Using SPL to develop AAL systems based on self-adaptive agents

Inmaculada Ayala, Mercedes Amor, and Lidia Fuentes

Universidad de Málaga, Departamento de Lenguajes y Ciencias de la Computación
Campus de Teatinos s/n, 29071 Málaga, España
{ayala,pinilla,lff}@lcc.uma.es

Abstract. One of the most important challenges of this decade is the Internet of Things (IoT) that pursues the integration of real-world objects in Internet. One of the key areas of the IoT is the Ambient Assisted Living (AAL) systems, which should be able to react to variable and continuous changes while ensuring their acceptance and adoption by users. This means that AAL systems need to work as self-adaptive systems. The autonomy property inherent to software agents, makes them a suitable choice for developing self-adaptive systems. However, agents lack the mechanisms to deal with the variability present in the IoT domain with regard to devices and network technologies. To overcome this limitation we have already proposed a Software Product Line (SPL) process for the development of self-adaptive agents in the IoT. Here we analyze the challenges that poses the development of self-adaptive AAL systems based on agents. To do so, we focus on the domain and application engineering of the self-adaptation concern of our SPL process. In addition, we provide a validation of our development process for AAL systems.

Keywords: Agents, SPL, IoT, Variability modeling

1 Introduction

One of the most important challenges of this decade is to integrate real-world objects in Internet, called the Internet of Things (IoT)[1]. An IoT system is composed of various types of devices like sensors and home appliances, which are now connected to the Internet using different technologies. These devices can support various types of systems being the Ambient Assisted Living (AAL) one of its key areas. The goal of the AAL is to extend the time people can live in their preferred environment by increasing their autonomy and self-confidence[16].

The development of the AAL in the IoT is challenging since it requires to manage hardware and software technologies which are continuously evolving. From the user point of view, the application should consider factors that facilitate its acceptance and adoption, such as the perceived ease of use and usefulness [19]. This is not an easy task, since AAL systems have reached a level of complexity where the skills required by the user to keep the system running is high. A solution to this issue is to reduce the interaction between the user and the devices of the system. However, these devices are usually subject to changes that

can affect their behavior, which could require user intervention. Also, system requirements could evolve at runtime, implying the addition of a new device by instance to be able to monitor a temporal disease. So, in our opinion, AAL systems should be self-adaptive [11], which implies to be able to adapt themselves autonomously to context changes resulting from device failure and the addition or loss of devices and services. In order to be self-adaptive, AAL systems should be endowed with self-adaptation capacities, which will be transparent to the final user.

The distributed nature of the IoT, autonomy, awareness and social behavior make software agents a suitable choice for the development of self-adaptive AAL systems. To implement an autonomous behavior for the nodes that compose an AAL system, agents must be embedded in these nodes. This is especially important with regard to the self-adaptation of the system. AAL systems can be composed of hundred of nodes that are distributed and interconnected using different means. So, traditional solutions for self-adaptation are centralized or are based on a fixed number of self-managers [11], which can result inadequate and non-viable for the variability of the IoT.

We have already proposed a Multi-Agent System (MAS) approach, called Self-StarMAS [2], where agents are embedded in IoT devices, also able to manage themselves. Using Self-StarMAS, we can develop the AAL application as a community of self-adaptive agents. Self-StarMAS has been previously applied to develop applications of the IoT [2]. However, since Self-StarMAS does not have a process to handle its variability, the implementation requires the development of different versions of these agents considering different devices, levels of cognitive capacity and degrees of self-adaptation. This process is important to guarantee that agent configurations meet application requirements, and the inter-operation between the different agents that compose the MAS is feasible.

In our opinion, current agent development processes are not adequate to develop AAL applications using IoT technology because they lack mechanisms to deal with the variability present in the IoT. An accepted solution to model variability is the Software Product Line (SPL) technology [6]. The benefits of SPLs are given by the reusability of the features common and variables, embodied in architectural elements during the development of a new product or configuration. This technology is being applied to the development of MASs. The integration of both technologies is known as MAS-PL (Multi-Agent System Product Lines) [17]. However, these MAS-PL approaches do not focus on solutions for the IoT.

To overcome these limitations, we have proposed an SPL process for the development of self-adaptive agents in the IoT [3]. Our starting point was Self-StarMAS, which was refactored to enable the development of MASs using an SPL. The process was defined using the Common Variability Language (CVL) [8], a domain-independent language for specifying and resolving variability. Here we analyze the challenges that poses the development of self-adaptive AAL systems based on agents. This analysis will be the starting point of the domain and application engineering of the self-adaptation concern of our SPL process. In addition, we provide a validation of our development process.

This paper is structured as follows: Section 2 presents our approach, Section 3 overviews CVL, Section 4 gives details of works related to our proposal, Sections 5 and 6 describes our SPL process, which is validated in Section 7. Our paper concludes with conclusions and future work.

2 Our approach

This section reviews the specific challenges in the development of agent-based AAL systems. General challenges in the development of AAL can be found in different works [16].

The first challenge that we identify is **to manage the variability of agents for AAL systems (C1)**. Agents pose advantages for the development of AAL systems [21, 10, 15]. However, current solutions do not allow the development of heterogenous MAS, because they are restricted to a single agent platform and network technology [21], which makes difficult the integration of new technologies. A development process for agent-based AAL systems must manage this variability according to application requirements.

Our second challenge is **to manage software and hardware dependencies (C2)**. Until now, hardware and software dependencies were not important in the agent paradigm since MASs consists in the interaction of a population of homogeneous agents. When agents are embedded in devices that compose the AAL application, these dependencies must be taken into consideration (e.g. some network technologies are only available in specific devices). These dependencies usually go unnoticed by software architects until the implementation and deployment stages, but they should be considered and incorporated earlier in the development process to avoid incompatibilities.

The third challenge that we identify is **the support for different degrees of self-adaptation (C3)**. What can be adapted by the agent and how can be done depend on specific features and functions of the device where the agent is embedded. This leads us to consider different degrees of self-adaptation, which are influenced by the requirements of the AAL system and features of the device where the agent is embedded. The consideration of different degrees of self-adaptation must also be incorporated from the early stages of the process.

The combination of SPL and self-adaptive agents makes possible to address these challenges. Our development process (see Fig. 1) follows the two life cycles schema of SPL processes that separates domain (DomE) and application (AppE) engineering. The DomE of our process (defined in [3]) analyzes the SPL for MASs in the IoT as a whole to define and produce the commonality and the variability of the SPL (see 2). The second process, the AppE, involves creating and configuring the architecture of agents, which are built by reusing domain artifacts and exploiting SPL engineering. The use of SPL enables the management of the variability presented in the AAL domain. Additionally, SPLs provide specific mechanisms to manage the dependencies between the different concerns. Self-adaptation for AAL systems can be incorporated in SPL, so it can be considered from the early stages of the development process taking into account different

degrees of it. In the following, we focus on the DomE of the self-adaptation concern of the agents and the AppE of a self-adaptive AAL system.

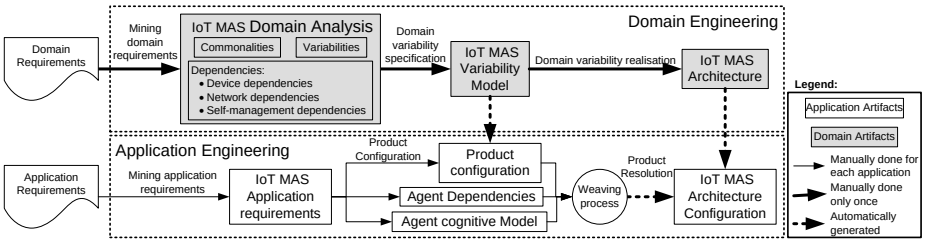

Fig. 1. SPL process for Self-StarMAS agents.

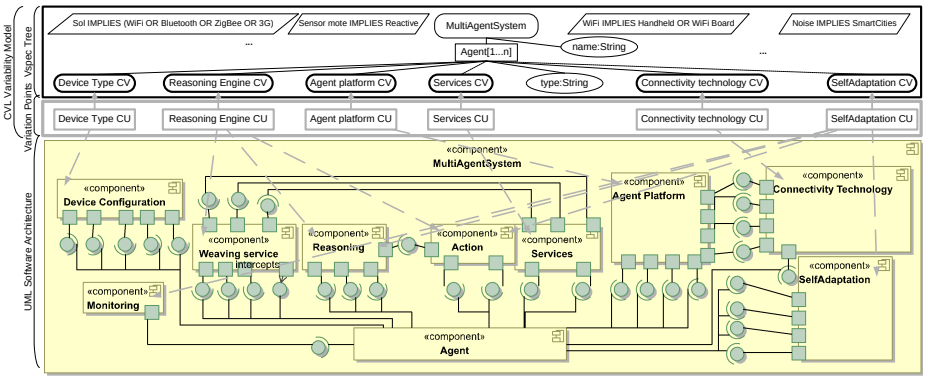

Fig. 2. Complete variability model of MAS for the IoT in CVL.

3 The Common Variability Language

CVL is a domain-independent language for specifying and resolving variability over any instance of any modeling language defined using a MOF-based metamodel [8]. The instance of the metamodel is referred to as the *base model* and CVL allows us to specify its *variability model*.

The *variability model* is a specification of the base model variabilities, and is divided into two parts. The first part, which is defined over the *base model*, marks its variation points. There are different types of variation points, in this contribution we use *existence*, to indicate the presence of an element in the base model, and *configurable units* that group a set of variation points.

The second part of the *variability model* is concerned with the definition of the relationships between the variation points. These are expressed by *variability specification*s (VSpec), which are organized using a hierarchical structure called a VSpec tree (see top of Fig. 2). The sub-tree under a VSpec means that the resolution of this VSpec imposes certain constraints on the resolutions of the VSpec in its sub-tree. These constraints will be explained in the following sections. Additionally, it is possible to specify *explicit constraints*, known as *crosstree constraints*. VSpecs are abstracts and they do not define which base model elements are involved nor how they are affected. The effect of the variability model on the base model is specified by *binding variation points*, which relates the base model and the VSpec tree.

Once the *base model* and the *variability model* have been defined, the *resolution model* is defined. This model results from selecting a set of VSpecs from the VSpec tree (i.e. the *variability model*). Then the CVL tool is executed to obtain the *resolved model*, a product model, which is a variation of the *base model* according to the choices that have been made in the *resolution model*.

4 Related Work

This section describes and discuses different approaches that are related to the presented work: development processes that use MAS-PL, self-adaptation for AAL systems and agent technology and AAL systems.

MAS-PLs have been applied to enhance the development of MAS in different ways. In [7] the Gaia methodology is modified to include SPL in the analysis and design phases of a MAS. The use of SPL allows to reduce by 48% the design documentation time at least in the case study presented, compared to the original Gaia. MaCMAS [18] is a methodology that uses formal methods and SPL to model autonomous and self-adaptation properties of MAS. It uses SPLs to model the evolution of the system taking into account the different products contained in the SPL. The work presented in [5] focuses on the AppE process by extending an existing product derivation tool for the MAS-PL domain. This proposal offers a complete SPL process with tool support to generate Jadex agents. These proposals are not suitable for the AAL domain since they do not consider the generation of agents for IoT devices and only [18] considers self-adaptation.

Self-adaptation for AAL systems has been proposed in different works. The eWALL project [14] proposes a prefabricated wall that has attached all of the ICT technology needed to enable a number of services for seniors. The work of this wall is based on self-adaptive sensors. The goal of the work proposed in [12] is to care elderly people by providing to caregivers data collected from their homes. The system uses self-adaptation to self-configure when new technology or services are provided and to adapt the context of use. The work presented in [22] uses a metaphor based on human emotions to model the self-adaptation of sensors in AAL applications. A system to permit elderly with balance disorders to live independently at home is presented in [4]. The designed control system is self-

adaptive, and it can be accommodated to conditions of users. These approaches demonstrate the benefits of endowing AAL systems with self-adaptation.

Agent technology and AAL systems are linked in several proposals. Due to limitations in space, here we mention just some of the most recent works. The paper [10] presents a context-aware MAS for care of the elderly that combines sensor technologies to detect falls and other health problems. In this case, agents are used to observe the elderly person from various points of view. An argumentative MAS is used in [15] to enable the reproduction and evaluation of inconsistent situations detected by AAL systems. They are part of an alarm management tool that supports carers to validate alarms raised by AAL systems. The goal of the system [20] is to help an elderly patient with his daily activities ensuring his security. The system uses Jade and Jason agents to implement a flexible architectural solution and reasoning about the patient condition. The work presented in [13] uses a MAS to control an AAL Flexible Interface. Agents adapt the interface based on the subject's requirements profile. The work [21] presents an environment of AAL created through the use of sensor networks and mobile agents. Agents are implemented using Jade and used for different purposes such as information retrieval. These are recent works that highlight the importance of the agent paradigm in the implementation of AAL systems and the relevance of a process that handles their development as we propose in here.

5 Domain Engineering of the Self-Adaptation concern

The DomE process (see Fig. 1) of our SPL process relies on the use of CVL and the Unified Modeling Language (UML) as MOF-based language. In [3] we defined the global variability model and architecture of the MAS for the IoT (see Fig. 2). Here we focus on the self-adaptation concern which is located in the VSpec *SelfAdaptation CV*, in the configurable unit *SelfAdaptation CU* and in the components *SelfAdaptation*, *Action*, *Reasoning* and *Monitoring*.

SelfAdaptation CV is a Composite VSpec (CVSpec), a type of VSpec that includes a VSpec tree inside. The root of the internal tree of this CVSpec (see Fig. 3) is *SelfAdaptation* that has three VSpecs that represents the elements *Monitor*, *Analyze* and *Plan* of the MAPE loop of the self-adaptive systems [11]. For this concern we have opted for an approach that focuses on the computational resources of the device where the agent is embedded. Thus, the child nodes under the *Monitor* VSpec consider physical resources of devices (e.g. *Battery*). The children of the VSpec *Plan* represent plans for fixing problems that can occur in the functioning of the agent (e.g. *Recover the location service*). Crosstree constraints of the *SelfAdaptation CV* represent dependencies external and internal dependencies of the self-adaptation concern (e.g. according to the crosstree *Noise*, the appearance of the VSpec *Noise lectures* in a resolution model depends on the appearance of the VSpec *Noise* in the same model) By selecting different plans and monitoring services, we effectively model different degrees of the self-adaptation for the agent.

Self-adaptation is a complex activity that involves different components of the system. Depending on the resolution model this is represented by the *Self-Adjusting* or the *SelfAdaptation* component. The main difference between these components is that *SelfAdjusting* is intended for an agent with a cognitive reasoning engine, and generates goals to fix situations classified in states that requires self-adaptation (e.g. *Activity decrease state*). While *SelfAdaptation* is intended for an agent with a reactive reasoning engine, and executes pre-defined plans.

6 Application engineering for AAL systems

The AppE process (see Fig. 1) is intended to generate the final architecture of the AAL system. Here we propose an AAL system that performs tasks to make the life of the elderly more secure and comfortable. Among other services, it monitors his/her physical condition to detect injuries caused by falls or other problems. To ensure that the system works properly without requiring user intervention, self-adaptation tasks have been taken into consideration. This system is designed as a self-adaptive MAS composed of agents embedded in the different devices that comprise the application (i.e. sensing devices and personal devices of the user). This section focuses on the configuration of the agent embedded in the user personal device (named *UserAgent*), which collects data from the other agents in the system to assist the elderly user.

The first step in the AppE is to select the VSpecs that satisfies the application requirements. To generate a valid product configuration the software architect maps the application requirements to the VSpecs of the VSpec tree. For example, to fulfil the self-adaptation requirement, the VSpec *SelfAdaptation CV* (see Fig. 3) has to be selected, and the same procedure is followed for the VSpecs tree inside *SelfAdaptation CV*. After that, a tool qould generate the *resolution model* that includes the constraints of the VSpec tree. For instance, in our case study, self-adaptation is concerned with elongating the life of the system. So, when *UserAgent* detects a low battery level it decreases its activity by performing different tasks to save battery. *Decrease Activity* requires *Battery* due to the crosstree constraint attached to *Decrease Activity* (see Fig. 3). The selection of both VSpecs also requires the selection of *SelfAdaptation*, *Monitor*, *Analyze* and *Plan* (because of parent-child relationships). Note that monitoring the battery level is not a requirement of the AAL system, but it is needed to obtain a valid configuration of the resolution model. So, this model not only contains VSpecs selected because of the application requirements, but it also includes VSpecs which are a result of crosstree constraints and parent-child dependencies.

Once the *resolution model* has been obtained, this is weaved with the *Agent cognitive Model* and the *Agent Dependencies* (see Fig. 1) using an ATL transformation [9]. The *Agent cognitive Model* includes goals, plans and knowledge specific to an application. The *Agent dependencies* model contains the dependencies between the *Agent cognitive Model* and the *IoT MAS Variability Model* (due to space limitations, a detailed description of these models is out of scope). Then the CVL tool uses the resultant model to generate the *resolved model*,

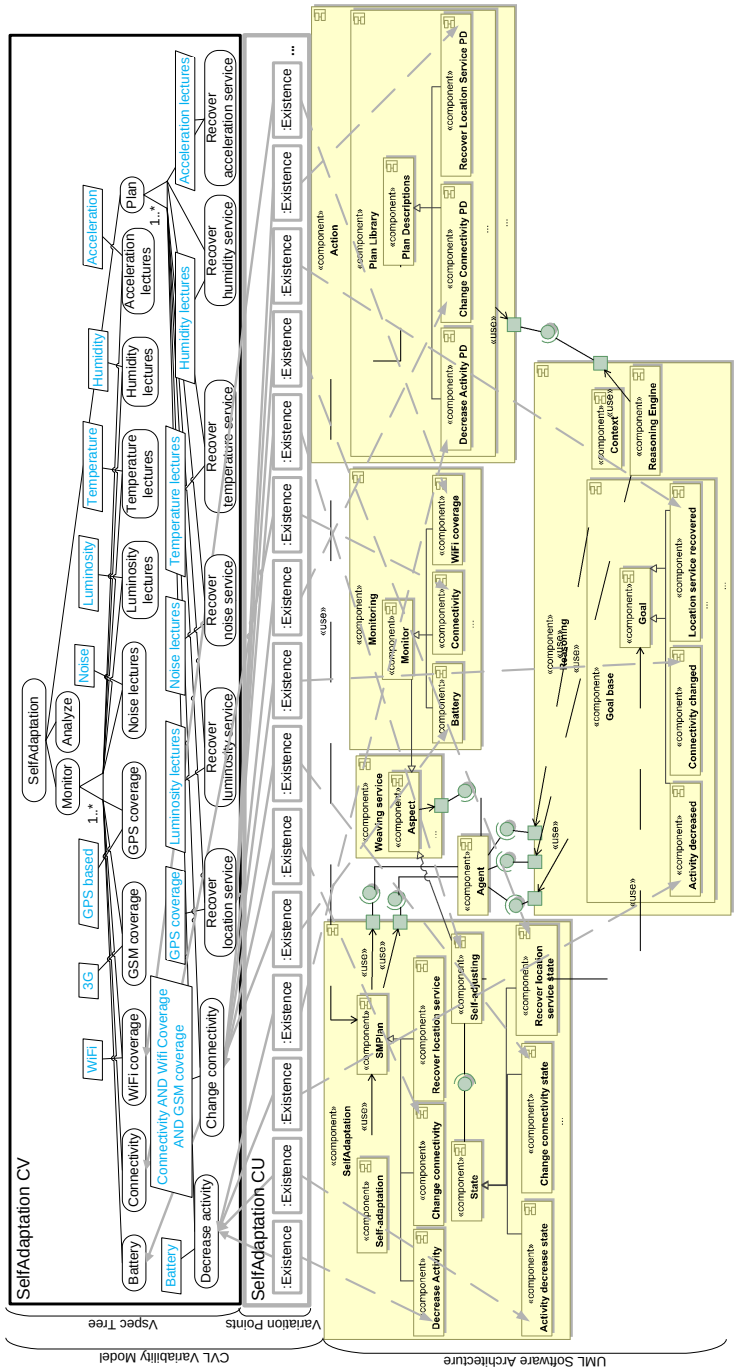

Fig. 3. Domain engineering of the Self-adaptation concern using CVL and UML.

which is a configuration of the system architecture with the set of components and connections that allow realizing the VSpecs present in the *resolution model*. This realization is derived from the Variation points, which are bound to elements in the VSpec tree and refer to elements of MAS for IoT architecture in *binding variation points*. For instance the variation point *:Existence* (see Fig. 3) bound to *Battery* indicates that if and only if this is selected in the resolution model, the *Battery* component will exist in the resolved model. The *resolved model* of *UserAgent* (see Fig. 4) includes the architectural components required to deal with two situations that require self-adaptation: to decrease the activity of the agent when the battery level is low and to recover the location service.

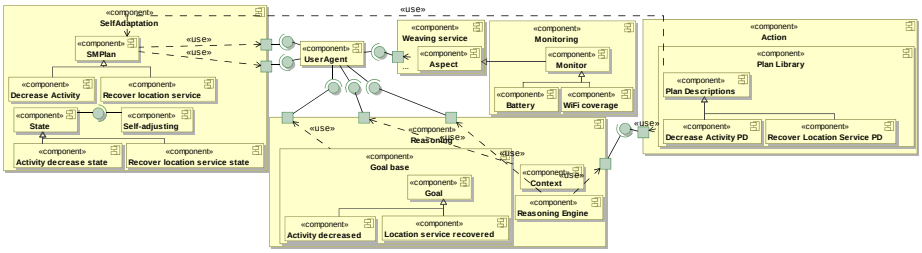

Fig. 4. UML diagram of the *UserAgent* architecture configuration.

7 Validation

In this section we analyze how our approach addresses the challenges raised in Section 2. To evaluate how well we address challenge C1, we calculate the degree of variability of the concerns of our agents. This metric is the division between the number of choices and valid resolutions and shows the expressive power of a VSpec tree or a subtree (the lower the degree, the higher the expressiveness of the tree). To apply this metric we have counted for each concern the number of components that could be injected to realize it (*Component* column in Table 1), the number of choices the software architect has to select in the AppE phase (*Choice* column) and the number of valid resolutions that can be generated (*Valid resolutions* column). Results show that as the number of choices increases, the number of architecture configurations also increases exponentially. For example, the number of self-adaptation resolutions is 35131, which means that the software architect could obtain any of these configurations only considering 22 choices. However, notice that the software architect does not need to be aware of this high variability, they only have to focus on selecting those choices that fit application requirements.

In order to evaluate the expressive power of our VSpec tree, our model have been compared with other SPL models with similar number of features. Eighteen models (see Table 2) have been compared in terms of the number of products that

Table 1. Degree of variability and dependency.

Concern	*Components*	*Choices*	*Valid resolutions*	*Degree of Variability*	*Intra-dependencies*	*Inter-dependencies*
Device Type	10	19	254	0.07	4	3
Reasoning	14	3	2	1.5	1	0
Agent platform	13	5	11	0.45	1	4
Services	10	14	2510	0.005	4	10
Connectivity	4	20	3423	0.005	11	8
Self-adaptation	43	22	35131	0.000626	10	8

can be generated and the degre of variability. These models have been extracted from the SPLOT (Software Product Line Online Tools) database[1], which has 717 models available to be analyzed. Specifically, we have selected models with a similar number of choices (between 70 and 100, see first comulm) and computed their number of valid configurations and degree of variability (provided in third and fourth column respectively). From the results we can conclude that our global SPL has a similar expressive power than other models with a similar number of choices, allowing to model a great variety of AAL systems from our VSpec tree.

To address challenge C2, we have included these dependencies in the VSpec tree using parent-children relationships and crosstree constraints. We evaluate the degree of dependency of our VSpec tree for the different concerns of our MAS counting these dependencies. We have distinguished between intra-dependencies (that occurs inside the same *CVSpec*, e.g. *Device Type*), and inter-dependencies (that occurs between different *CVSpecs*, e.g. between *Device Type* and *Reasoning Engine*). The modeling of dependencies eases the task of the software architect, since these dependencies are included automatically. The identification of dependencies can be an error-prone and complex task because it requires an expert in different domains. In the case of the self-adaptation 18 dependencies are identified. Using our approach we can guarantee that dependencies are automatically considered, and all required components will be included.

Achieving challenge C3 is addressed by our VSpec tree and the associated SPL architecture. The first one includes different VSpecs to model typical self-adaptation tasks, like recovering specific services. In addition, our architecture includes two types of self-adaptation specific to agents with goal-oriented or reactive engines, and a predefined set of tasks for self-adaptation.

8 Conclusions

In this paper we have analyzed the challenges that poses the development of self-adaptive AAL systems based on agents in the IoT. In our opinion challenges that should be addressed by agent development processes are (i) the management of

[1] SPLOT website: http://www.splot-research.org

Table 2. Comparison of SPL models in terms of number of valid configurations and degree of variability.

Model	*Number of Choices*	*Configurations*	*Variability Degree*
Database Tools	70	9,84E16	0,0083319
Video Player	71	4,5E+13	1,9061E-06
Reuso - UFRJ - Eclipse 1	72	226E+8	4,7758E-10
Car selection	72	3E+8	6,3626E-12
Toko	72	4,08E+11	8,6448E-09
Quality Attributes Functionalities	72	1,85E+11	3,9252E-09
Speech Recognition	75	652800	1,72779E-15
Photosharing	76	5,74E+12	7,5908E-09
J2EE web architecture	77	1,81E+10	1,2004E-11
Fish	80	2,25E+13	1,8582E-09
Webmail	81	4,49E+11	1,8581E-11
Self-StarMAS	**83**	**2.58E+7**	**1.6681E-7**
Frameworkprodemge	87	1,53E+11	9,8576E-14
Billing	88	3,87E+12	1,249E-12
Model Transformation	88	1,65E+13	5,3411E-12
Model Transformation	88	1,66E+13	5,3489E-12
Coche ecológico	94	2,32E+7	1,1725E-19
PGL_add	94	6,24E+8	3,1529E-18
UP Structural	97	1,48E+13	9,3455E-15

the variability of agents for AAL systems and (ii) the software and hardware dependencies, and (iii) the support for different degrees of self-adaptation. We have considered these issues to extend our SPL process for agents in the IoT. We have shown how to use our proposal to model and configure AAL systems and presented results that validate our proposal. As future work, we are working on the application of Dynamic SPL to enhance the self-adaptation of the agents and in the development of different tools to support the automatization of the resolution process.

Acknowledgments. This work is supported by the project Magic P12-TIC1814 and by the project HADAS TIN2015-64841-R.

References

1. Atzori, L., Iera, A., Morabito, G.: The internet of things: A survey. Computer Networks 54(15), 2787 – 2805 (2010)
2. Ayala, I., Amor, M., Fuentes, L.: The sol agent platform: Enabling group communication and interoperability of self-configuring agents in the internet of things. J. Ambient Intell. Smart Environ. 7(2), 243–269 (Mar 2015), `http://dl.acm.org/citation.cfm?id=2756713.2756723`
3. Ayala, I., Amor, M., Fuentes, L., Troya, J.M.: A software product line process to develop agents for the iot. Sensors 15(7), 15640 (2015)

4. Bakola, I., Bellos, C., Tripoliti, E., Bibas, A., Koutsouris, D., Fotiadis, D.: An adaptive home environment supporting people with balance disorders. In: XIII MEDICON, vol. 41, pp. 1213–1216. Springer (2014)
5. Cirilo, E., Nunes, I., Kulesza, U., Lucena, C.: Automating the product derivation process of multi-agent systems product lines. Journal of Systems and Software 85(2), 258–276 (2012)
6. Clements, P., Northrop, L.: Software product lines: practices and patterns, vol. 59. Addison-Wesley Reading (2002)
7. Dehlinger, J., Lutz, R.R.: Gaia-pl: A product line engineering approach for efficiently designing multiagent systems. ACM TOSEM 20(4), 17:1–17:27 (2011)
8. Haugen, O.: Common variability language. Tech. Rep. ad/2012-08-05, Object Management Group (Aug 2012)
9. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: Atl: A qvt-like transformation language. In: 21st OOPSLA. pp. 719–720 (2006)
10. Kaluža, B., Luštrek, M., Dovgan, E., Gams, M.: Context-aware mas to support elderly people. In: Proc. of the 11th AAMAS. pp. 1485–1486. IFAAMAS (2012)
11. Krupitzer, C., Roth, F.M., VanSyckel, S., Schiele, G., Becker, C.: A survey on engineering approaches for self-adaptive systems. Pervasive Mob. Comput. 17, Part B, 184 – 206 (2015)
12. Kucher, K., Weyns, D.: A self-adaptive software system to support elderly care. In: Modern Information Technology (2013)
13. McNaull, J., Augusto, J., Mulvenna, M., McCullagh, P.: Flexible context aware interface for ambient assisted living. Human-centric Computing and Information Sciences 4(1), 1 (2014)
14. Mihovska, A., Kyriazakos, S., Prasad, R.: ewall for active long living: Assistive ict services for chronically ill and elderly citizens. In: IEEE SMC. pp. 2204–2209 (2014)
15. Muñoz, A., Serrano, E., Villa, A., Valds, M., Bota, J.A.: An approach for representing sensor data to validate alerts in ambient assisted living. Sensors 12(5), 6282 (2012)
16. Nehmer, J., Becker, M., Karshmer, A., Lamm, R.: Living assistance systems: An ambient intelligence approach. In: Proc. of the 28th ICSE. pp. 43–50. ACM (2006)
17. Nunes, I., de Lucena, C.J., Kulesza, U., Nunes, C.: On the development of multi-agent systems product lines: A domain engineering process. In: AOSE X, vol. 6038, pp. 125–139. Springer (2011)
18. Peña, J., Rouff, C.A., Hinchey, M., Ruiz-Cortés, A.: Modeling nasa swarm-based systems: using agent-oriented software engineering and formal methods. SoSyM 10(1), 55–62 (2011)
19. Renaud, K., van Biljon, J.: Predicting technology acceptance and adoption by the elderly: A qualitative study. In: Proc. of SAICSIT. pp. 210–219. ACM (2008)
20. Sernani, P., Claudi, A., Palazzo, L., Dolcini, G., Dragoni, A.F.: Home care expert systems for ambient assisted living: A multi-agent approach. In: Proc. of AgeingAI. CEUR-WS (2013)
21. Su, C.J., Chiang, C.Y.: Pervasive community care platform: Ambient intelligence leveraging sensor networks and mobile agents. Intern. J. Syst. Sci. 45(4), 778–797 (2014)
22. Thomas, A., Moore, P., Evans, C., Sharma, M., Chima, P., Vijay, V., Rmeileh, S.: Emotive sensors for intelligent care systems: A heuristic discussion of autonomic wireless sensing systems. In: 7th CISIS. pp. 499–504 (2013)