SmartyCo: Managing Cyber-Physical Systems for Smart Environments

Daniel Romero, Clément Quinton, Laurence Duchien, Lionel Seinturier, Carolina Valdez

SmartyCo: Managing Cyber-Physical Systems for Smart Environments

Daniel Romero[1], Clément Quinton[2], Laurence Duchien[1], Lionel Seinturier[1], and Carolina Valdez[3]

[1] Université Lille 1 & Inria, CRIStAL (UMR CNRS 9189) laboratory, France
{first.last}@inria.fr

[2] Politecnico di Milano, DEIB, Piazza L. Da Vinci, 32 - 20133 Milano, Italy
clement.quinton@polimi.it

[3] Media.lab, Instituto Pladema, UNCPBA, Tandil, Argentina
Cvaldezgandara@alumnos.exa.unicen.edu.ar

Abstract. *Cyber-Physical Systems* (CPS) are composed of heterogeneous devices, communicating with each other and interacting with the physical world. Fostered by the growing use of smart devices that are permanently connected to the Internet, these CPS can be found in *smart environments* such as smart buildings, pavilions or homes. CPS must cope with the complexity and heterogeneity of their connected devices while supporting end-users with limited technical background to configure and manage their system. To deal with these issues, in this paper we introduce SmartyCo, our approach based on *Dynamic Software Product Line* (DSPL) principles to configure and manage CPS for smart environments. We describe its underlying architecture and illustrate in the context of smart homes how end-users can use it to define their own CPS in an automated way. We then explain how such an approach supports the reconfiguration of smart devices based on end-users rules, thus adapting the CPS *w.r.t.* environment changes. Finally, we show that our approach is well-suited to handle the addition and removal of CPS devices while the system is running, and we report on our experience in enabling home inhabitants to dynamically reconfigure their CPS.

1 Introduction

Nowadays, the definition of *Cyber-Physical Systems* (CPS), *i.e.*, systems controlling physical elements in the real world from software applications, is a reality. This is due to the emerging Internet of Everything paradigm, where smartphones, tablets, PCs and different devices are connected to the Internet. These CPS enable a 'smart everywhere society', where roads, cars, trains, offices, public spaces or homes interact smartly. CPS must thus be context-aware, *i.e.*, able to react to environment changes by considering the technological heterogeneity, concurrency and availability of connected devices.

For instance, homes have recently been considered as smart environments and several works [5, 7, 13] focus on the Do-It-Yourself Smart Home paradigm. However, these approaches do not provide enough flexibility in terms of services that can be used, produce technological lock-in, or introduce additional costs. Thus, allowing end-users to configure, control and update their own CPS is a challenging task, as it requires a solution managing the variability of such CPS at design and runtime, while enabling these users to easily cope with this variability.

In order to face this challenge, we present in this article SmartyCo, our *Dynamic Software Product Line* (DSPL) [6] based approach to capture smart environment configurations via extended *feature models* [1] and define smart CPS. In particular, SmartyCo provides three main contributions: *(1)* A software product line for deriving the system configuration according to user requirements and available devices, *(2)* a support for runtime adaptation via end-user rules and *(3)* a bidirectional mechanism for keeping the software product line and the cyber-physical system synchronized in presence of adaptations. The context-awareness and the reactivity of the system are reified by defining event-condition-action rules, which execute tasks required by users under given conditions while maintaining the consistency of the whole system. The rules together with the possibility to incorporate new services into the system represent the dynamic part of the product line.

The next section presents the technologies that makes possible the definition of our approach. Then, Section 3 provides an overview of SmartyCo and explains the rule definition mechanism. Sections 4 describes a prototype, while related work is discussed in Section 5. Section 6 concludes the paper.

2 Background Information and Enabling Technologies

As already mentioned, smart environments are equipped with several devices:

Set-top boxes, that provide interactive services such as search and configuration services. In recent years, these boxes have gained computation power by increasing their storage and processing capabilities, and have become more user-friendly by relying on well-designed interfaces. For instance, the Minix Neo X-7, an Android-based box, can be used to install, configure and control new software to manage cyber-physical systems.

Smartphones, that are permanently connected to the Internet through the 3G, 4G and Wi-Fi spots. Thanks to dedicated applications, smartphones can also be used to interact with physical equipments, *e.g.*, switching channels on television or listening to music on external loudspeakers.

Appliances and ***devices***, which can be controlled via PC, smartphones and tablets. For example, light switches, thermostats, and cameras from the Belkin WeMo and Z-Wave families can be controlled via Internet services such as *If-This-Then-That* (IFTTT) [8] which is based on event-condition-action rules.

This variability in terms of technologies requires an approach enabling their coordination to run as a unified system. Furthermore, an automation solution has to consider different environment designs, has to be flexible, extensible, intuitive, trustworthy and has to make end-users feel that they are gaining control [14]. In the next section we introduce our approach to deal with these issues.

3 Managing Cyber-Physical Systems With SmartyCo

3.1 Overview

SmartyCo relies on *Dynamic Software Product Line* (DSPL) principles [2]. A software product line reduces the efforts for producing a family of software products, *e.g.*, CPS. Within this family, products share common functionalities

(*i.e.*, commonalities) and differ according to variable ones (*i.e.*, variability) [10]. A domain expert identifies this variability, which is reflected into a variability model, *e.g.*, a *feature model* [9]. Features from this variability model are mapped to concrete reusable software artifacts, or *assets*, which are bound together to yield the software product. In SmartyCo DSPL, each CPS is thus a product. A DSPL supports binding variability at runtime, thus enabling a system to reconfigure itself while running. Such principles allow SmartyCo to deal with the static configuration of CPS for smart environments while supporting the adaptation of such systems when required. The SmartyCo DSPL generates controllers that are deployed on Android-based set-top boxes. Such controllers allow the definition of rules to manage the CPS as well as rules to deal with the context-awareness of the devices in use. Those controllers share a generic common configuration, but differ in some variation points according to the rules defined by the end-user and the available devices. SmartyCo copes with this technological variability by describing CPS using feature models. Figure 1 provides an overview of SmartyCo.

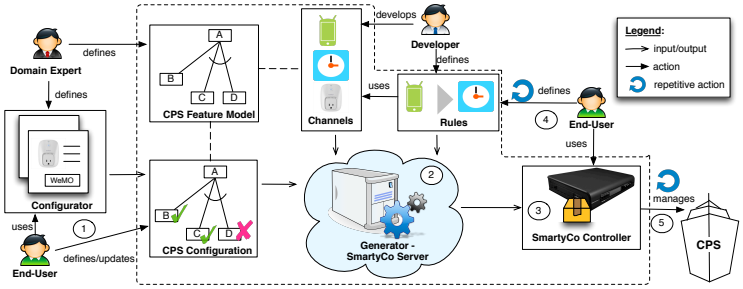

Fig. 1. Overview of the SmartyCo DSPL

End-users rely on the `Configurator` for defining a CPS configuration by selecting the required devices and their location ①. The `Generator`, deployed on the SmartyCo `Server` running in a private Cloud to ensure data privacy, processes such a configuration to check its validity. To do that, it translates extended feature models into a constraint satisfaction problem and solves it relying on the Choco solver [3], as described in [11]. Then, the `Generator` combines assets related to selected devices to produce the CPS `Controller` ②, which is deployed on the set-top box ③. Assets in SmartyCo are channels (*i.e.*, sensors and actuators) and rules. End-users define rules when needed via the `Controller` ④, which orchestrates the CPS ⑤. Channel developers also define rules to keep the system consistent. With such an approach, the SmartyCo DSPL provides a customized CPS based on available devices.

3.2 Extended Feature Models to Manage CPS Variability

In SmartyCo, we rely on model driven engineering principles [12] to define the feature models of smart environments. Figure 2 depicts our metamodels for such environments. These metamodels rely on and extend the *metamodel* we proposed in [11] to define feature models with cardinalities and attributes.

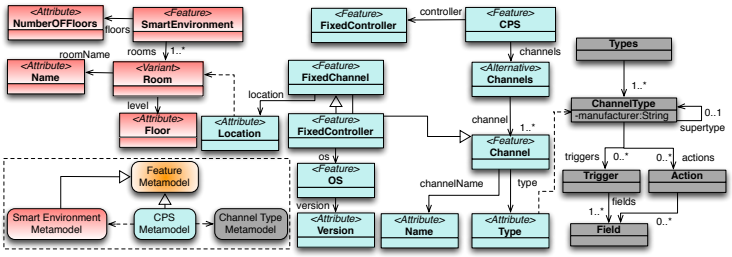

Fig. 2. Metamodels to manage variability in smart environments

The `Channel Type` metamodel (*cf.* gray boxes) enables the definition of actuators and sensors providing actions and triggers (or events). Inspired by IFTTT [8], our approach allows smart objects to work for end-users through simple rules. Each rule has a trigger, an action and a priority to deal with conflicts. Rules with higher priority are executed and if two or more rules have the same priority, they will be executed in the order of their definition. Examples of channel types are smartphones and Belkin WeMo thermostats, while *if someone turns on the thermostat, send me a SMS* is a rule. The `CPS` metamodel (*cf.* blue boxes) is used to specify concrete channels. They can be fixed with a location (*e.g.*, a crock-pot is in the kitchen) or mobile (*e.g.*, smartphones). We also introduce `Fixed Controllers` that are installed on set-top boxes to control channels and rules. Finally, the `Smart Environment` metamodel (*cf.* degraded salmon boxes) enables the definition of the environment that hosts the CPS. The `CPS` metamodel uses the `Smart Environment` metamodel to specify the location of `Fixed Controllers`. Each smart environment can be configured independently by instantiating these metamodels. Below, we explain and illustrate their use.

Why extended feature models? The `Configurator` (*cf.* Figure 1) enables the user to deal with the variability of CPS and smart environments. This physical and software variability is reflected in the feature models, which are extended with cardinalities and attributes [1]. Feature models enhanced with such extensions allow the definition of *(1)* multiple instances for the same feature, and *(2)* complex constraints over these features. For example, several instances of the same feature (*e.g.*, a channel or a room) can be defined and configured, while attributes provide additional information about features such as location or type. Finally, these extended feature models support the definition of complex relationships involving cardinality and attribute values such as *the configuration of one or more Z-Wave Switches requires the configuration of at least one Z-Stick Controller* (to enable the definition of a Z-Wave device network).

Figure 3 depicts the use of these extensions with instances of the `CPS` and `Smart Environment` metamodels. The related Channel Type model (not shown here) contains types related to WeMo and Z-Wave devices. The lower part of the figure depicts the feature model of `MyHome`, an instance of a smart environment. This smart home is made up of a bedroom, a living room, a bathroom and a kitchen. The upper part depicts a CPS feature model (`HomeCPS`) that is composed by several devices. In particular, it describes two different cases involving the use of features whose cardinality upper bound is greater than one. On one hand,

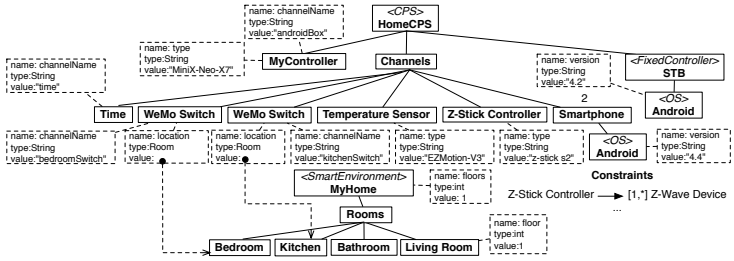

Fig. 3. CPS configuration in a smart home (Excerpt)

two different instances of the `WeMo Switch` feature are configured. Indeed, several devices of the same type may be present in the same home. On the other hand, two identical instances of `Smartphone` are also configured, meaning that two home inhabitants have the same smartphone model.

For the CPS to be consistent, some constraints must be defined. In this example, the configuration of an Aeotec `Z-Stick Controller` requires the presence of at least one Z-Wave device. This kind of constraints is properly handled by extended feature models and the related reasoning tools, *e.g.*, the Choco solver.

3.3 Dynamic Adaptation for CPS

In this section, we explain how rule definition and channel addition are used to modify running configurations of SmartyCo cyber-physical systems.

Rules. In SmartyCo, event-condition-action rules keep DSPL products consistent at runtime. In particular, SmartyCo distinguishes between CPS rules and User rules. Both rules use channels as providers of events and actions. The difference between these rules is related to permissions and priorities. End-users cannot modify CPS rules, while User rules are defined and modified by end-users. Regarding priorities, CPS ones are specified by the channel developers while the User rules are defined by the end-users. We provide below examples of CPS rules (CPS-R) and User Rules (User-R) for the scenario depicted in Figure 3.

```
CPS-R if(bedroomSwitch.state != available) then androidSetTopBox.disableRules(bedroomSwitch); priority = 10;
User-R1 if (bedroomTemperatureSensor.value >= 20) then bedroomSwitch.turnOff; priority = 10;
User-R2 if (time.value == 22:00) then bedroomSwitch.turnOn; priority = 5;
```

For instance, the rule User-R1 indicates that if the temperature in the bedroom is greater or equal to 20 degrees, then the bedroom switch is turned off. Some defined rules can conflict when dealing with the same actuator. For example, User-R1 and User-R2 can both be triggered at the same time. In such a case, the state of the switch would be `on` because of User-R2, the last executed rule. Such a state could be the one expected by the end-user or not. To avoid such an uncertainty, the end-user defines priorities on the rules as natural numbers. For instance, User-R1 has 10 as priority and User-R2 has 5. Thus, whenever both rules are triggered, the priority is given to User-R1. To process such priorities, we use a simplified version of the rule composition algorithm presented by Zave et al. [14]. We also consider the rule type to evaluate priorities. CPS rules have a higher priority than User rules, independently of the `priority`.

In order to avoid dysfunctions related to unreachable channels, when an end-user configures a cyber-physical system, three CPS rules are set for each channel: *(1)* a rule that disables the User rules related to the channel, so that they cannot be modified or executed. CPS-R is an example of this rule for `bedroomSwitch`; *(2)* a rule that notifies the end-users through their smartphone about the channel unavailability; *(3)* a rule that enables again the channel rules once the channel is available, *i.e.*, when an end-user solves the issue related to the channel.

As previously described, these default CPS rules avoid incorrect behaviors of the system at runtime. Additionally, each User rule checks the availability of channels before condition or trigger validation. To do this, SmartyCo relies on a `state` trigger related to each channel. As most of rule triggers, the channel state is polled every 15 minutes by default, value that can be changed by the user.

Channel Addition. The SmartyCo DSPL also enables end-users to change from one `Controller` configuration to another one through the removal and addition of channels at runtime. That is why SmartyCo requires that end-users define the evolution of the cyber-physical system with these modifications. To integrate new channels into the system, end-users select from a list of channels the one related to the new device. If the channel functionality is already installed, the SmartyCo `Controller` detects the new device and asks the end-user to configure the new channel by selecting the correct appliance and location. However, when a channel needs to be installed, an update is required through the SmartyCo `Server`, which computes and executes the required changes on the current `Controller` configuration. After this, the end-user configures the new channel. Whether an update is required or not, a new channel cannot be available for rule definition if SmartyCo cannot check its state. To remove a channel from the system, such a channel must not be required anymore. In such a case, all rules related to the removed channel are deleted.

4 Experience

We developed a prototype of SmartyCo. The `Server`, implemented in Java, exposes the `Configurator` as a REST service deployed in a private Cloud based on OpenStack. Once generated by the DSPL, the Android-based Controller is installed manually in the set-top box. For our experiments, we used a smartphone based on the Android 4.2 version and a MiniX Neo X7 as set-top box. We also used an Aeotec Z-Stick Controller connected to this set-top box to interact with Z-Wave devices. This configuration was then deployed in a smart environment made of 4 rooms. In the current implementation, 5 Z-Wave devices and their channels are available, as well as a channel for Android phones. In the SmartyCo DSPL, the extended feature model reflecting this configuration is translated into a constraint satisfaction problem with 14 constraints, leading to 20 different configurations. Even with a limited number of devices (*i.e.*, 6), end-users already have to deal with an important number of system configurations.

For the first experiment, we evaluate the time required to access devices when using channels. For this evaluation, we selected an On/Off switch and we

performed 50 runs to turn it on and off using the dedicated channel deployed in the set-top box. We computed the average time for the two operations, which was measured at 824 ms for *turnOn* and 888 ms for *turnOff*. These low values show that our CPS rules can be executed in a reasonable time and that the state verification has a negligible impact on the execution time of user rules. The second experiment checks the correctness of the system's behavior when some devices become unreachable. A 3-in-1 motion-temperature-luminance sensor together with a switch are part of the system, and each one is configured with a User rule and the default CPS rules. These rules are desynchronized, which means that the system rules are executed first, and the end-user rule 2 minutes later (this last value was defined arbitrarily for testing purposes). Both devices are then turned off. The system rules successfully detect the problem and disable the user rule. Once the devices are turned back on, the `Controller` detects the presence of the devices and the system continues running properly.

5 Related Work

SPOK [4] is an End-User Development Environment for smart homes. The environment, based on OSGi, provides a middleware layer enabling interactions with different kind of devices and a pseudo natural language to define event-condition-action rules. The language proposed by SPOK could be integrated into SmartyCo to provide a textual definition of event-condition-action rules including the loops usage. The Controller in SmartyCo can play the middleware role by enabling access and control of the physical world via the channels.

Humle et al. [7] present a user-oriented approach based on JavaBean components that encapsulate sensors and actuators. Authors also propose a jigsaw-based editor, where each JavaBean component is represented as a jigsaw puzzle piece that can be composed with other pieces. On the other hand, CAMP [13] is a system enabling the definition of home applications without focusing on target devices. CAMP allows users to specify the application tasks and goals by means of a subset of natural language. We can exploit ideas related to GUIs proposed by these approaches in order to improve the SmartyCo user interface.

Dey et al. [5] introduce iCAP, a system enabling end-users to build context-aware applications without writing any code. The users define if-then rules and spatial, temporal and personal relationship-based rules, together with the devices to be used. iCAP proposes an approach similar to SmartyCo, where end-users can build and configure their system via rules. However, it is not clear how the system can be extended to include new devices, how they are configured or which actual devices available in the environment can be controlled. The dynamic part of SmartyCo enables the inclusion of new devices when they become available.

6 Conclusions

In this paper we presented SmartyCo, our solution for dealing with the variability of smart environment systems managed by end-users. SmartyCo is based on DSPL principles to manage the variability of such systems at design and runtime. In particular, the runtime adaptations are twofold. First, they are defined via end-user rules requiring the activation or deactivation of certain devices

under certain conditions, and validated by checking the related feature model configuration. Second, it is possible to easily add new devices when they become available. In such a case, the consistency of the DSPL is ensured by predefined rules avoiding the usage of unavailable devices.

Our experience in building a prototype confirms that we are close to enable an easy and partially automated configuration of *dynamic* CPS complying with any smart environments. Issues related to privacy and security as well as vulnerabilities related to devices must also be considered when modeling and generating the CPS. We also need to further consider conflicts between rules and their priorities by putting the accent in the behavior of the system regarding possible end-user behavior overriding action rules.

References

1. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated Analysis of Feature Models 20 Years Later: A Literature Review. Inf. Syst. 35(6), 615–636 (Sep 2010)
2. Capilla, R., Bosch, J., Trinidad, P., Ruiz-Corts, A., Hinchey, M.: An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry. Journal of Systems and Software 91(0), 3 – 23 (2014)
3. Choco Team: Choco: an Open Source Java Constraint Programming Library. Research report 10-02-INFO, École des Mines de Nantes (2010)
4. Coutaz, J., Demeure, A., Caffiau, S., Crowley, J.L.: Early Lessons from the Development of SPOK, an End-user Development Environment for Smart Homes. In: Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing. pp. 895–902. UbiComp'14 (2014)
5. Dey, AnindK. and Sohn, Timothy and Streng, Sara and Kodama, Justin: iCAP: Interactive Prototyping of Context-Aware Applications. In: Pervasive Computing, vol. 3968, pp. 254–271 (2006)
6. Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K.: Dynamic Software Product Lines. Computer 41(4), 93–95 (April 2008)
7. Humble, J., Crabtree, A., Hemmings, T., kesson, K.P., Koleva, B., Rodden, T., Hansson, P.: "Playing with the Bits" User-Configuration of Ubiquitous Domestic Environments. In: Ubicomp. vol. 2864, pp. 256–263 (2003)
8. IFTTT: Put the internet to work for you (2015), `https://ifttt.com/`, [Online; accessed 16-April-2015]
9. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (foda) feasibility study. Tech. rep., Carnegie-Mellon University Software Engineering Institute (November 1990)
10. Pohl, K., Böckle, G., Linden, F.J.v.d.: Software Product Line Engineering: Foundations, Principles and Techniques (2005)
11. Quinton, C., Romero, D., Duchien, L.: SALOON: a Platform for Selecting and Configuring Cloud Environments. Software – Practice and Experience p. 10.1002/spe.2311 (Jan 2015)
12. Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. Computer 39(2), 25–31 (Feb 2006)
13. Truong, K.N., Huang, E.M., Abowd, G.D.: CAMP: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home. In: Proceedings of the 6th International Conference on Ubiquitous Computing. pp. 143–160 (2004)
14. Zave, P., Cheung, E., Yarosh, S.: Toward User-Centric Feature Composition for the Internet of Things. Tech. rep., AT&T Laboratories Research (2014)