development and evaluation of adaptive and pervasive applications, and a lack of empirical studies as to what extent ontologies can be used for large scale deployments with user involvement and assessment [7].

To contribute to meeting these new challenges, we introduce a novel approach for creating adaptive smart-space applications that we call Piecemeal Service Engineering (PSE). These applications are aggregated from pieces of models and code artifacts that define concepts, data, functionality, and the behavior of software pieces. All the models are defined separately. Briefly, the main goal of a smart-space application is to define how the selected pieces work together. Thus, the application description defines what pieces are to be combined and how it is to be done. The role of the architectural knowledge base (AKB) is to ease and speed up the development of smart-space applications. The AKB provides knowledge as principles, ontologies, models, patterns and rules implemented by appropriate modeling languages such as OWL [8], UML [9] and SPARQL [10]. Thus, the languages used also represent knowledge that is considered standards in the related engineering domains. Therefore, the knowledge base provides a stable development platform for applications that may evolve frequently. Dynamism and adaptation are defined by rules, analysis models and the order of execution of functionalities to be performed when a specific application-dependent event occurs.

The paper has been structured as follows. Section II gives the background information of other researchers' studies and our own research results. Sections III and IV introduce the PSE approach with an example application. Section V presents the evaluation results from the case studies carried out in the laboratory and industrial settings. The conclusions section gives closing remarks.

II. BACKGROUND

A. MODEL- AND KNOWLEDGE-DRIVEN SOFTWARE ENGINEERING

Model-Driven Architecture [11] is an approach that guides the specification of information systems. Here, a model means a formal specification of part of the function, structure and behavior of a system. The idea is to separate descriptions of functionality from the implementation specifications and thus provide interoperability and portability of a system. Implementation-independent descriptions last longer than implementation specifications that change as soon as a better technology is available.

Software architecture modeling concentrates on multiple views. ISO/IEC/IEEE Std. 42010 [12] recommends an architectural view as a representation of a whole system from a perspective of a related set of concerns. Architectural views have formed the bases of a number of design methods during the last few years. Considerable interest exists in applying UML and various views to modeling software (product-line) architectures [13]. Moreover, several methods have been developed to model architecture views with the UML notation [14]. The focus has been on providing a common understanding of a system of systems under construction and to determine its/their commonalities and variabilities. Thus, these methods have focused on managing design-time variabilities. Recently, the focus has transferred to representing and managing the diversity of systems' behaviors at run-time. For example in [15], model-driven engineering methods are extended to integrate new functionalities and behaviors into running enterprise systems by integrating organizational and coordination theories with model-driven architecting to manage the development and deployment of dynamic service-oriented business applications. The approach has similarities to ours, but smart spaces are ecosystems that embody dynamism on all levels: business collaboration and networking, implementation technology, communication, information, functionality, behavior and coordination levels. Thus, due to this dynamism, coping with the complexity of smart spaces is extremely important.

The open issues of dynamic systems are how to minimize information communication/processing overheads and adapt the behaviors of applications to changing conditions in their operational environment [16]. Because information overheads have an impact on performance, applications should avoid exchanging and reasoning about information that is potentially useless for them. Moreover, environments that run intelligent smart-space applications are open, heterogeneous, and variable [1]. Therefore, it is crucial to ensure on-the-fly interoperability by adaptive middleware technologies and applications that behave adaptively and efficiently in several deployment settings and run-time conditions [16]. Adaptive architectures and ontologies for defining commonalities of contexts, services and quality attributes form the basis for defining common vocabularies that intelligent applications can exploit. An extensive analysis of self-adaptive systems and their capabilities is given in [17]. Here we summarize some of the most important capabilities to illustrate the current status of methodologies provided for intelligent application development, ontologies used for enhancing modeling architectures and managing context and quality at run-time.

The approach introduced in [18] integrates model-driven development and ontologies. It is based on the model abstractions of MDA and commonly used modeling languages such as UML and OWL. The approach also enhances the software engineering process with a domain ontology for modeling

platform-independent applications and a context ontology for reasoning. The approach is abstract; it lacks information about how the context ontology is to be specified and processed at run-time, and how it is to be transformed for the use of intelligent applications. In [7], a thorough survey of software technologies of adaptive user-centric pervasive software tackles the existing software technologies for formal modeling, knowledge representation and reasoning. Those authors have concluded that the literature still lacks contributions concerning, among others: i) systematic approaches for design, development and evaluation of adaptive and pervasive systems and applications, ii) an elaborate analysis of the rules used in adaptive and pervasive systems, and iii) proofs reflecting that existing proposals can scale well for large deployments. This paper focuses on these issues and contributes to solving them in smart-space application development.

The development of adaptive and pervasive applications is heavily based on the context modeling and reasoning techniques used for achieving adaptability. In [19], the authors represent object-role based models, spatial models and ontology based models as the approaches of context modeling. Architectural adaptation approaches have been surveyed and compared in [4]. Because the applicability of technologies heavily depends on the requirements of the application and its users, there is need for diverse technologies. However, irrespective of the technologies used, the focus should be on smooth integration of human intelligence and machine capabilities, with emphasis on human aspects and on cost-effective development methodologies and tools that support the development-time and run-time adaptations [7].

The context-aware service-creation framework [20] includes several artifacts that support smart-space development: context ontology, a context modeling language and a tool environment that assists in context-aware service creation. The tool environment supports context model definition and validation, context model-to-model transformation and context model-to-code generation. The transformation provides a mapping from the context metamodel to the target metamodel. The approach tackles the structural and static parts of intelligent application creation. However, the dynamic aspects of context-aware services are not supported.

Recently, interest in using ontologies for describing and managing quality attributes has increased due to the growing awareness of the importance of quality characteristics in service oriented systems. Moreover, the quality aspects need to be managed not only at design-time but also at run-time. In [21], an ontology for non-functional requirements was introduced with three views: an intra-model dependency view for describing the relations between the software entities, an intermodal dependency view for describing the structure of interdependent entities, and a measurement view for defining measurable requirements. These views are required for managing quality properties at run-time. The intermodal view defines what the quality property is and how to implement it. The intramodel dependencies are used for reasoning purposes, and the metrics are used for defining quality goals and measuring how well these goals have been met.

B. OUR EARLIER WORK ON MODEL AND KNOWLEDGE DRIVEN SOFTWARE ENGINEERING

Our earlier work has focused on enhancing the use of architectural knowledge in the development of service-oriented architectures. First, we defined a service taxonomy, a reference service architecture and a set of basic services as reusable artifacts for developing wireless services and mobile applications based on styles, patterns and communication standards [22]. Second, we enhanced our approach by defining a method for specifying quality requirements in an unambiguous way [23] so that systematic quality evaluation, e.g., concerning integrability, extensibility and reliability, could be carried out from architectural models. We also developed a set of tools for evaluating quality from models and running systems [24], [25].

In [24] and [26] the ontology orientation is used for defining quality-attribute ontologies, especially for defining their metrics. On the one hand, the quality- and model-driven design methodology, which is also introduced, exploits ontology-oriented design for specifying, representing and managing quality attribute specific knowledge by ontologies. On the other hand, architectural knowledge is specified, represented and managed as styles and patterns. Both types of models can evolve separately. The mapping of quality properties to functional elements of architecture is made by a tool chain that supports each development phase of the model and quality driven service engineering.

In [27], the vision of the Piecemeal Service Engineering (PSE) approach was introduced for the first time. The vision was inspired by our experience with software product families, model-driven development, ontology-oriented design and our first steps in the development of smart spaces [1]. Recently, we have developed an adaptation framework for the run-time quality management of situation-based services [28], [29]. However, there are still some obstacles for dynamic and intelligent applications. Reusable solutions are required both for reasoning and for adaptation. We selected a W3C standard language, SPARQL, for context reasoning. However, analysis models and reasoning support for run-time quality management are almost totally missing. Studies are still required to provide viable solutions to design and implement dynamism in smart-space applications and to apply the developments in both small and large scale applications. Thus, the solutions, methodologies and tools should provide a scalable and cost-efficient way of developing intelligent smart-space applications.

III. PIECEMEAL SERVICE ENGINEERING APPROACH

Our contribution is the PSE approach intended for smart-space application development. The approach is kept as simple as possible, so that application developers can focus on application specifics and need not concern themselves with generic technical and, in most cases, complex things. Thus, the complexity is covered by the artifacts stored in the

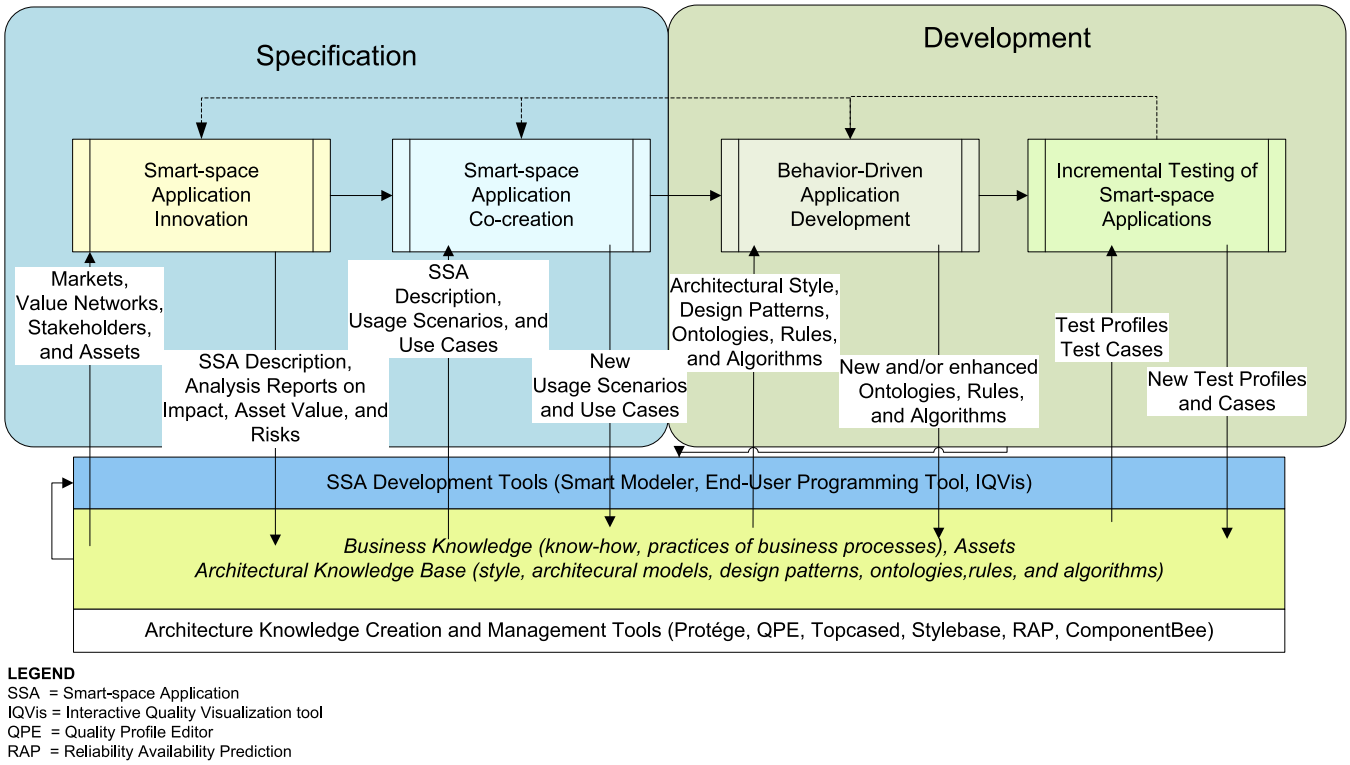

FIGURE 1. **Workflow of the piecemeal service engineering approach.**

knowledge base and provided for the use of the application developers through the smart-space application development tools (Fig. 1). Before the content of the knowledge base can be exploited, it must be created and stored in a standard way. Therefore, only standard modeling languages, such as OWL, UML and SPARQL, and tools that support these languages, are used. Here, we focus on the application development, i.e., how to create new applications by exploiting the content of the knowledge base. Therefore, the main phases (depicted in Fig. 1) of the approach answer the following questions:

1. How to take into account the business, domain and technical concerns in smart-space application development?
2. How to break down the application idea into pieces and how can different actors contribute the application development?
3. How to carry out the behavior-driven smart-space application development?
4. How to test an adaptive smart-space application?

The PSE approach is an iterative and incremental development approach. The above main phases of PSE are iterated, and depending on the development phase and type of the application, one can iterate all of them or a selection of them, taking into account that all of the phases following the selected first phase are to be executed. In these four main phases, several methods and tools are used. To keep this presentation understandable, we illustrate the steps with examples, or refer to other publications where further information and examples are given. The following subsections present the specification part of the PSE. Thereafter, Section IV presents the development part. First, the phases and their outcomes are introduced. Thereafter, we discuss the knowledge provided to the application developers and what new knowledge could be created during the application development. Whenever possible, we also give examples of how and with what tools to exploit the existing knowledge.

A. PSE: SPECIFICATION PHASE

1) SMART-SPACE APPLICATION INNOVATION

The first phase of the PSE - the Smart-space Application Innovation - mainly addresses the domain aspects of the application development, also taking into account the business drivers and technical constraints. The phase proceeds by carrying out the following activities:

- *Idea identification* – What type of application? What is it for and to whom will it be delivered? Why? These are questions to be discussed in the group of business, domain, and technical professionals that collects ideas for new applications. The work results in a list of draft ideas for applications with justifications why they are important. A brainstorming workshop with a facilitator is used as a method for collecting application ideas.
- *Idea scoping and analyses* – The ideas are analyzed from different viewpoints. Business analysis defines and

justifies potential markets, value networks, roles of partners in the value network, costs and pricing. Domain analysis provides information about the trends in the application fields by forecasting and analyzing the stakeholders' future needs. An application idea may also be pre-evaluated with end-users via participatory methods and supporting tools (e.g., interviews and social media for collecting users' opinions, and thereafter, co-designing user-application interactions via visual models, paper prototypes or mockups). The domain analysts also assist business people to scope the value network and estimate what the value of the new application would be for each actor in the network. Technical experts analyze what technical challenges there would be in the development of the application. The following is also analyzed: what existing assets are available, what is their quality and openness? What new assets and technologies are required? Different feature, domain and quality analysis methods may be used. Risk analyses that have their own methods are to be made from all viewpoints: business, technology and the application domain. Generally, different types of analyses are made, each needing different methods and practices. We do not go into details here. This phase results in the initial description of a smart-space application and related analytical reports on its expected impacts, available assets and identified risks and their severities.

The smart-space application innovation phase exploits the knowledge base and tools as follows:

- *Business know-how* is dependent on the organizational culture, on individuals and on the business practices of how business knowledge is collected, stored and shared in the organization. Forecasts provide analyzed information about markets and technology trends. Some companies provide commercial business analysis services on markets, value networks, partnerships, interest groups, etc., that may help in decision making. However, the core of business knowledge is kept secret. To some extent it can be opened to partners with long-term relationships with the organization. Part of the business knowledge is tacit, and if documented it is confidential. Thus, a generic (semi-)formal method for sharing business knowledge is difficult to establish. If business patterns can be identified, they are kept as internal knowledge and/or shared informally.
- *Assets* – Documents of existing assets are used to resolve their availability, suitability and quality for the application development in hand. The decision to utilize these assets impacts on scheduling, costs, profits, etc.
- *Architectural knowledge* – Technical and business people can use the architectural knowledge base to estimate what artifacts can be used as is, possibly modified, what the quality of the artifacts is and if the quality is satisfactory. The architectural knowledge and tooling is used to estimate the quality and functionality of existing resources, and thereafter, the effort and cost required for the application development.
- *Tooling* – In analyzing the existing artifacts of the architectural knowledge base, the following tools are used [30]: Protégé is used for analyzing ontologies. Stylebase is used for evaluating existing architectural models. If there are model and code assets, the RAP tool can be applied to predict the reliability and availability from the models and/or test them in a running system via the ComponentBee. Open source tools for sharing and managing architecture knowledge have been explored in [31] and [32].

Example - Enhancing an existing smart environment

The SUM-SS (the Seamless Usage of Multiple Smart Spaces) pilot (described in [28]) consists of multiple smart environments (e.g., smart city, office, and home), multiple devices (laptops, tablets, mobile phones), and integrates multiple technologies. The intent of the example application is to enhance this existing smart environment, and specifically the smart home/office part of it, with a new device that will improve the user experience and motivate the users of a smart building to act in ways that decrease energy consumption. To demonstrate the first phase of PSE, the existing smart environment, containing a home automation system that allows the energy consumption by different devices to be monitored, is enhanced by a new system, LumEnActive [33], that projects information onto different surfaces. A new smart-space application (SSA) is required for this new system to indicate which devices are consuming the most energy, and thereafter propose energy-saving tips to the users. This new application helps to save energy by making users aware what devices consume the most energy. In addition, it allows authenticated and authorized users of the space to control the energy consumption by turning off selected devices. Some parts of the application are readily available in the AKB: for example, the energy consumption data of the devices, reasoning agents for calculating costs, user authentication and access control, and some simple uses cases of the LumEnActive system used to display static information on walls. This new application requires integration of these assets into a working scenario in the smart space.

2) SMART-SPACE APPLICATION CO-CREATION

The second phase of the PSE - the Smart-space Application Co-creation – is crucial because the success of the SSA development mainly depends on how the development work is divided into parts, what dependencies there are between these pieces and what their priority order is, and how the co-workers contribute their development. This phase includes the following activities:

- *Describing a usage scenario of an application* – This activity continues the work of the first phase by using brainstorming in small groups as a method for innovating technical aspects of the new application. People from different technology areas are involved in the work. Application domain experts also provide their

knowledge for describing the scenario of the SSA. The result is a revised version of the SSA scenario description that includes a short narrative story with a context description that gives an understanding of why and where the SSA is needed, and how and by whom it is used. Thus, the usage scenario specification motivates developing the application by giving hints on technologies that could be used for implementation.

- *Breaking a usage scenario into pieces* – The next activity is to break down the scenario description of the SSA into a meaningful set of use cases that illustrate specific aspects of the scenario. The work is performed concurrently by a set of focused teams.
  - One team looks the SSA from the application domain point of view: they define the end-user stakeholders, their interests in and requirements for the application as a set of use cases.
  - Another team focuses on the stakeholders involved in the development and maintenance of the SSA. Thus, they look at the actors who are involved in the life cycle management of the SSA (product/service managers, architects, designer, testers, etc.), and define the use cases from their viewpoints. For example, what is the architectural style to be followed and the technological constraints the SSA implementation has to fulfil?
  - A third team focuses on what contextual information is required for application's situational behavior, how it is to be collected, shared, analyzed, interpreted and used by the application. In practice, the third team has to check and describe how the use cases defined by teams 1 and 2 work together, i.e., how the SSA must behave in different situations and if all required pieces have been defined by the developers.
  - The rest of the teams each focus on a specific quality attribute: the security team looks the scenario from the security viewpoint, the performance team describes how performance and scalability issues are to be considered, and so on.

Thus, as a result of the usage scenario description activity there is a collection of use cases that describe the same usage scenario from different viewpoints. These descriptions are kept separate because they have different priorities and evolution, and their reuse is easier when their intent is focused and easy to understand. For example, security has so many sub-characteristics that it is easier for reuse purposes to break the security goal down further and store smaller pieces of use cases. This phase results in a reusable usage scenario and related use cases. Thus, the SSA description is a set of behavioral models stored in the architectural knowledge base for reuse.

The smart-space application co-creation phase exploits the knowledge base and tools as follows:

- *Business knowledge* – provides information about previous SSA descriptions that were evaluated but not realized. Thus, some of the SSA ideas accepted for development may not have been implemented, and the reasons behind these decisions are valuable in making feasibility checks while defining further the usage scenario and use cases in hand. Thus, knowledge of unsuccessful earlier actions in SSA development is reused to avoid repeating the mistakes.
- *Assets* – The key artifact is the SSA description and the related analysis reports provided as output from the first phase – the SSA innovation. The available SSA descriptions are used for analyzing the differences of applications. Existing usage scenarios and use cases may provide ready-made building blocks for the SSA co-creation phase, especially while defining the use cases from different viewpoints. This will have impact on prioritizing use cases. It is also important that all artifacts stored in the knowledge base prior to this particular SSA co-creation phase can be reused. This is especially important for defining timing, cost and technological constraints, and prioritizing and scheduling how to develop and maintain the SSA.
- *Architectural knowledge* – In defining the SSA usage scenario, the architectural knowledge used covers the design principles. Instead of defining a style as a structural diagram, we analyzed 56 application scenarios of three types of smart spaces: personal spaces, smart indoor spaces and smart cities. The analysis resulted in 16 quality requirements and 12 high priority functional requirements; these requirements were used as architectural drivers while defining the 12 design principles, i.e., the architectural style of the interoperability platform (IOP) for different types of smart spaces. [1] Table 1 introduces a set of design principles that all SSAs exploiting the IOP must follow. As seen, the principles define only the basic architectural elements, such as shared information and service, and the capabilities that address functionality and/or quality, such as extensibility, security and context. Thus, this knowledge has impact on which viewpoints of the SSA architecture must be defined, i.e., what use cases are needed. In addition to design principles, architectural design decisions may also be needed to define principles (i.e., design decisions) related to the development and maintenance processes [34].
- *Tooling* – In this phase no specific tools are required. However, the use of a standard tooling environment is beneficial, especially one with support for design flow. Therefore, we preferred to use the Eclipse platform and the TOPCASED tool that is one of the plug-ins running on top of the open source tool platform. Prioritization of use cases could be managed, e.g., by a simple Excel form. However, due to the life cycle management and evolution of the SSA application, it might be necessary to use an appropriate web-based project management tool. For commercial software, Atlassian JIRA [35] is a good agile project management tool. For an open

TABLE 1. Fragment of design principles.

Principle	*Definition*
Shared information	The IOP manages a shared information search domain called Smart Space (SS), accessible and understood by all the authorized applications. Information is about the things existing in the environment or about the environment itself. The information is represented in a uniform and use-case independent way. Information interoperability and semantics are based on common ontologies that model information.
Simplicity	The IOP deals with information. The IOP information level is use-case agnostic.
Service	An SS is implemented as a semantic information broker (SIB) service, offered by a service platform and intended for the sharing of interoperable SS information. Each application interfaces to one or more SSs through a Smart-space Application Protocol (SSAP).
Extensibility	The IOP provides functionalities to insert and remove information. IOP functionality may be extended with domain ontologies and with information manipulation services. If these services become commonly usable, they are called "IOP extensions".
Security	Information security is an IOP extension, handled both at the service level and information level. Appropriate ontologies define whether the IOP is required to respect privacy, enforce authentication and access control policies at granularity finer than SS itself, or if shared information integrity, confidentiality and trust need to be provided.
Context	Context management is an IOP extension. IOP should enable the aggregation of interoperable information for the benefit of application usability and IOP performance. As the information returned by the IOP depends on the query and available information, the ontology is to define context semantics.
Legacy	Legacy devices and systems access and exchange information through the IOP. Such exchanged information is modeled by Domain Ontologies. Legacy devices may provide information to the IOP and subscribe to information from it.

TABLE 2. Use Case 1: Authentication.

Name	**Authentication – Certification/presence**
Id	UC Authentication 1
Description	Authenticate a user
Trigger	A user tries to access information that requires authentication. A user tries to access information that requires higher authentication level.
Preconditions	User is not authenticated.
Postconditions	User is authenticated with level1 authentication.
Actions	1. When a user joins a smart space via an NFC tag, his/her presence in the space is confirmed. 2. When a user joins a secure smart space, the user's certificate is checked for authentication.

TABLE 3. Use Case 2: Authentication.

Name	**Authentication – Biometric**
Id	UC Authentication 2
Description	Authenticate a user
Trigger	A user tries to access information that requires authentication. A user tries to access information that requires higher authentication level.
Preconditions	User is not authenticated.
Postconditions	User is authenticated with level2 authentication.
Actions	1. User's walk pattern is recorded and sent to the smart space for authentication. 2. User is authenticated.

source and/or free tool, Agilo for trac [36] is a good choice.

Example – Enhancing an existing smart environment

Our usage story for the enhanced smart-space environment is as follows: When entering the smart home, Anna sees a spot of information on the wall that shows the five devices consuming most energy. However, Anna wants to know what devices and appliances are monitored, and what the current indoor conditions are. Therefore, she authenticates herself to the smart home. Because Anna's authentication level gives rights to obtain detailed information on specific objects in the house, the requested status information is mirrored on the wall. She notices that temperature is too high and wants to increase cooling. However, because Anna's authentication level does not give rights to control the indoor conditions, she is asked to contact the house owner, John, to obtain the rights for control, or to stop three devices that are warming the room the most. Anna selects the last option and stops the devices through her mobile phone according to the instructions given on the wall.

TABLE 4. Use Case 3: Authentication.

Name	**Authentication – user name / password**
Id	UC Authentication 3
Description	Authenticate a user
Trigger	A user tries to access information that requires authentication. A user tries to access information that requires higher authentication level.
Preconditions	User is not authenticated.
Postconditions	User is authenticated with level3 authentication.
Actions	1. User inputs his/her user name and password for the smart space 2. User's credentials are checked. 3. User is authenticated.

Tables 2–4 depict three reusable use cases of a high-level use case of '*Authenticate the user*', which is part of a usage scenario of '*Accessing different sensitivity-level information in the garden*'. The use cases show how to handle different authentication levels for one case in a smart greenhouse demonstrator [37] and SUM-SS pilot [28] (information security specific usage scenarios). The goal is to adapt these use cases for the energy-efficient smart home environment. First, when the gardener arrives at the greenhouse he/she is 1) authenticated by physical presence and analyzing the certification level (e.g., by touching a NFC tag). Second, when the gardener walks in the garden, he/she is 2) authenticated biometrically by walking recognition and authorized to view specific details of the garden. Third, when the gardener wants to modify private information (modify inventory, change prices, etc.) he/she is 3) authenticated by asking his/her

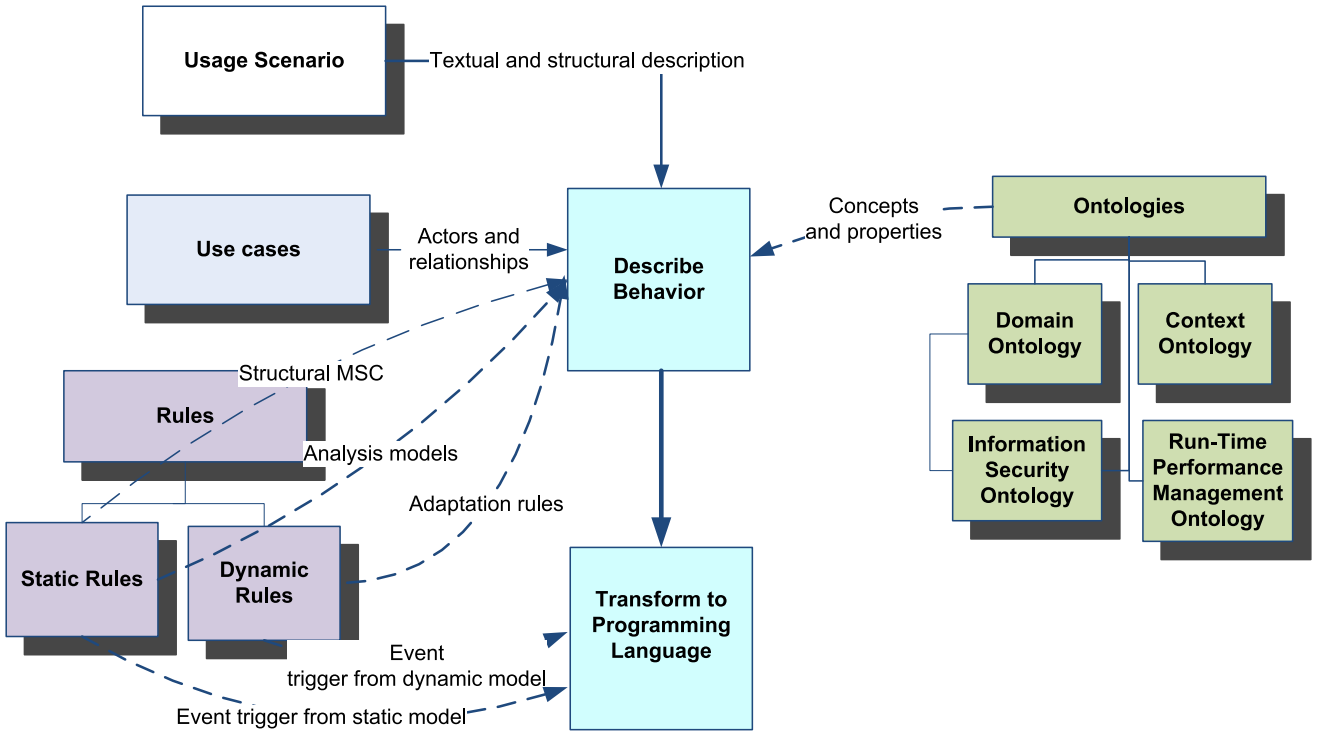

FIGURE 2. Behavior-driven application development.

username and a password. The selection of the authentication method is typically done case-by-case. However, an intelligent algorithm can be used for making this selection automatically.

IV. PSE: DEVELOPMENT PHASE

A. BEHAVIOR-DRIVEN APPLICATION DEVELOPMENT

The third phase of the PSE – Behavior-Driven Application Development - starts the application design by exploiting the usage scenario description, the set of use cases that define the specific viewpoint of the usage scenario and the reusable artifacts provided in the AKB. The term 'intelligent application' is used to describe the intelligent collaboration of a set of SSAs at a given time. These SSAs are able to behave intelligently due to their inherent dynamism that enables proactive and reactive responses to the end-users' needs.

The behavior-driven application development consists of two activities (Fig. 2): i) First, application behavior is described by integrating the pieces of artifacts from the AKB. ii) Next, the application behavior description is transformed to the target language. If the actors (i.e., software agents and services) required for the application and specified in the use cases are already available in the AKB, the transformation concerns only translation of the rules used into SPARQL queries.

The *Describe Behavior Activity* includes the following steps:

1) Use the usage scenario description as a starting point for defining the main behavior of the application. Define the agents and services required for the behavior and define the interaction between the actors.
2) Annotate the behavior description (sequence diagrams) with the concepts and properties from the related ontologies.
3) Select the appropriate rules related to the selected ontology and map them to the actor in the specific state. If no appropriate rule is available, define it in textual format.

The activity results in an annotated message sequence diagram or a set of them.

The *Describe Behavior activity* exploits the knowledge base and tools as follows:

Assets – The key assets are the usage scenario description of the SSA under development and the related set of use cases.

Architectural knowledge from the AKB is heavily used: ontologies, models, patterns and rules. While principles form the first body of the AKB, ontologies form the second. Ontologies were developed concurrently and in an iterative way, and resulted in a domain ontology, context ontology, security ontology and run-time performance ontology. Domain ontologies were distilled from the application scenarios defined for four smart spaces; personal, home, work, and city.

The *context ontology* development was started by defining the levels the context ontology was to cover: physical, digital, situational, historical, user and social contexts [28]. The physical context ontology defines the concepts related to environmental monitoring objects. The digital context ontology defines the concepts related to the actors and features

that are involved in service creation and in the execution environment. Thus, the digital context could also be called a spatial context. Within the context of a specific SSA, however, the situation and user contexts are used for scoping the context further. The situational context defines the temporal view of an SSA and the user context sets specific restrictions and preferences as to how the service/application should behave. Moreover, historical and social contexts may give additional knowledge for more intelligent reasoning according to the user's background and the intent of the social community for which the user is acting. Because context ontology plays a key role in several architectural levels and forms the core artifact for dynamism and adaptation, we consider the context ontology as a core ontology of smart spaces; it is required in every application that intends to behave in an intelligent way by taking into account the space and time where and when the behavior occurs.

Information security measuring ontology (ISMO) [38] takes two existing ontologies as a starting point: the Software Measurement Ontology (SMO) [39] and the Ontology for Information Security (OIS) [40]. The purpose of combining these two ontologies is to achieve an ontology that makes it possible to measure the fulfilment of security requirements, i.e., security goals and levels. In other words, the purpose is to enable an operational measurement of security correctness. Therefore, the requirements are described with the vocabulary from the OIS. Fulfilment of requirements is measured with indicators that combine several measures defined in the SMO. The security measures, i.e., indicators, are different for each security goal; e.g., the levels of authentication and non-repudiation are measured with different measures. However, these measures can utilize the same base measures. The same security goal can be achieved with different countermeasures, which in turn might require their own measures. Hence, there are only a few concept-to-concept mappings between these two ontologies, but additional mappings appear when the measures are instantiated. By using a terminology of these two ontologies, a mapping refers to the equivalent property between the concepts of these two ontologies. Adding mappings for instantiated measures requires domain expertise, i.e., the capability to recognize applicable measuring techniques for a particular security goal and related mechanisms. Furthermore, the mapping requires a capability to recognize threats that affect the particular security goal and/or mechanism. The mappings and usage of ISMO are discussed more thoroughly in [38].

Architectural models and patterns form the third body of reusable architectural knowledge in the AKB. Depending on the interoperability level [28], different architectural patterns are used. The semantic interoperability is supported by the information broker (SIB) architecture [41]. The focus is on understanding data, and information is used as an object of integration without the knowledge of how it is used. The RDF Schema, ontologies and semantic web technologies provide means to implement the semantic broker architecture.

Dynamic interoperability is supported by a *micro-architecture* that follows the MAPE-K (Monitor, Analyze, Plan and Execute – with Knowledge) pattern. The focus is on context changes, and events are used as objects of integration. Models are used for defining abstractions, concepts, relationships, rules, functionality and behavior on the dynamic interoperability level of agents. These agents may also follow specific patterns and form reusable building blocks. For example, the run-time monitoring framework introduced in [42] uses the Façade pattern to provide a coherent interface for agents to communicate with the semantic information broker. Separation of concerns is followed also, to improve reusability by offering unique interfaces for ontology providers, application logic and dynamic behavior. Behavior can be changed at run-time due to the use of the Strategy pattern. Moreover, the monitor manager enables to enhance the monitoring framework at run-time by adding new monitors and controlling them via well-defined interfaces. Thus, the same architectural pattern can be applied to design both simple and more complex monitors. To change the ontology at run-time, a configuration URI is provided by each monitor. This allows a monitor agent to be configured via the semantic information broker service. Moreover, a specific wrapper layer is provided for agents that use different programming languages.

Rules have an important role in application descriptions, and therefore they have been introduced separately in Fig. 2. Rules can be related to monitoring, analysis or adaptation and they can be static or dynamic. On one hand, the monitoring rules related to (security) base measures are static and are implemented as software monitors executed as part of applications. Adaptation rules, on the other hand, are dynamic and can easily be changed at run-time [28]. Analysis may concern context or quality attributes such as security and performance. Thus, the analysis rules are primarily static, i.e., they are predefined in the analysis models referred in the ontology. However, the link to the relevant analysis model can be changed at run-time, and therefore new analysis models can be added to the running systems [29].

The rules are represented in English IF-THEN-ELSE structures or logical operations. The type of description language depends on what is described. Analysis models are described with statements close to natural English. Context-related rules are mostly defined with IF-THEN-ELSE structures, whereas logical operations are useful, for example, for describing security adaptation rules. The main criterion for selecting a rule specification language is that its abstraction level fits the level of abstraction required from the behavior description, and the rules are understandable for all stakeholders involved in the application development. Thus, the behavior description is the only design artifact that describes the application as a whole.

Tooling – In creation and use of the AKB, the following tools can be used. The Protégé tool [43] was used for defining ontologies. The quality attribute ontologies, e.g., the security ontology, are thereafter imported to the Quality Profile Editor [26] (see Fig. 1) that makes it possible to create quality

profiles for the SSA and map them to the UML models of the SSA. However, we wanted to keep quality ontologies and UML models separate so that the ontologies could easily evolve and be updated after application deployment. Thus, we did not use QPE, but stored the ontologies used into the semantic information broker (SIB), thus being part of the IOP. Stylebase [26] is a tool for AKB management. A designer uses it to find appropriate patterns for developing the SSA. Smart Modeler [44] can be used to create a visual model of the application by utilizing existing model blocks from AKB repositories. A block can be, for example, implementation of one use case. As for a continuous integration platform in the development teams, Jenkins [45] can be used to automate building.

As part of Behavior-Driven Application Development, the *Rule Specification* is feasible when performed as follows:

1. Select why the rule is required by positioning its place in the MAPE-K model (Fig. 3).
2. Define why you need the rule. The intent of the rule can be related to context, security or performance management.
3. Decide if a static rule is sufficient or if a dynamic one is required. This decision heavily relates to the evolvability and extensibility of the system/space but also increases the complexity and might result in decreased performance. Typically, rules are first created static and later refined and implemented as independent dynamic models that can be updated at run-time.
4. Define the analysis results as a set of trade-off analysis rules, e.g., the rules for ranking security first instead of performance or vice versa.

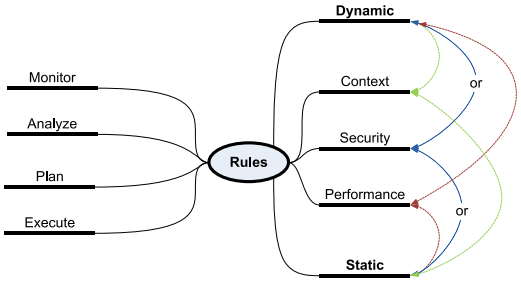

FIGURE 3. Taxonomy of rules.

As a result of this phase, the behavior of the SSA is defined in a set of rules that are implemented by a real rule language (more dynamic) or a specific programming language (more static).

The *Transform to Programming Language* activity exploits the knowledge base as follows:

Architectural Knowledge: The last body of the AKB foundations includes standards and related implementations. Modeling and programming exploit standard languages and schemas such as UML [9], OWL [8], RDF [46] and SPARQL [10]. UML is used for representing architectural models and OWL for ontology descriptions. In case of intelligent services, the means of representing semantics plays an important role. Therefore, the RDF schema was selected for representing semantics of data as RDF triples. After several experiments, SPARQL was selected for making queries to a semantic triple data base and making decisions based on rules defined as logical sentences or mathematical expressions as part of the related ontologies, e.g., security ontology and context ontology. The goal of making ontologies reusable was guaranteed by selecting a standard description language for them, storing the defined ontologies as RDF triples in the same storage with data and making queries with a standard query language.

The initial rule classification (Fig. 3) helps in finding an appropriate rule for the case in hand by addressing the sought-for rule's intent (why), objective (what) and means (how). When the rule classes are mature enough, the rule taxonomy can be defined for searching for an appropriate rule via an identification mark. In this way, the rules can become reusable assets for different types of intelligent applications. Examples of some static and dynamic rules are provided in the following experiment section.

Tooling – Smart Modeler can be used by developers to make new rules or utilize existing rules from the knowledge base. An end-user programming [47], [48] tool can be used also by end-users to create simple rules or reuse existing rules from their earlier applications.

Example – Enhancing an existing smart environment

In the earlier phase, we depicted the use cases for authentication. Here we define the use cases for the new application and the rules related to the existing use cases and the new ones if needed. The behavior can described more precisely as sequence diagrams (see [28]), but in this this case we use textual presentation.

Reused use cases:

- Security use case 1
  (a) Description: User can authenticate him/herself in the home space.
  (b) Solution: Rule – Monitor (Security, Static) see Fig. 3. The user authenticates herself to the home space.
- Adapted security use case 2
  (a) Description: Depending on the authentication level, different information is projected to the environment with the LumEnActive system or different actions can be taken. Level <= 2: Controlling of home appliances not allowed and information shown. Level 3: Controlling of specific (e.g., humidity, temperature) home appliances allowed.
  (b) Solution: Rule – Monitor, Analyze, Plan (Security, Dynamic). The authentication service is adapted to the situation using dynamic security rules, so they can be changed at run-time according to needs (see the taxonomy of rules (Fig. 3).

Newly created use cases:

- Use case 1: When a person enters to the room, the system displays the top 5 energy consuming devices on the wall. Rule: monitor (Context, Static, and Dynamic). Example rules for this scenario as IF-THEN-ELSE: 1) IF motion sensor shows movement or user(s) is/are authenticated, THEN display top-5 energy consumers, ELSE do nothing.
- Use case 2: When there are several persons in the room, the system displays general energy saving tips. Rule: analyze (Context, Static, Dynamic).
- Use case 3: When there is no activity in the room, nothing is displayed. Rule: monitor (Context, Dynamic).
- Use case 4: When a user controls some device through the SIB, the system displays the current energy consumption of that device by projecting the information onto a surface near the device. Rule: analyze (Context, Dynamic).

These new use cases do not describe where the logic must reside. One use case can contain multiple rules that complete the behavior of that particular use case. For example, in (newly created) Use case 1 we decided that the LumEnActive system should contain more static rules, e.g., behavior for different use cases. The LumEnActive system also contains a static rule to show Use case 4 for 30 seconds and return to the previous use case; other use cases are shown until the next one is requested to be shown. Use cases 1 to 2 display at a predetermined position in the wall; Use case 4 displays above the device that is controlled.

B. INCREMENTAL TESTING OF SMART-SPACE APPLICATIONS

Testing of intelligent applications can be coarsely classified as follows:

- GUI application – End-user application that combines static and/or dynamic rules, normal programming logic, and GUI(s).
- Legacy adapter – Glue software that uses static or dynamic rules or just makes the legacy software available in the semantic information broker without dynamicity.
- Dynamic agent – Agent that is fully controlled and guided by the dynamic rules assigned for it.

Testing the functionality of agents is a rigorous task because most of the testing is conducted with asynchronous data that is being produced and consumed from the SIB. For example, if a planning rule makes a trade-off between performance and security, to avoid reconfigurations at rapid rate it should react only when it has all the necessary information. The testing phase is divided into three main activities (see Fig. 4):

- Testing an SSA with the test cases in the test environment, i.e., unit and integration testing according to steps 1-4.

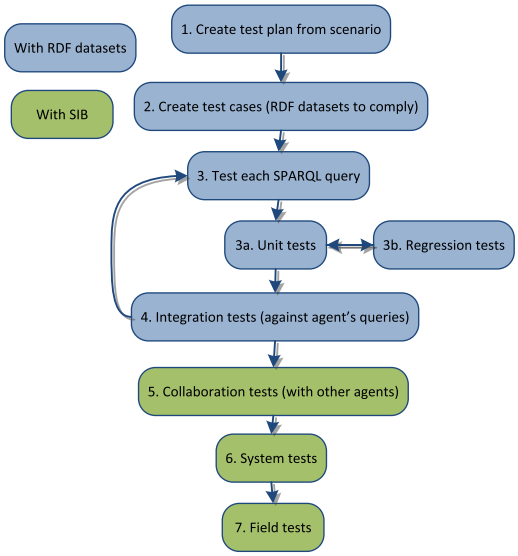

FIGURE 4. Testing process of intelligent applications.

- Testing an SSA as part of a smart space, i.e., collaboration and system testing according to steps 5-6.
- Testing an SSA by end-users in field tests according to step 7.

The Interactive Quality Visualization (IQVis) tool can be used in various steps to visually see the interactions between different aspects of the tested software [49]. With the IQVis tool, the system qualities can be monitored and viewed at run-time. For example, different teams can see how different qualities vary and how trade-offs are made at run-time in a given scenario. With the help of configurable monitoring (e.g., [42]), the IQVis tool can be exploited even further.

The testing process (for dynamic agents) (Fig. 4) includes the following steps:

1. The testing process starts by creating a *test plan* and test cases following the defined usage scenarios.
2. *RDF datasets* are then defined according to the usage scenarios to cover the functionality defined in use cases with test suites (i.e., many test cases for one use case). The RDF datasets help in automating the testing of single queries; they are attached to the specific test cases. The RDF dataset constitutes a single moment in time, including all the context information related to that query. It is essentially a snapshot of the context in the smart space at a given point in time. The RDF datasets provide static semantic information to be used with automated test cases to cover the basic functionality. A more advanced utilization of datasets is to start with the context needed in a use case and execute SPARQL queries that modify the dataset, and then execute follow-on queries that utilize information modified in the previous query.
3. The *test cases* made for a single query and a single agent are then executed using a query engine supporting SPARQL and RDF, e.g., Apache Jena [50]. Thus, the

unit testing (3a) uses different RDF datasets to test the queries with different contexts. A test harness is recommended in this step; test-driven development is a good practice for testing individual queries, because the test cases can be done according to the usage scenarios, reusing them from the AKB, and therefore, the development process is sped up with verified scenarios and test cases. The RDF datasets are also reused in regression testing (3b). Regression tests should be executed, e.g., when i) ontology is changed, ii) a new query is introduced, or iii) existing queries are refactored to verify that the modifications are as intended.

4. The *integration testing* is made in the bottom-up fashion because bugs in queries are discovered faster. Testing software components that work with live data is tedious. They are not the best possible type of components to test with unit testing, but have benefits in discovering the easiest and most common bugs before moving to live testing the system. Therefore, different datasets are used to test how the queries affect each other. A test harness is recommended in this phase also. Moreover, the SIB can also be used a testing platform. SIB is often required because of its subscribe feature.
5. In *collaboration testing*, multiple agents are combined and tested to see whether their interactions are correct. The tests are made in the actual execution environment to see how the application functions in a real environment. This step is closer to integration testing than system testing. If the combined agents that form an intelligent application do not include a GUI, a separate unit test might be unnecessary, but the logic of an agent has to be tested in the form of rules. If the combined agents have a GUI and/or static logic parts, unit tests must be conducted before starting the integration testing with dynamic parts of the intelligent application. The usual case is that static software components are enhanced by adding a dynamic part that enables some type of intelligence, e.g., an ability to adapt the application behavior according to measured context information. Moreover, the GUI is also to be tested.
6. In the *system testing*, the system as a whole with all the relevant applications, agents and platform services functioning together are tested in a real environment, i.e., in the selected smart space(s).
7. Finally, the SSPA is tested in *field tests*, where real end-users (typically non-ICT professionals) use the services in their processes and activities. This phase needs its own test plan for collecting users' experiences and analysis of the collected feedback data.

The incremental testing phase for smart-space applications exploits the knowledge base and tools as follows:

- *Assets* – Previously created test cases and user stories can be utilized from the knowledge base in the creation of new test cases. Accompanying snapshots of smart-space situations can also be utilized to drive the smart-space application(s) into the correct stage for testing individual requirements.
- *Tooling* – Both Atlassian JIRA [35] and Agilo for trac [36] can be used for the testing process to identify discovered bugs. In addition, BugZilla [51] can be used to notify development teams of bugs found. For the management of test cases, Tarantula [52] can be used to describe the test cases to be used in scenarios and also reuse existing cases.

Example – Enhancing an existing smart environment

Here we show how the testing was conducted in this laboratory example. We give examples of datasets used to unit test the required SPARQL queries of one example use case. Our purpose is not to show every step in detail, but to give a rough view of the process. For this reason, we concentrate on testing the dynamic application and leave out testing static rules.

The reused use cases (defined in the previous section) had already been tested and were therefore usable directly from the AKB without extensive testing. Examples of security analysis rules and measures used for authentication are presented in [38]. Their functionality with our newly created use cases had to be verified in the collaboration and system tests.

Following the testing process described earlier, we now present how the parts of new Use case 1 were tested.

1. We defined the test cases for the use case. We needed to test the functionality for detecting the users, and the functionality for showing the top 5 energy consumers, as follows:

Test suite 1 for testing detecting user presence:

- Test case 1: Motion sensor detects no movement and there are no logged-in users.
- Test case 2: Motion sensor detects no movement and there are logged-in users.
- Test case 3: Motion sensor detects movement and there are no logged-in users.
- Test case 4: Motion sensor detects movement and there are logged-in users.

Test suite 2 for testing showing top five energy consumers:

- Test case 1: Show Use case 1 and list ten devices with different energy consumptions in the room. The information of the topmost five energy consumers is sent to LumEnActive system according to the domain ontology.
- Test case 2: Same as the Test case 1, but show Use case 1 is false.
- Test case 3...N: Variable energy consumptions.

Test suite 3 for testing the LumEnActive side with RDF datasets consisting of five different devices.

2. We realized the test cases. Two example Test case RDF N3 [53] datasets for Use case 1 (without prefixes) containing the expected results that the query should return (for test reports) are presented below:

#Simplified RDF dataset for Test Case 1.1: Motion sensor detects no movement and there are no logged in users.

:sensor rdf:type :MotionSensor.
:sensor :hasMovement 0.
d:Space :hasLoggedInUsers 0.
:TestCase :ExpectedResultObject :False.

#Simplified RDF dataset for Test Case 1.2: Motion sensors detects no movement and there are logged in users.

:sensor rdf:type :MotionSensor.
:sensor :hasMovement 0.
d:Space :hasLoggedInUsers 1.
:TestCase :ExpectedResultObject :True.

3. We realized a rule to a SPARQL query, e.g., as a stub that does not pass all the tests. This enabled us to run the query against different datasets.

#StubQuery: Detect movement in the room (without prefixes)

```
CONSTRUCT {
    :Agent :showTopEnergyConsumers :True.
}WHERE{
    ?sensor rdf:type :MotionSensor.
    ?sensor :hasMovement 1.
}
```

As seen, the above query lacks a check for logged-in users. This bug can be discovered in the unit tests, if they are exhaustive enough. A motion sensor does not show movement if people do not move enough, even though users are present, and therefore information on logged-in users needs to be shown. The final rule is presented below. It has passed all the unit tests made for the use case. The SPARQL query also requires other properties, e.g., hasLoggedInUsers, which is reasoned in another rule, but is not presented here, because it was developed and tested earlier. A more difficult bug to discover is that, e.g., Use case 3 (also Use case 2) uses the same information as this query for Use case 1 and shows energy tips when there are multiple users. We therefore had to add a check on the number of users in this query. This type of bug is discovered at the latest in integration or collaboration tests, where the behavior overlaps. The final query for the Use case 1 is presented below:

#Query 1.1: Detect movement in the room final version

```
PREFIX :     <http://www.SOFIA.net/Context#>
PREFIX d:    <http://www.SOFIA.net/Domain#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  CONSTRUCT {
    d:Agent d:showScenario 1.
  }WHERE{
    ?sensor rdf:type :MotionSensor;
            :hasMovement ?mov.
    d:Space :hasLoggedInUsers ?users.
    BIND(?mov + ?users AS ?res).
    FILTER(?res >= 1).
    FILTER(?users < 2).
  }
```

After unit tests, one agent's functionality is tested with the semantic information broker. Thereafter, the system tests can be performed with the actual execution environment from the end-user perspective to ensure the correct behavior of different use cases.

V. EVALUATION RESULTS

The PSE approach has been tested in different test-beds. Parts of the PSE approach have been evaluated in 1) different use cases with simple technology and feature demonstrations to 2) pilots of multiple smart spaces communicating with each other and 3) full industrial use cases with industrial partners. The groups of users testing the approach have evolved from developers and few end-user testers in laboratory cases to real end-user testers of the applications developed by the approach, and also end-users creating new smart ad-hoc applications themselves with their mobile devices. Testing of the PSE approach most recent at this writing was conducted in an end-user programming use case [47], [48] and simultaneously in a contract project with an industrial customer to build a semantic facility-management system, which was trialed on School premises at the city of Oulu [54]. This last industrial use case, especially, utilized all of the building blocks of PSE and AKB to introduce quickly a semantic facility data management system with which users can interact. Table 5 summarizes the selected test cases from early individual demonstrations (from 12 different demonstrations scenarios (rows 1...12 in Table 5) in which building blocks of PSE were tested) to testing of the entire approach in the pilot and follow-up industrial projects. The column 'Case, year' describes the name of the use case and the year it was conducted, 'Lessons learned' gives what was learned about application development in the experiment, 'New knowledge pieces' notes what new knowledge or assets were introduced into the AKB with this use case, and 'Evaluated part' stands for the part of the PSE approach that is evaluated. The numbering is as follows:

1. Smart-space Application Innovation
2. Smart-space Application Co-creation
3. Behavior-Driven Application Development
4. Incremental Testing of Smart-space Applications

Table 5 summarizes how the PSE approach has been continuously developed during our smart-space application development over the 4-5 year period. PSE provides the following advantages:

- Because the PSE approach relies heavily on standard technologies and especially semantic web technologies and ontologies, it has been very useful in integrating different types of systems.
- The advantage is that the approach can be used basically with any type of dynamic semantic web technology. We have utilized it with smart spaces and SIB technologies, but it has been used also without them, e.g., with the Virtuoso semantic database. Therefore, this is partially a technology-independent approach.
- The PSE approach does not compete with agile methods

TABLE 5. Summary of the evaluation of the PSE: lessons learned and knowledge created in each evaluation activity.

Case, year	Evaluated part	Lessons learned	New knowledge pieces	Evaluation type
Smart Greenhouse demo, 2009 [37]	1, 2, 4	Overall definition of a smart-space application. Applicability of existing software engineering methods and ontology orientation in smart-space application development.	Usage scenarios and use cases. Initial context, security, performance ontologies, and domain ontology for sensors and actuators.	Laboratory case
Risk-based security adaptation in a greenhouse, 2010 [55]	3	Development of information security ontology based on the knowledge of measurement science and information security definitions. Combining and applying them in a smart-space application with risk level measuring and simple adaptation rules.	Security ontology, security adaptation at run-time	Laboratory case
Interactive Quality Visualization (IQVis) tool for testing/verifying quality adaptations, 2010 [49]	4	How to visualize the structure of the smart space. How to test smart-space applications with visualizations when adaptations occur and tradeoffs are made between, e.g., security and performance. The tool enabled to monitor and view the changes in the system's security performance.	Visualization ontology, IQVis tool	Laboratory case
User authentication in Smart Spaces, 2011 [38]	1, 2	Developing Information Security Measuring Ontology (ISMO) and using it to measure user authentication strength in smart spaces. Creating first versions of rules by analysis models expressed in natural-language English.	Security ontology, authentication use cases and adaptation use cases. Analysis rules related to user authentication.	Laboratory case
Cross-domain scenario between personal and home space, 2011 [56]	2, 3	Utilizing ontology and model-driven development in smart-space application development. How to create an application with a visual editor and create reusable rules for context that can be reused in the model.	Reusable models for context situations	Laboratory case
Reusable Knowledge Processors, 2011 [42]	3, 4	Creating reusable pieces of software for smart-space applications. Run-time configurations of smart-space applications were also tested. This enables the applications to be configured at run-time to, e.g., monitor changing situations.	Design patterns and implementation for reusable agents and monitor framework	Laboratory case
ARTEMIS Technology Conference, Bologna, 2011 [59]	1, 2, 3, 4	Integrating different scenarios from different stakeholders together to form a richer scenario, and integrating work between teams in different international locations.	Experience / instructions on working with multiple teams and locations. Ontology integration.	End-users
ARTEMIS/ITEA2 Co-Summit, Helsinki, 2011 [58]	1, 2, 3, 4	Demonstrating smart spaces remotely using HD-video. Collecting feedback from end-users familiar with the development of smart-space applications.	Full application development.	End-users
SOFIA pilot, Innovation Kitchen Oulu, 2011, described in [28], one scenario video in [57]	1, 2, 3, 4	Creating a user experience based on multiple smart spaces. Feedback from ordinary users (i.e., not familiar with smart-space application development) using the smart-space applications via mobile phones.	This pilot integrated earlier demonstrations as a whole pilot with multiple smart spaces.	End-users
End-user programming of smart-spaces applications, 2011 [47]	3, 4	How to transfer smart-space application development from professional development to end-user development. How do SIB implementations perform when a mobile phone is running multiple interpreted smart-space applications created by an end-user?	First prototype of end-user programming framework for mobile devices. Example reusable end-user applications.	Laboratory case, developers as testers
End-user programming of smart-spaces applications, 2012 [48]	1, 2, 3, 4	Describing two ways to create end-user applications with mobile devices: command-oriented and goal-oriented.	Enhanced end-user programming framework.	End-users
Situation-based and self-adaptive applications, 2012 [28]	1, 2, 3, 4	How to create dynamic context-aware smart-space applications using rules. Taking into account tradeoffs between different quality levels, especially in dynamic scenarios.	New rules, adaptation behavior, use cases.	Laboratory case
Smart Operation and Maintenance of indoor Environments (SOME), 2012	1, 2, 3, 4	Innovation of a new smart-space application into an existing environment to enrich facility management services. Integration of multiple different systems into the smart space. Introducing indoor environment conditions to actual facility users.	Indoor environment ontology. Business innovation, scenarios for facility management.	End-users of the field test environment.
Enhanced SOME applications, 2013 [54]	1, 2, 3, 4	Addressing user experience through user feedback loop to the facility managers. Visualizing the spaces of the facility to users using 2.5D visualization.	User feedback ontology, facility management ontology.	End-users of the field test environment.

such as Scrum; instead, it can be used together with them. The goal is the same – to ease the development complexity by providing the developers tools and knowledge that can be reused case after case.

- The benefits of the PSE approach grow after smart-space development iterations, because the knowledge base gains new information. Therefore, existing assets are exploited by combining services and components in application development. Thus, PSE generates cost savings.
- Innovation is independent on location, distance and missing expertise. For example, a multi-national group of persons was able to set up an innovative application to demonstrate a new application for an international audience based on a phone call and a week of coding labor. Additionally, end-users who evaluated the pilot addressed the novelty of the implemented smart environment and saw real impact on easing everyday life and having potential access into new markets. Thus, PSE encourages innovation and collaboration in an effective way.
- Integration of different solutions (i.e., networks, servers, systems, and applications) is straightforward and non-laborious, and pieces of assets are able to interoperate. Due to the AKB (and especially ontologies and mechanisms for handling them), interoperability of assets is achieved. Thus, the use of the PSE with the AKB saves time and effort in all development phases – from scenario specification to application testing.
- Domain ontologies are valuable tools for teaching novices the specialty of a particular domain. Thus, the AKB also provides tools for skills development, not only application development.
- The development process is much faster when there are earlier scenarios available for reuse, either directly or with small modifications/additions. In the case example, we had earlier scenarios, test cases, and rules made for the security domain and this reduced the time to develop this example case. Test-driven development was applied because it fits well into creating dynamic rules from the scenario descriptions.

In spite of many advantages there are still limitations and issues that need further improvement:

- Identification of business patterns has not progressed enough. Therefore, future studies should focus on business ecosystems of smart environments. Particularly, stakeholder roles, business model elements and value co-creation methods should be explored both in open and closed smart environments.
- Smart-space users are active and willing to improve their experiences with personalized applications. Therefore, the programming and visualization facilities of the end-user programming tool needs to be extended

and improved. Additionally, the use of open data together with private data needs further study.

- Description methods for defining application scenarios need to be systematized so that scenario descriptions can be reused more extensively. Visualization techniques could be used to illustrate and share the application ideas among different stakeholders.
- Evolution of ontologies needs new technologies: the evolution should be handled at design time, i.e., in the software development process, and as an operative activity of the dynamic smart environment. This last is challenging because ontology alignment and interweaving is to be handled at run-time, if not in real time.
- The search possibilities of reusable artifacts should be addressed more deeply, even though Stylebase can be utilized for that purpose. The descriptions and context of the artifacts should be very informative.
- Prioritization of run-time qualities (e.g., security, performance) should be more easily addressable in the development phase and also in the testing phase. At the stage of this work, prioritization is mostly decided in iterations between the development teams. Being able to configure these at run-time would be useful for advanced end users. This requires codified design decisions to be provided by the AKB.

Rule-based adaptation at run-time is very difficult to test, especially when we have multiple non-functional qualities and context situations to be taken into account. Testing these types of situations requires further research and tools to be able to cover all of the possible outcomes of adaptations in one agent and adaptations taking place in multiple agents at once. This problem is not specific to PSE (and there exists much research in rules, race conditions, and test coverage, etc.), but it poses problems for the PSE approach as well.

VI. CONCLUSION

The new technologies such as semantic web and autonomic computing provide enriched means to solve problems in the development of intelligent smart-space applications. In this paper, the PSE approach was introduced for dealing with these issues by applying model and ontology orientation for describing design knowledge, which is further exploited as pieces of artifacts at run-time for creating intelligent applications. The use of the PSE approach was illustrated by a laboratory example and validated in an incremental way in laboratory and industrial case studies. First, the main building blocks were tested separately in demonstrations. Second, the tested building blocks were integrated incrementally by developing three releases of a cross-domain pilot study that covered four different smart spaces: smart personal spaces, smart home, smart office and smart city. The applications developed were also evaluated by end users in three different test environments. Finally, the PSE was applied to the development of smart operation and maintenance applications tested by end users in the field. In summary, a faster development process was gained through reusable, tested and verified assets developed based on the design knowledge of the AKB.

REFERENCES

[1] E. Ovaska, T. Salmon Cinotti, and A. Toninelli, "The design principles and practices of interoperable smart spaces," in *Advanced Design Approaches to Emerging Software Systems: Principles, Methodologies and Tools*. Hershey, PA, USA: IGI GLobal, 2012, pp. 18–47.

[2] (2013, Sep. 26). *Internet of Things* [Online]. Available: http://en.wikipedia.org/wiki/Internet_of_Things

[3] (2013, Sep. 26). *Big Data* [Online]. Available: http://en.wikipedia.org/wiki/Big_data

[4] E. Ovaska, L. Dobrica, A. Purhonen, and M. Jaakola, "Exploration of technologies for autonomic dependable service platforms," in *Proc. 6th Int. Conf. Softw. Database Technol.*, Seville, Spain, 2011, pp. 115–124.

[5] (2013, Sep. 26). *Crowdsourcing* [Online]. Available: http://en.wikipedia.org/wiki/Crowdsourcing

[6] C. Gravino, M. Risi, G. Scanniello, and G. Tortora, "Do professional developers benefit from design pattern documentation? A replication in the context of source code comprehension," in *Proc. 15th Int. Conf. MODELS*, 2012, pp. 185–201.

[7] A. Soylu, P. De Causmaecker, D. Preuveneers, Y. Berbers, and P. Desmet, "Formal modelling, knowledge representation and reasoning for design development of user-centric pervasive software: A meta-review," *Int. J. Metadata, Semantics Ontol.*, vol. 6, no. 2, pp. 96–125, 2011.

[8] (2013, Jan. 30). *OWL* [Online]. Available: http://www.w3.org/TR/owl2-overview/

[9] (2013, Jan. 30). *Unified Modeling Language (UML) 2.4.1*, OMG, New York, NY, USA [Online]. Available: http://www.omg.org/spec/UML/2.4.1/

[10] (2012, May 25). *SPARQL Query Language for RDF, W3C Recommendation* [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[11] "MDA guide version 1.0.1.," in Document No. omg/2003-06-01, 2003.

[12] *Systems and Software engineering—Architecture Description*, ISO/IEC/IEEE, Standard 42010:2011, 2011.

[13] H. Gomaa, *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architecture*. Redwood City, CA, USA: Addison-Wesley, 2005.

[14] J. Osis and E. Asnina, *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, J. Osis and E. Asnina, Eds. IGI Global, 2010, p. 518.

[15] J. C. Nieves *et al.*, "Coordination, organisation and model-driven approaches for dynamic, flexible, robust software and services engineering," in *Service Engineering*. New York, NY, USA: Springer-Verlag, 2010, pp. 85–115.

[16] M. Bencomo, A. Bennaceur, P. Grace, G. Blair, and V. Issarny, "The role of models@run.time in supporting on-the-fly interoperability," *Computing*, vol. 95, no. 3, pp. 165–190, 2013.

[17] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Autonomous Adaptive Syst.*, vol. 4, no. 2, article 14, 2009.

[18] A. Soylu, P. De Causmaecker, and P. Desmet, "Context and adaptivity in pervasive computing environments: Links with software engineering and ontological engineering," *J. Softw.*, vol. 4, no. 9, pp. 992–1013, 2009.

[19] C. Bettini *et al.*, "A survey of context modelling and reasoning techniques," *Pervas. Mobile Comput.*, vol. 6, no. 2, pp. 161–180, 2010.

[20] A. Achilleos, K. Yang, and N. Georgalas, "Context modelling and a context-aware framework for pervasive service creation: A model-driven approach.," *Pervas. Mobile Comput.*, vol. 1, no. 6, pp. 281–296, 2010.

[21] M. Kassab, O. Ormandjieva, and M. Daneva, "An ontology based approach to non-functional requirements conceptualization," in *Proc. 4th Int. Conf. Softw. Eng. Adv.*, 2009, pp. 299–308.

[22] E. Niemelä, J. Kalaoja, and P. Lago, "Towards an architectural knowledge base for wireless service engineering," *IEEE Trans. Softw. Eng.*, vol. 31, no. 5, pp. 361–379, May 2005.

[23] E. Niemelä and A. Immonen, "Capturing quality requirements of product family architectures," *Inf. Softw. Technol.*, vol. 49, 11–12, pp. 1107–1120, 2007.

[24] E. Ovaska, A. Evesti, K. Henttonen, M. Palviainen, and P. Aho, "Knowledge based quality-driven architecture design and evaluation," *Inf. Softw. Technol.*, vol. 52, no. 6, pp. 577–601, 2010.

[25] M. Palviainen, A. Evesti, and E. Ovaska, "The reliability estimation, prediction and measuring of component-based software," *J. Syst. Softw.*, vol. 84, no. 6, pp. 1054–1070, 2011.

[26] A. Evesti, E. Niemelä, K. Henttonen, and M. Palviainen, "A tool chain for quality-driven software architecting," in *Proc. Softw. Product Line Conf. SPLC*, Sep. 2008, pp. 360–360.
[27] E. Ovaska, "Ontology driven piecemeal development of smart spaces," in *Proc. 1st Int. Conf. Ambient Intell.*, 2010, pp. 148–156.
[28] S. Pantsar-Syväniemi, A. Purhonen, E. Ovaska, J. Kuusijärvi, and A. Evesti, "Situation-based and self-adaptive applications for the smart environment," *J. Ambient Intell. Smart Environ.*, vol. 4, no. 6, pp. 491–516, 2012.
[29] A. Evesti, J. Suomalainen, and E. Ovaska, "Architecture and knowledge-driven self-adaptive security in smart spaces," *Computers*, vol. 2, no. 1, pp. 34–66, 2013.
[30] (2013, Sep. 27). *QADA Tools* [Online]. Available: http://www.vtt.fi/sites/qada/qada_tools.jsp?lang=en
[31] K. Henttonen and M. Matinlassi, "Open source based tools for sharing and reuse of software architectural knowledge," in *Proc. Joint Working IEEE/IFIP Conf. Softw. Archit. (WICSA) 3rd Eur. Conf. Softw. Archit. (ECSA)*, Cambridge, MA, USA, Sep. 2009, pp. 41–50.
[32] R. C. de Boer and H. van Vliet, "Experiences with semantic Wikis for architectural knowledge management," in *Proc. 9th Working IEEE/IFIP Conf. Softw. Archit., WICSA*, 2011, pp. 32–41.
[33] Conante. (2012, Jun. 19). *LumEnActive* [Online]. Available: http://www.conante.com/products/lumenactive/?lang=en
[34] P. Kruchten, "An ontology of architectural design decisions in software-intensive systems," in *Proc. 2nd Groningen Workshop Softw. Variability Manag.*, Groningen, The Netherlands, 2004.
[35] (2013, Dec. 4). *Atlassian JIRA* [Online]. Available: https://www.atlassian.com/software/jira
[36] (2013, Dec. 4). *Agilo for Trac* [Online]. Available: http://www.agilofortrac.com/
[37] A. Evesti, M. Eteläperä, J. Kiljander, J. Kuusijärvi, A. Purhonen, and S. Stenudd, "Semantic information interoperability in smart spaces," in *Proc. 8th Int. Conf. Mobile Ubiquitous Multimedia (MUM)*, 2009, pp. 158–159.
[38] A. Evesti, R. Savola, E. Ovaska, and J. Kuusijärvi, "The design, instatiation and usage of information security metrics ontology," in *Proc. 2nd Int. Conf. Models Ontol. Based Des. Protocols, Archit. Services*, Budapest, Hungary, 2011, pp. 1–9.
[39] F. García *et al.*, "Towards the consistent terminology of software measurement," *Inf. Softw. Technol.*, vol. 48, no. 8, pp. 631–644, 2006.
[40] A. Herzog, N. Shahmehri, and C. Duma, "An ontology for information security," in *Techniques and Applications for Advanced Information Privacy and Security: Emerging Organisational*. New York, NY, USA: Ethical and Human Issues, 2009.
[41] J. Honkola, H. Laine, and R. Brown, "Smart-M3 interoperability platform," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Riccione, Italy, 2010, pp. 1041–1046.
[42] J. Kuusijärvi and S. Stenudd, "Developing reusable knowledge processors for smart environments," in *Proc. IEEE/IPSJ 11th Int. Symp. Appl. Internet*, Munich, Germany, 2011, pp. 286–291.
[43] (2014, Feb. 25). *Protégé. A Free, Open-Source Ontology Editor and Framework for Building Intelligent Systems* [Online]. Available: http://protege.stanford.edu/
[44] M. Palviainen and A. Katasonov, "Towards ontology-driven development of applications for smart environments," in *Proc. 8th PERCOM Workshops*, 2010, pp. 696–701.
[45] (2013, Dec. 4). *Jenkins CI. An Open-Source Continuous Integration Tool* [Online]. Available: http://jenkins-ci.org/
[46] (2013, Jan. 30). *RDF Vocabulary Description Language, RDF-Schema* [Online]. Available: http://www.w3.org/TR/rdf-schema/
[47] M. Palviainen, J. Kuusijärvi, and E. Ovaska, "Framework for end-user programming of cross-smart space applications," *Sensors*, vol. 12, no. 11, pp. 14442–14466, 2012.
[48] M. Palviainen, J. Kuusijärvi, and E. Ovaska, "A semi-automatic end-user programming approach for smart space application development," *Pervas. Mobile Comput.*, 2013.
[49] J. Kuusijärvi, "A demo on using visualization to aid run-time verification of dynamic service systems," in *Proc. 3rd Int. Conf. Softw. Test., Verification, Validation Workshops (ICSTW)*, Paris, France, 2010, pp. 319–324.
[50] (2013, Dec. 4). *Apache Jena* [Online]. Available: http://jena.apache.org/
[51] (2013, Dec. 4). *BugZilla* [Online]. Available: http://www.bugzilla.org/
[52] (2013, Dec. 4). *Tarantula* [Online]. Available: http://www.testiatarantula.com/
[53] (2014, Feb 25). *Notation 3: A Readable RDF Syntax* [Online]. Available: http://www.w3.org/TeamSubmission/n3/
[54] I. Niskanen, A. Purhonen, J. Kuusijärvi, and E. Halmetoja, "Towards semantic facility data management," in *Proc. 3rd Int. Conf. Intell. Syst. Appl.*, 2014.
[55] A. Evesti and E. Ovaska, "Ontology-based security adaptation at run-time," in *Proc. 4th IEEE Int. Conf. Self-Adaptive Self-Oranizing Syst. (SASO)*, Oct. 2010, pp. 204–212.
[56] S. Pantsar-Syväniemi, J. Kuusijärvi, and E. Ovaska, "Context-awareness micro-architecture for smart spaces," in *Advanced in Grid and Pervasive Computing*. Decatur, GA, USA: GPC, 2011.
[57] (2011, Sep.). *Smart Door Video* [Online]. Available: www.youtube.com/watch?v=anRW0y2r1Q0
[58] (2011). *Artemis ITEA2 Co-Summit* [Online]. Available: http://www.artemis-ia.eu/cosummit2011_home
[59] (2011). *Artemis Technology Conference* [Online]. Available: http://www.artemis-ia.eu/atc_2011_home

EILA OVASKA received the Ph.D. degree from the University of Oulu in 2000. Before 2000, she was a Software Engineer, a Senior Research Scientist and a Leader with the Software Architectures Group, VTT Technical Research Centre of Finland. Since 2001, she has been a Research Professor with VTT and an Adjunct Professor with the University of Oulu. Her current areas of interest are service architectures, self-monitoring, and self-adaptation. She has acted as a Workshop and Conference Organizer and as a Reviewer for scientific journals and conferences. She has co-authored over 130 scientific publications. She is a member of the IEEE and IEEE Computer Society.

JARKKO KUUSIJÄRVI received the B.Sc. (Tech.) and M.Sc. (Tech.) degrees from the University of Oulu in 2008 and 2010, respectively. Since 2010, he has been a Research Scientist with the VTT Technical Research Centre of Finland. His current areas of research interests include mobile applications, ontology-driven software engineering, security visualization, and cyber security.