Li XS, Tao XP, Song W *et al.* AocML: A domain-specific language for model-driven development of activity-oriented context-aware applications. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 33(5): 900–917 Sept. 2018. DOI 10.1007/s11390-018-1865-9

AocML: A Domain-Specific Language for Model-Driven Development of Activity-Oriented Context-Aware Applications

Xuan-Song Li[1,2], *Member, CCF*, Xian-Ping Tao[2], *Senior Member, CCF, Member, IEEE*
Wei Song[1,2], *Senior Member, CCF, Member, IEEE*, and Kai Dong[3]

[1]*School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China*
[2]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China*
[3]*School of Computer Science and Engineering, Southeast University, Nanjing 211189, China*

E-mail: lixs@njust.edu.cn; txp@nju.edu.cn; wsong@njust.edu.cn; dk@seu.edu.cn

Received March 5, 2018; revised July 18, 2018.

Abstract Activity-oriented context-aware (AOCA) applications are representative in pervasive computing. These applications recognize daily-life human activities, perceive the environment status related to the activities, and react to ensure the smooth performance of the activities. Existing research proposed a specific light-weight, incremental method to support the development of such applications; however it is not easy to learn and use. This paper aims to facilitate the development of such applications and improve the productivity of developers. We propose AocML, a textual domain-specific language which provides a high-level abstraction of AOCA applications. Specifically, we first show the software model of AOCA applications and the abstract syntax of AocML. Then, we introduce the concrete syntax of AocML. We also implement the tools for AocML, including the development environment as well as the generation of Java code and ontology specification. Moreover, we use a case study and evaluation to demonstrate the advantages of AocML.

Keywords pervasive computing, context-awareness, model-driven development, domain-specific language

1 Introduction

Pervasive computing[1] is a computing paradigm which embeds computing resources into the environment, and provides services for users ubiquitously and transparently. One of the key properties of pervasive computing applications is context-awareness[2], that is, such applications can sense the environment and react based on the environment.

Among these applications, we focus on a typical kind of applications with the above computing paradigm, namely, activity-oriented context-aware (AOCA) applications[3]. Users of AOCA applications live in a smart space and perform autonomous activities such as sleeping, reading a book, and watching TV. For different activities, users may have different requests for the environment. For example, when a user is reading in a living room, the living room is required to be bright; when he/she is sleeping in a bedroom, the bedroom needs to be relatively dark and has an appropriate temperature. The computing system recognizes users' activities and senses the environment status. According to this information, it provides services in order to ensure the smooth performance of the activities. AOCA applications are common in the research of pervasive computing. Typical AOCA applications include smart home[4], smart meeting room[5], elderly care systems[6], etc.

These applications have some features such as openness of the environment and personalization of applica-

Regular Paper

Special Section on Software Systems 2018

The work was supported by the National Key Research and Development Program of China under Grant No. 2017YFB1001801, the National Natural Science Foundation of China under Grant Nos. 61702263, 61761136003, and 61373011, the Natural Science Foundation of Jiangsu Province of China under Grant No. BK20171427, and the Fundamental Research Funds for the Central Universities of China under Grant No. 30917011322.

tion requests. Therefore, it is difficult for developers to deploy the environment resources and analyze the requests completely once and for all. The general pervasive computing applications development methods (e.g., [7-8]) do not consider the activity-oriented incremental development; thus they cannot support the development and maintenance of AOCA applications in a flexible way.

To deal with this challenge, the existing work[3] proposed a lightweight, incremental programming framework. This framework separates the concerns of environment resource descriptions and the application requests definitions. Furthermore, the application requests are dependent on user activities and can be further separated into constraints related to each activity. Although the programming framework and an API (application program interface) of AOCA applications have been proposed, the development of such applications still lacks enough guidance. The developers also need to spend a considerable amount of time learning how to use API.

Domain-specific language (DSL) is an alternative way against API. It provides a development tool which is tailored towards a particular application domain in a higher level. In some specific domains, DSLs are able to improve the productivity of development[9-11].

In this paper, we present a textual DSL named AocML (short for Activity-Oriented Context Model Language) for the model-driven development (MDD) of AOCA applications in order to promote the development support for such applications. The main contributions are as follows.

1) We propose a high level model for AOCA applications to describe the entities in such applications.

2) We propose AocML for specifying AOCA applications to facilitate the development of such applications.

3) We implement a platform supporting the development of AOCA applications based on AocML. This platform includes an AocML development environment and a code generator which generates software artifacts from AocML codes.

The rest of this paper is organized as follows. In Section 2, we give a review of AOCA applications and the development method. We also discuss the design choices of a DSL for AOCA applications. Then, we analyze the metamodel of AOCA applications in Section 3. We present AocML concrete syntax in Section 4 and the implementation of the tools for AocML in Section 5. We give a case study in Section 6 and an evaluation in Section 7. We review the related work in Section 8, and finally conclude the paper in Section 9.

2 Background and Rationale

In this section, we first introduce the concept and development method of AOCA applications. We then discuss the requirements of the DSL for AOCA applications.

2.1 Activity-Oriented Context-Aware (AOCA) Applications

Among various kinds of context-aware applications, we use the term "AOCA applications" to refer to the applications which recognize user activities, sense environment information related to the activities, and influence the users and environment in order to provide environment-related support for the activities.

AOCA applications have the following characteristics.

Activity Relativity. AOCA applications provide environment-related support for autonomous activities of users[①]. In other words, the requirements of these applications are represented as the environment constraints related to user activities. This is the essential feature of these applications.

Shareable Environment Infrastructure. Various users may perform various activities in an environment of smart space. This environment can be encapsulated as a proactive infrastructure which is independent of the activities. This infrastructure is shared by different activities or AOCA applications. Such infrastructure is open and dynamic because the resources are often added into it or removed from it.

Spatiotemporal Locality. From the temporal perspective, each activity has a duration period. AOCA applications only need to consider environment constraints in each duration period instead of the globe requirements. From the spatial perspective, the environment constraints in a time period usually only relate to partial resources. The applications may organize the partial resources to provide services for user activities.

Generally, context-aware systems consist of at least three layers.

1) *Environment Data Source Layer.* This layer encapsulates the devices and services which are able to sense or influence the environment. It describes the system's capabilities of sensing and influencing the environment.

[①] An activity is seen as a system of human "doing" whereby a subject works on an object in order to obtain a desired outcome[12].

2) *Context Management Layer.* This layer attaches sematic meaning to environment information. Further, it performs managing processes such as fusion and filter.

3) *Business Logic Layer.* This layer describes the business logics of applications and the requests for environment.

Some efforts (such as [13]) layer these applications in other ways. These ways can be viewed as a refinement of the previous three layers by dividing one or more layers.

On the basis of the three layers, different kinds of applications describe the attributes and relationships in each layer according to the characteristics of the applications. For AOCA applications, these layers are shown in a structure as Fig.1.

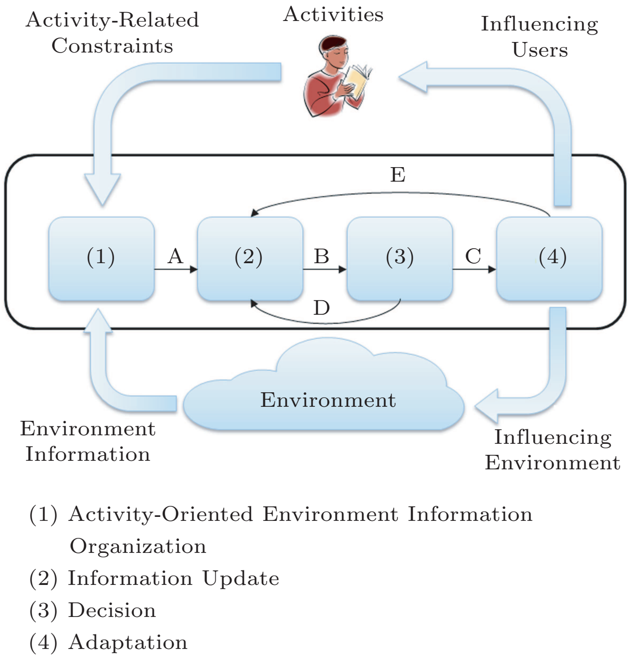


Fig.1. Layers of AOCA applications.

The environment data source layer of AOCA applications defines an environment infrastructure which can be shared by various activities. The attributes in the environment can be sensed and influenced by the applications. The business logic layer of AOCA applications consists of various activities of various users and the requests of these activities. The requests represent as constraints on the environment. AOCA applications maintain a suitable environment for activities to ensure the smooth performance of the activities. The context management layer of AOCA applications contains four iterative phases.

Activity-Oriented Environment Information Organization. The requirements of AOCA applications are related to activities. In the runtime, a part of information in the environment related to a phase of an activity is organized. After the organization, the environment information waits for data update (arrow A). This part of information performs as the base of context-aware decision and adaptation. When the activity changes, the information will be reorganized.

Information Update. The system obtains the updated data value of the attributes in the environment. After the data update, it turns to the decision phase (arrow B).

Decision. The system judges whether the environment satisfies the constraints of current activities. If the environment does not satisfy the constraints, it turns to the adaptation phase (arrow C); otherwise, it turns to the information update phase to wait for future data update (arrow D).

Adaptation. When the environment does not satisfy the constraints, the system influences the environment or influences the user activities. Then, the system turns to the information update phase to wait for future data update (arrow E).

2.2 Development Method of AOCA Applications

Because of the characteristics of AOCA applications, there are some difficulties to develop such applications. First, the resources in the shared environment are often added or moved. Therefore, not only the values in the environment but also the capabilities of sensing and influencing the environment are open and dynamic. Second, activity-related requests for the environment are personalized. Such requests are different from person to person. When a new user enters the system, he/she may add personalized requests. These difficulties make developers difficult to deploy the entire environment resources and design the requests during the development phase once and for all.

The existing work[3] proposed a lightweight, incremental programming framework for AOCA applications in order to facilitate the development and maintenance. In this method, there are two kinds of developers focusing on different parts of the system.

Infrastructure Developers. These developers deal with the environment. They construct and maintain an environment infrastructure. The infrastructure firstly specifies the types of environment attributes which can be sensed or influenced (named as "features", e.g., light intensity, temperature). Then, it specifies the instances of environment attributes related to specific devices (e.g., sensors, lamps).

Application Developers. These developers deal with the requests for the environment. They specify the constraints of different activity types. Furthermore, the personalized constraints should be concerned.

This method reflects the idea of "separation of concerns". The infrastructure developers do not need to consider the requests of activities. The application developers just need to read a description provided by the infrastructure developers in order to know which features can be considered by the system. They do not need to know the details of the infrastructure. The attributes in the environment and the constraints in the applications are lightweight and pluggable.

This work[3] also provided an API to develop AOCA applications with this programming framework and a supporting platform PAOC.

2.3 DSL Requirements

The existing work proposed the development method and an API to facilitate the development of AOCA applications. The developers still take effort to learn how to use this API. Therefore, it is unfavourable for the involvement of the domain experts. Furthermore, development artifacts are not concise enough. The conciseness will affect the productivity of such applications.

Model-driven development (MDD) methods attempt to specify a system at a high level of abstraction in order to improve the capability for automation in the development and the quality of the applications[14]. The models are usually expressed in domain-specific languages (DSLs)[15]. MDD methods should provide model transformations to generate software artifacts from the models written by DSLs. Compared with general-purpose languages (GPLs), DSLs have at least the following advantages[9,16-17].

Concrete Expression of Domain Knowledge. As domain-specific functionality is coded in a concrete human-readable form at a high level of abstraction, software artifacts are not arcane for the developers. It will reduce the difficulty of developing, testing, and modifying.

Direct Involvement of the Domain Experts. A program expressed in DSL usually has a style which matches the format typically used by the domain experts. It helps domain experts take part in the lifecycle of the software and cooperate with the developers. The domain experts may even specify, implement, verify, and validate some artifacts directly.

In this work, we intend to propose a DSL for the domain of AOCA applications, named AocML. On the basis of the above advantages, this DSL is easier to learn for both developers and domain experts of these applications. The difficulty in learning API will be avoided. Furthermore, applications can be developed with less code by a concise DSL than by a GPL, so that the productivity may be improved.

AocML satisfies the following requirements.

Abstraction of Separated Concerns. As previously mentioned, the development method uses the idea of separation of concerns. Therefore, the separated concerns should be identified[18]. DSL should give a high-level abstraction of the concerns by analyzing the domain concepts.

Openness. Resources in the environment and personalized requests are often modified or added during the lifecycle. DSL should support incremental development. The artifacts of development should be lightweight and pluggable.

Ease of Use and Reuse. The syntax of the proposed DSL should be intuitive to use. The artifacts should be easy to reuse in similar scenarios.

Generally, a DSL has three elements[15]: the abstract syntax which describes the domain concepts as well as the relationship among them and is normally specified in a metamodel, the concrete syntax which is based on the abstract syntax and provided for the developers, and the semantics which is usually described by a translation to other languages, especially a GPL (such as Java).

Therefore, DSL development consists of three phases[19]: the analysis of domain-specific terminology in an abstract form, the design of concrete syntax, and the implementation of a code generator. For AOCA applications, we show these three phases in Section 3~Section 5, respectively.

3 Analysis: Concept Model of AOCA Applications

In this section, we discuss the concept model of AOCA applications for the lightweight, incremental development. This model consists of two parts: 1) the software model to describe the development and runtime artifacts; 2) the metamodel of the domain concepts to describe the abstract syntax of AocML.

3.1 Software Model of AOCA Applications

Fig.2 displays the static model of AOCA applications. This model separates the development artifacts into two parts, i.e., the infrastructure and the activity-related environment constraints of applications.

The infrastructure part describes the features in environment and the ability to sense or influence these features. This part is developed independently. The infrastructure can be shared by various applications. More specifically, it contains the followings.

Description of Owners. Owners describe entities which own the environment attributes. It contains not only the physical locations where the activities are performing but also the persons (users) who are holding the sensors or other devices. The sensors, services, and devices are deployed in the environment to sense or influence the attributes in the owners. This part describes the locations (e.g., rooms) and the users related to the system.

Description of Features. Features reflect the status of the owners, e.g., temperature of a room, blood pressure of a person. They indicate the types of environment data which can be obtained by the system. This part describes these concept-level features, such as light intensity, noise intensity, temperature, and blood pressure.

Description of Sensing Ability. Some devices or services (e.g., sensors) are deployed in the environment to obtain the specific features of specific owners. They indicate the sensing ability of the infrastructure. This part focuses on the deployment of each sensor or service by defining the feature which it senses (e.g., TelosB sensors are able to sense attributes with the feature of light intensity) and the related owner.

Description of Influencing Ability. Some devices controlled by the computer system are deployed in the environment to influence the specific features of the environment. They indicate the influencing ability of the infrastructure. This part focuses on the deployment of each device by defining the feature which it influences (e.g., lamps are able to influence attributes with the feature of light intensity) and the related owner.

In the application part, we only consider the activity-related environment constraints. Other requirements are beyond the scope of this work. The first step is to describe activity types involved in the applications. Then, the second step is to define the environment constraints which reflect the requests for the environment to ensure the smooth performance of the activities. These constraints can be considered separately oriented to the activities. There are two kinds of environment constraints.

• The default constraints related to specific activity

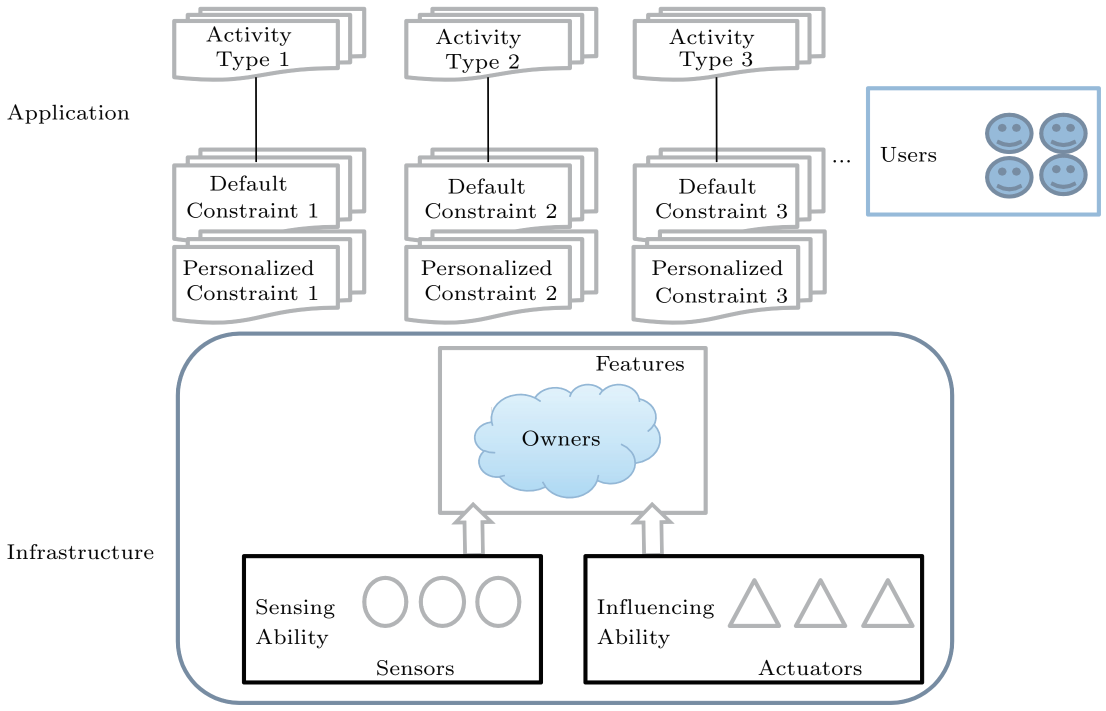


Fig.2. Static model of AOCA applications.

types reflect the general requests of each activity type. For example, the activity type "reading" usually requires the environment has suitable light intensity. The application developers define the default constraints on the basis of the understanding of an application.

• The personalized constraints of a specific user describe the user has special requests of an activity type. For example, for the activity type "reading", some users may require a brighter environment than general users. Furthermore, some users may require for some features beyond the scope of default constraints. For example, the blood pressure of elderly users should be monitored at any time, but this feature is usually not contained in the default constraints of general activity types. The application developers define the personalized constraints on the basis of the investigation of users.

In the runtime, AOCA applications composite development artifacts of infrastructure and environment constraints by a supporting system. We use the term "context" to combine an activity with related environment resources. Fig.3 shows the runtime model of AOCA applications.

In this system, pieces of context are organized in the runtime and managed by proactive components. The interactions among infrastructure part, context part, and application part (activities) consist of the following aspects.

• Context components generate context by organizing environment data according to constraints of current activity.

• When an activity changes, the constraints may change, so that the context will be regenerated.

• Context components influence the environment by using devices (e.g., lamp) to adjust some features.

• If the constraints of an activity cannot be satisfied, the context component will notify the user by some devices (e.g., smart phone).

3.2 Abstract Syntax of AocML

In order to describe the abstract syntax of AocML, we use a metamodel of AOCA applications. This metamodel which is shown in Fig.4 presents the main concepts and relationships of the software model. We briefly discuss these concepts as follows.

Feature. The infrastructure developers create these objects. The types of sensors/devices are combined with the features.

Location. The infrastructure developers create these objects when there are some sensors/devices deployed in a location.

Person. The infrastructure developers create these objects when a person is holding some sensors. If a person object has not been defined by the infrastructure developers, the application developers may create it when they need to define the personalized constraints.

Environment Attribute (*EnvAttribute*). The infrastructure developers create an environment attribute to describe an element in the environment that can be accessed. The instances of sensors/devices are combined with the environment attributes.

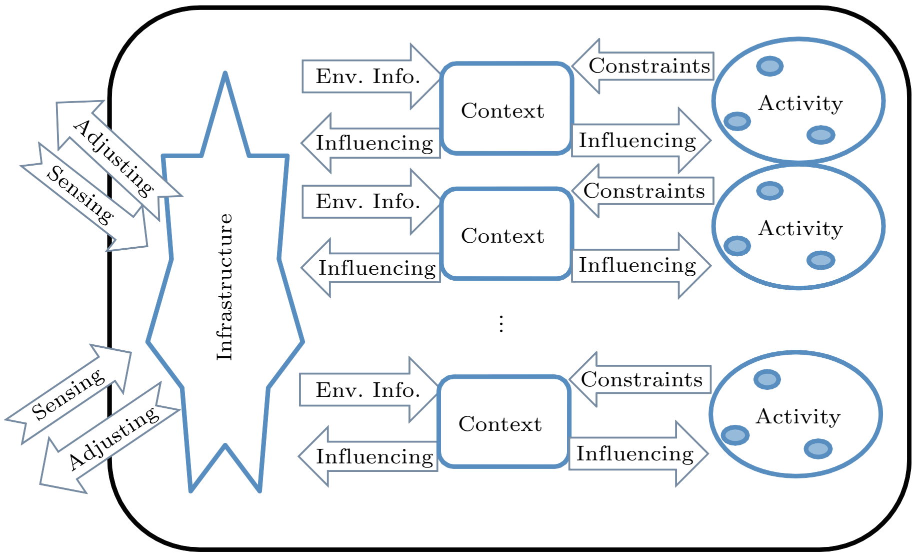


Fig.3. Runtime model of AOCA applications. Env. Info. means enviroment information.

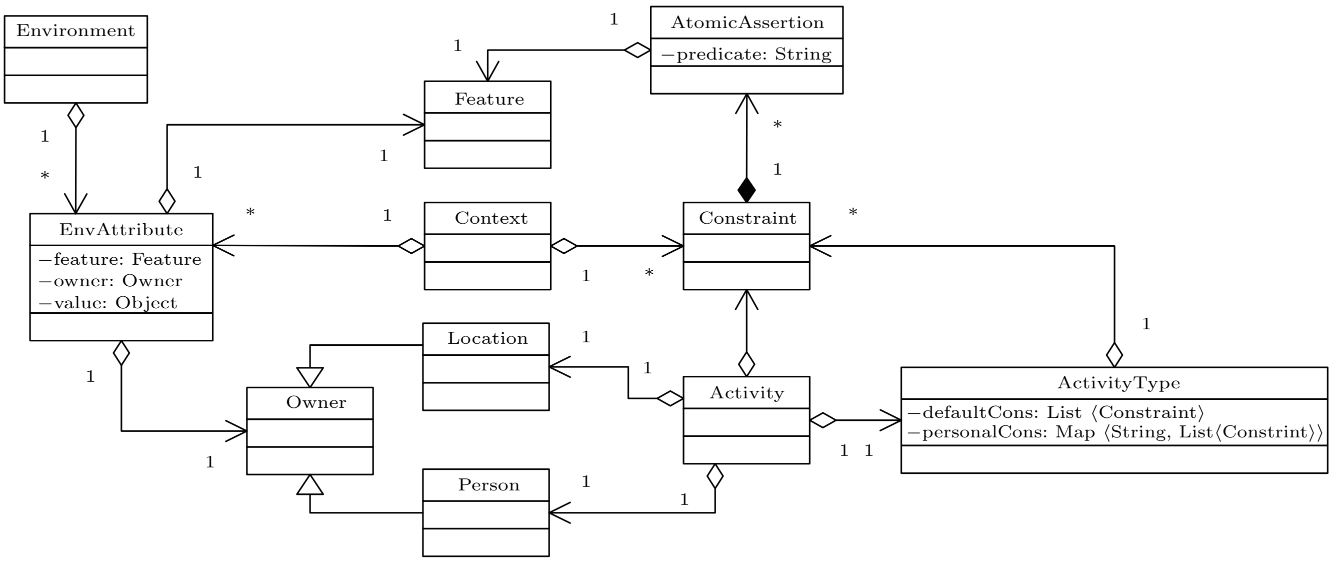


Fig.4. Metamodel of AOCA applications.

Environment. The supporting system organizes the environment attributes as an environment.

Activity Type. The application developers create these objects.

Constraint and Atomic Assertion. An atomic assertion is to judge whether environment attributes with a specific feature satisfy a condition (e.g., $<temperature, greaterThan(10)>$). A constraint is a set of atomic assertions. The application developers define the default constraints for activity types and the personalized constraints for persons.

Activity. In the runtime, the supporting system receives the results of activity recognition (e.g., [20]), and then creates these objects.

Context. Here, we follow Dourish's[21] understanding of the properties of context, i.e., the scope of context is defined dynamically; context is particular to each occasion of activity. In our model, a context[3] is considered as a set of environment attributes related to a specific activity. It also maintains the constraints of the activity. The supporting system creates these objects.

In summary, the tasks of the infrastructure developers are to define features, owners (locations/persons), and environment attributes; the tasks of the application developers are to define the activity types, default constraints, and personalized constraints.

4 Design: AocML Concrete Syntax

On the basis of the software model of AOCA applications and the abstract syntax of AocML, we propose a textual concrete syntax for creating development artifacts. This language is platform-independent. In this work, we implement this language by Xtext[22], a widely used framework for development of DSLs.

The details of the syntax are shown with BNF (Backus-Naur Form) in Fig.5. In this definition, ID and INT are two predefined terminals which indicate the identifier in Xtext and the integer type, respectively.

According to the software model, AocML-based development also contains two parts, i.e., environment infrastructure development and applications development.

4.1 Environment Infrastructure Development

In our method, each sensor or service which senses an environment attribute is encapsulated as a probe. Each device or service influencing an environment attribute is encapsulated as an actuator.

Fig.6 shows an example of environment infrastructure development. Lines 3∼8 define a feature. Besides the feature name (LightIntensity), the developers also need to declare the owner type (Location or Person) of the corresponding attribute and the related probe types, actuator types, operators. The probe types and actuator types specify the system ability of sensing and influencing the environment attributes with this feature. The operators specify the ability of judging the environment attributes with this feature. Line 9 defines an owner room810 by giving its name and type. There are two devices in room810. Line 10 defines a probe which encapsulates a light intensity sensor by giving its

```
<DomainModel>             ::=  <EnvModel> | <AppModel>
<EnvModel>                ::=  "envPackage" <QualifiedName> "{" {<Feature> | <Owner>}"}"
<Feature>                 ::=  "feature" <ID> "{"
                               "owner_type" <OwnerType>
                               "probe_type" <ProbeType> {<ProbeType>}
                               "actuator_type" {<ActuatorType>}
                               "operator" {<JudgeFun>}"}"
<ActuatorType>            ::=  <ID>
<ProbeType>               ::=  <ID>
<Owner>                   ::=  "owner" <ID> ":"<OwnerType>"{"{<Probe> | <Actuator>}"}"
<OwnerType>               ::=  "Location"|"Person"
<Actuator>                ::=  "actuator" <ID> ":"<ActuatorType>
<Probe>                   ::=  "probe" <ID> ":"<ProbeType>
<AppModel>                ::=  "appPackage" <QualifiedName> "{" {<ConstraintType>}"}"
<ConstraintType>          ::=  <DefaultConstraint> | <PersonalizedConstraint>
<DefaultConstraint>       ::=  "activity" <ID>
                               "constraint" {<Constraint>}
<PersonalizedConstraint>  ::=  "person" <ID>
                               "activity" <ID>
                               "constraint" {<Constraint>}
<Constraint>              ::=  <ID>":"[ "["<TimePeriod>"]" ] {<Assertion>}
<TimePeriod>              ::=  <ID>
<Assertion>               ::=  "assertion" <AtomicAssertion>{<AtomicAssertion>}
<AtomicAssertion>         ::=  <ID> ","<JudgeFun> ","<Threshold>
<QualifiedName>           ::=  <ID>{"."<ID>}
<JudgeFun>                ::=  ("<"|">")["="]|"="|"<>"|"inRange"
<Threshold>               ::=  <Float> |("["|"(")<Float>","<Float>(")"|"]")
<Float>                   ::=  ["+"|"-"]((<INT>["."<INT>])|("."<INT>))
```

Fig.5. AocML concrete syntax.

name and probe type. Similarly, line 11 defines an actuator which encapsulates a controllable lamp by giving its name and actuator type. With the given probe/actuator type and owner, the supporting system is able to combine the probe/actuator with an environment attribute.

```
1  envPackage demoEnv
2  {
3   feature LightIntensity{
4    owner_type Location
5     probe_type LightIntensityProbe
6      actuator_type Lamp
7       operator <= >= inRange
8   }
9   owner room810:Location {
10   probe pr1:LightIntensityProbe
11   actuator at1:Lamp
12  }
13 }
14
```

Fig.6. Environment infrastructure development example.

4.2 Application Development

The application developers deal with the constraints of activities. They need to specify default and personalized constraints of each activity type.

According to the idea in [3], a constraint can be specified as a conjunction of assertions. Each assertion is a disjunction of several atomic assertions. Furthermore, a label of time period (e.g., "At night", "17:00-18:00", "In Sunday") is optional.

$$constraint \equiv as_1 \wedge as_2 \wedge \ldots \wedge as_n : [time\ period],$$

$$as_k \equiv atoAs_1 \vee atoAs_2 \vee \ldots \vee atoAs_m,$$

where as_k is an assertion, and $atoAs_i$ is an atomic assertion.

Fig.7 shows an example of application development. Lines 3~10 specify the default constraints of activity types Sleeping, WakingUp, and Reading. For the activity Reading, the environment should meet at least one condition of "the curtain is rolled up" and "the lamp is turned on". Lines 12~16 specify the personalized constraints of Bob. Bob needs a darker and warmer environment to sleep. He also needs a suitable environment (the light intensity should not be too high) when he wakes up at night.

```
1  appPackage demoApp
2  {
3   activity Sleeping constraint consSleep:
4    assertion LightIntensity, <=, 80
5    assertion Temperature, inRange, [20,24]
6    assertion NoiseIntensity, <=, 30
7   activity WakingUp constraint consWakeUp:
8    assertion LightIntensity, >=, 200
9   activity Reading constraint consRead:
10   assertion CurtainUp, =, 1 or LampOn, =, 1
11
12  person Bob activity Sleeping constraint consBS:
13    assertion LightIntensity, <=, 50
14    assertion Temperature, inRange, [22,28]
15  person Bob activity WakingUp constraint consBW:
16      [AtNight] assertion LightIntensity, inRange, [130,250]
17 }
18
```

Fig.7. Application development example.

5 Implementation: Tools for AocML

The proposed AocML represents the development artifacts of AOCA applications at a high-level of abstraction. From the perspective of MDA (Model-Driven Architecture)[23], it is a PIM (platform-independent model). In this work, we implement the development support for this DSL. Furthermore, we provide a model-to-text transformation to convert the AocML models into Java codes. These codes are generated for the existing runtime supporting platform PAOC[3]. As AocML model is not complicated in the mapping between model and code, we transform the model to code directly instead of introducing model-to-model transformations to a PSM (platform-specific model). Therefore, this transformation has some advantages, for instance, the debugging task will be easier, the development time will be reduced, and the transformation is faster[24]. Apart from the Java codes, we provide another transformation to generate OWL specification for AocML ontology base.

The development phase is shown in Fig.8.

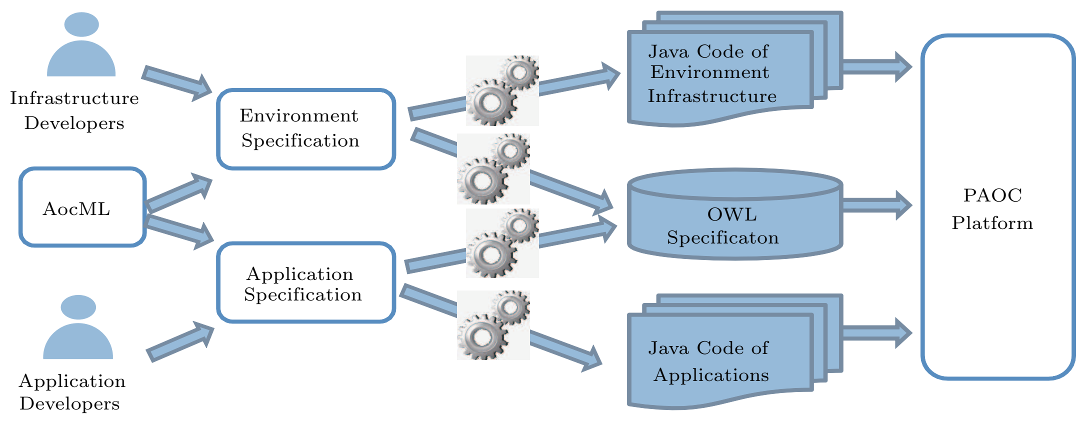

Infrastructure Developers
AocML
Application Developers
Environment Specification
Application Specification
Java Code of Environment Infrastructure
OWL Specificaton
Java Code of Applications
PAOC Platform

Fig.8. Development phase based on AocML.

5.1 AOCA Applications Development Support

Generally, the basic tools for a textual DSL contain[25]:

1) an editor to develop applications with the DSL,

2) a parser (model injector) to extract models from a DSL textual specification, and

3) a code generator to transform the models into software development artifacts.

In this work, we use Xtext[22] to implement the development environment of AocML. Xtext provides a framework to specify the grammars of DSLs and automatically generate domain-specific editors. This tool also includes components such as a parser, a linker, a type checker, and a compiler integrated by EMF (Eclipse Modeling Framework). It is easy for DSL developers to design a DSL and implement an Eclipse-based IDE (integrated development environment). IDE has some utility functions such as highlighting of keywords and display of syntax errors.

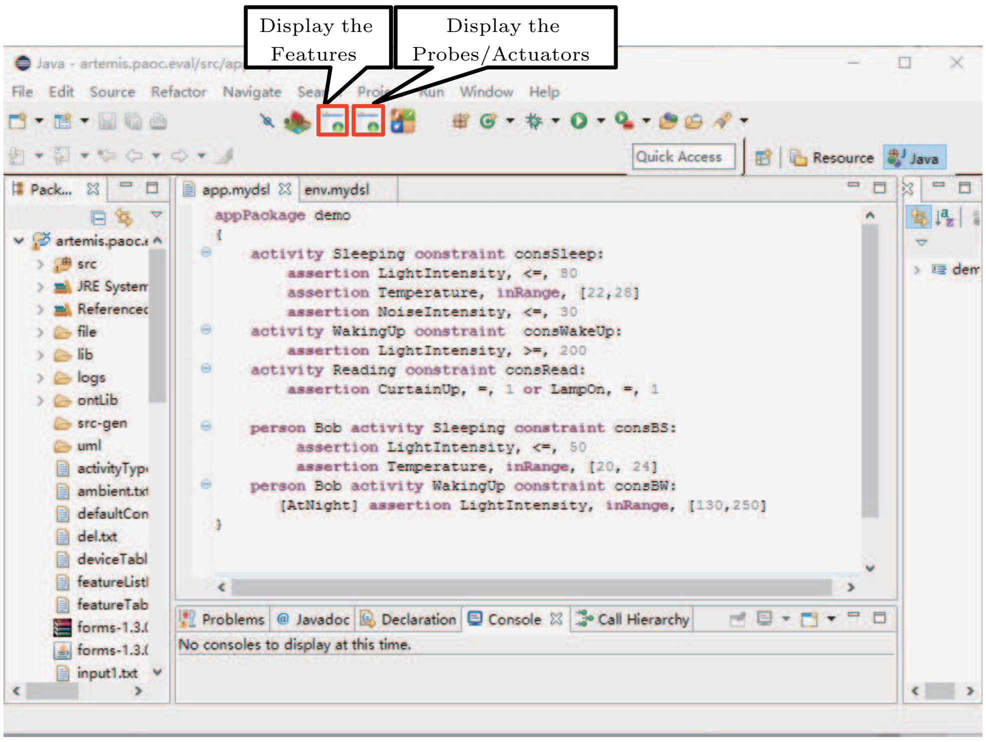
Fig.9 shows the implemented IDE based on Xtext. We provide not only the infrastructure development and application development interfaces, but also some other utility tools, e.g., buttons for invoking JFrame-based tables which display the specified features and probes/actuators.

5.2 Java Code Generation from AocML Model

PAOC (platform for activity-oriented context)[3] is a Java-based platform for supporting the development and runtime of AOCA applications. This platform provides a Java API for deploying the resources in environment infrastructure and specifying constraints of activities. The interfaces of PAOC API are based on an object-oriented model which is similar to the meta-model of AOCA applications (Fig.4). The main concepts of API are consistent with the abstract syntax of AocML. Therefore, developers of AOCA applications can specify the infrastructure and constraints with AocML, instead of using PAOC API.

In order to generate Java code from the developed AocML model automatically, we implement a code generator with Xtend②. This generator parses every ele-

②http://www.eclipse.org/xtend/, July 2018.



```
appPackage demo
{
    activity Sleeping constraint consSleep:
        assertion LightIntensity, <=, 80
        assertion Temperature, inRange, [22,28]
        assertion NoiseIntensity, <=, 30
    activity WakingUp constraint  consWakeUp:
        assertion LightIntensity, >=, 200
    activity Reading constraint consRead:
        assertion CurtainUp, =, 1 or LampOn, =, 1

    person Bob activity Sleeping constraint consBS:
         assertion LightIntensity, <=, 50
         assertion Temperature, inRange, [20, 24]
    person Bob activity WakingUp constraint consBW:
       [AtNight] assertion LightIntensity, inRange, [130,250]
}
```

```
No consoles to display at this time.
```

Fig.9. IDE for AOCA applications.

ment in an AocML model and creates the corresponding Java code. This generator is integrated in the Xtext-based IDE of AocML.

As mentioned in Fig.8, the model-to-text transformations are also separated into two parts. Fig.10 presents an example of generating code from the AocML environment model which is shown in Fig.6. The generation is organized in the following way.

```
1  public class DemoEnv {
2   public static void setInfrastructure () {
3       FeatureManager fm=FeatureManager.getInstance();
4       ProbeManger pm=ProbeManger.getInstance();
5       ActuatorManager am=ActuatorManager.getInstance();
6       OwnerManager om=OwnerManager.getInstance();
7
8       Feature fea1=fm.createFeature("LightIntensity");
9       fea1.setOwnerType("Location");
10      pm.getProbeType("LightIntensityProbe").setRelatedFeature(
            fea1);
11      am.getActuatorType("Lamp").setRelatedFeature(fea1);
12      fea.setJudgeFun("<=");
13      fea.setJudgeFun(">=");
14      fea.setJudgeFun("inRange");
15
16      om.createOwner("Location","room810");
17
18      Probe pr1=pm.createProbe("LightIntensityProbe","pr1");
19      pr1.setOwner("room810");
20
21      Actuator at1=am.createActuator("Lamp","at1");
22      at1.setOwner("room810");
23    }
24 }
25
```

Fig.10. Example of code generated for environment model.

• Instances of PAOC managers are created (lines 3∼6, Fig.10). These managers are designed with the singleton pattern[26] so that these instances are globally unique for each manager.

• A feature is created (lines 8∼14, Fig.10) by setting its owner type (setOwnerType method), related probe types and actuator types (setRelatedFeature method of probe/acutator type), and related operators (setJudgeFun method). The getProbeType/getActuatorType method is invoked to get a probe/acutator type. If the probe/acutator type is null, the system will create a type object and return it.

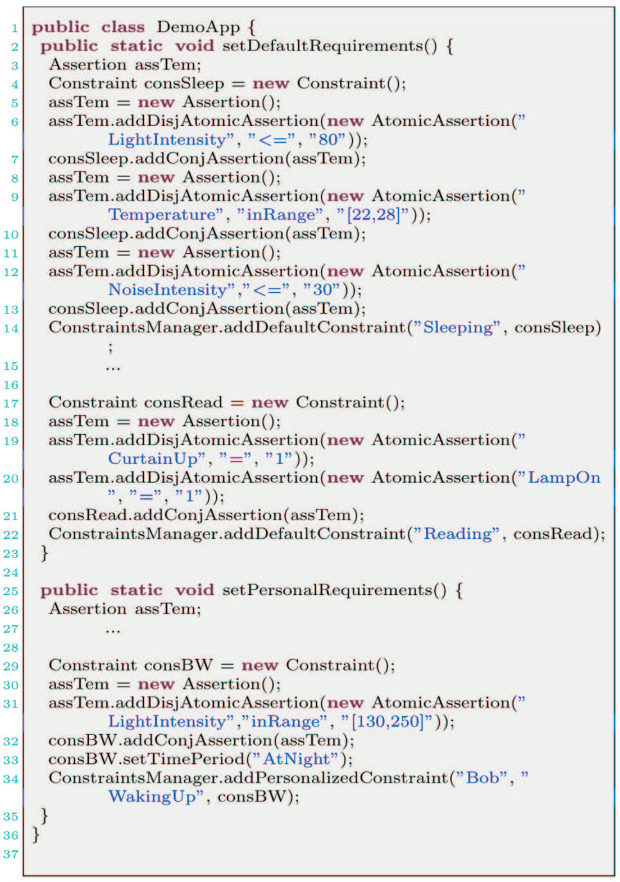
• An owner is created (line 16, Fig.10) by giving its type and name.

• A probe/actuator is created (lines 18∼22, Fig.10) by giving its type and name, together with setting its owner.

Fig.11 presents a fragment of generating code from the AocML application model which is shown in Fig.7. The generation is organized in the following way.

• Two specific methods, setDefaultRequirements and setPersonalRequirements, are created for the specifications of default constraints and personalized constraints, respectively.

• Each atomic assertion is created, and then added as a disjunctive term of an assertion by the method addDisjAtomicAssertion. Lines 18∼20 in Fig.11 show an example of an assertion with multiple atomic assertions.



```
 1 public class DemoApp {
 2  public static void setDefaultRequirements() {
 3   Assertion assTem;
 4   Constraint consSleep = new Constraint();
 5   assTem = new Assertion();
 6   assTem.addDisjAtomicAssertion(new AtomicAssertion("
          LightIntensity", "<=", "80"));
 7   consSleep.addConjAssertion(assTem);
 8   assTem = new Assertion();
 9   assTem.addDisjAtomicAssertion(new AtomicAssertion("
          Temperature", "inRange", "[22,28]"));
10   consSleep.addConjAssertion(assTem);
11   assTem = new Assertion();
12   assTem.addDisjAtomicAssertion(new AtomicAssertion("
          NoiseIntensity","<=", "30"));
13   consSleep.addConjAssertion(assTem);
14   ConstraintsManager.addDefaultConstraint("Sleeping", consSleep)
          ;
15        ...
16
17   Constraint consRead = new Constraint();
18   assTem = new Assertion();
19   assTem.addDisjAtomicAssertion(new AtomicAssertion("
          CurtainUp", "=", "1"));
20   assTem.addDisjAtomicAssertion(new AtomicAssertion("LampOn
          ", "=", "1"));
21   consRead.addConjAssertion(assTem);
22   ConstraintsManager.addDefaultConstraint("Reading", consRead);
23  }
24
25  public static void setPersonalRequirements() {
26   Assertion assTem;
27        ...
28
29   Constraint consBW = new Constraint();
30   assTem = new Assertion();
31   assTem.addDisjAtomicAssertion(new AtomicAssertion("
          LightIntensity","inRange", "[130,250]"));
32   consBW.addConjAssertion(assTem);
33   consBW.setTimePeriod("AtNight");
34   ConstraintsManager.addPersonalizedConstraint("Bob", "
          WakingUp", consBW);
35  }
36 }
37
```

Fig.11. Example of code generated for application model.

• An assertion is organized as a conjunctive term of a constraint by the method addConjAssertion (lines 7, 10, 13, 21, and 32, Fig.11).

• A constraint is added as a default constraint of an activity by the method addDefaultConstraint (lines 14, 22, Fig.11) or a personalized constraint related to a user by the method addPersonalizedConstraint (line 34, Fig.11).

• The method setTimePeriod is used to set a time period for a constraint (line 33, Fig.11).

5.3 OWL Specification Generation from AocML Model

In the supporting platform, we use an ontology base to maintain and share knowledge among every module. In computer science, ontology is "an explicit specification of a conceptualization"[27]. The same with general ontology bases, AocML ontology also has two levels:

• a TBox which describes the concepts and properties is usually related to object-oriented classes;

• an ABox which describes the facts, and is usually related to instances of classes.

The ontology base is described with OWL (Web Ontology Language)[28]. The TBox of AocML ontology is manually developed in accordance with the metamodel of AOCA applications (Fig.4). Fig.12 presents the main concepts and properties in this TBox. The TBox is valid for all applications supported by AocML.

The OntologyBase class, which is designed with the singleton pattern, maintains the ontology base. The

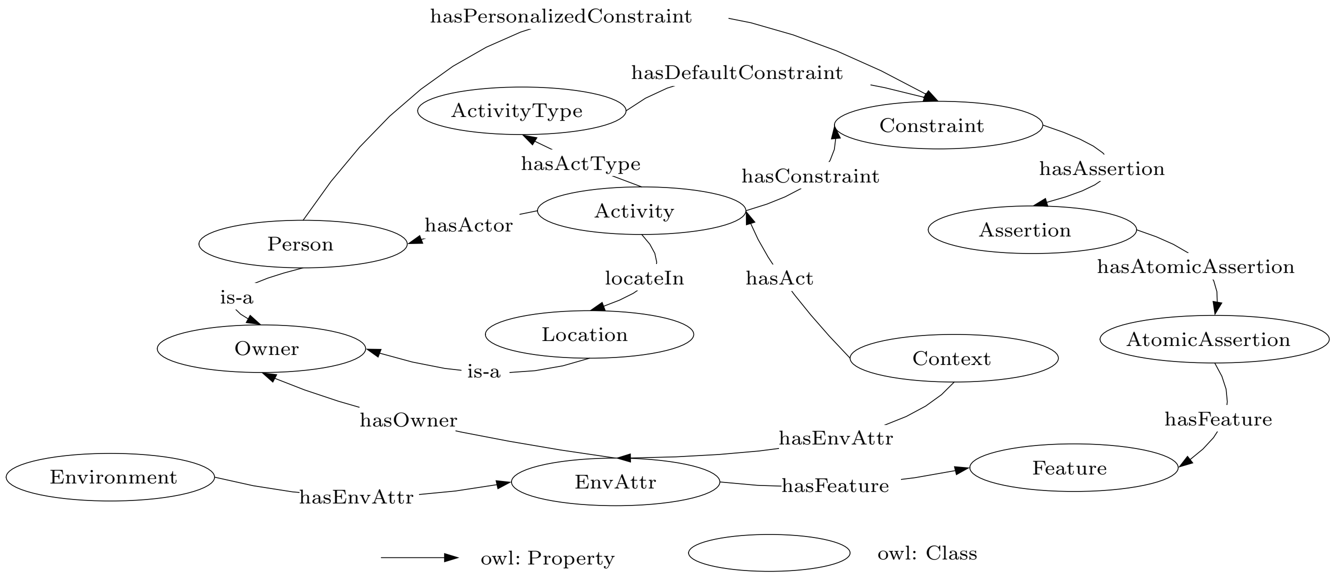


Fig.12. Main concepts and properties in the TBox of AocML ontology.

main related classes and interfaces are shown in Fig.13. All the classes of metamodel concepts implement the OntResource interface. This interface indicates that these classes are ontology resources, so that each class corresponds to a concept in the TBox. When the elements of AOCA applications are created or changed, the relevant instances in the ABox of AocML ontology will be created or changed. The class AocMLOntModel encapsulates OWLModel in Protégé API[3] and provides utility interfaces such as construction of a model, input and output of a model from an OWL file, operations on classes and instance (e.g., creating resources, deleting resources, adding property values). The Reasoner class encapsulates the inference engine of Protégé, which is based on Jess[4]. This class supports inferences on the instances.

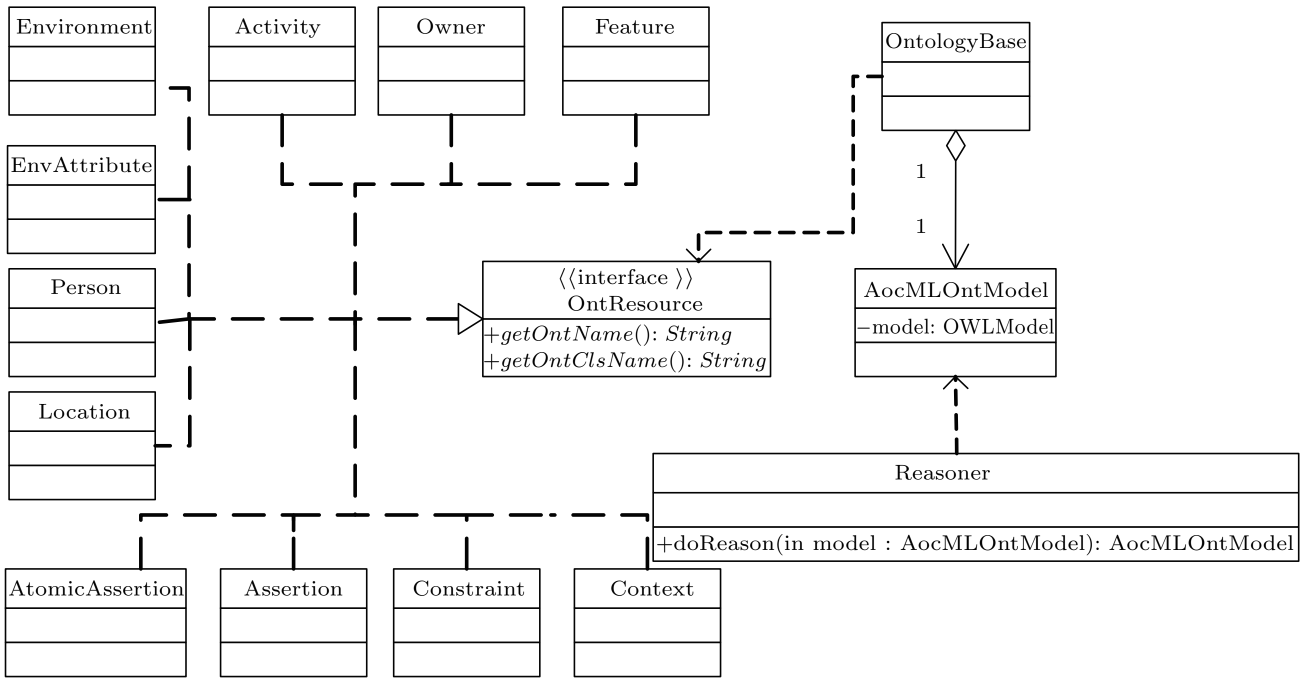


Fig.13. Class diagram of the ontology base.

During the development, when the supporting system parses the elements in an AocML model, the instances corresponding to the elements are created in the ontology base. Meanwhile, the relationships among the instances are added as the property values. Note that hasPersonalizedConstraint is a ternary relationship among Person, ActivityType, and Constraint, which is difficult to be described by OWL. In our implementation, we add the name of the activity type as the prefix of a constraint name, so that it can be viewed as a binary relationship between Person and Constraint.

6 Case Study

In this section, we introduce the development of a smart meeting room as a case study. We firstly overview the scenario and then present the specified AocML model of the infrastructure and application. Furthermore, we discuss about some design guidelines.

6.1 Scenario: A Smart Meeting Room

Smart meeting room, which is a practical pervasive computing scenario, draws widely attention in academia and industry. The basic task of such a system is to perceive the environment and control the devices in the room in order to ensure the smooth progress of the meetings. As it is a typical scenario, many researchers use it as a case study to show the practicability of their work. Different researchers organize this application in different ways on the basis of their research ideas. For example, Context Toolkit[7] organizes the system surrounding the Conference Assistant application in handheld devices of meeting attendees. This system acquires

[3]http://protege.stanford.edu/, July 2018.

[4]http://www.jessrules.com/, July 2018.

the environment information and pushes some suggestions to the attendees. Another work, FollowMe[5], organizes the system in the form of workflow. The pieces of environment information are used as events for triggering some activities in the workflow.

In our work, we view the smart meeting room scenario in the perspective of AOCA applications. More specifically, the development artifacts are separated into two parts.

Infrastructure of a Smart Meeting Room. The hardware devices and software services which are able to sense or influence environment information are encapsulated in the infrastructure of the smart meeting room system. The development of this part is independent of the application requirements. The typical sensors include TelosB sensors (Fig.14(a)) which are able to sense light intensity, temperature, humidity and IRIS sensors (Fig.14(b)) which are able to sense noise intensity. The typical services include the weather forecast based on web service. The devices in the system (e.g., lamps, curtains) can be controlled by some approaches, such as sending control signals by the Modbus protocol⑤. The status of the devices (e.g., whether a lamp is turned on, whether a curtain is rolled up) can be acquired by the query mechanism of the Modbus protocol, so that we also view these status as environment attributes. The infrastructure of a smart meeting room can be designed as part of a large infrastructure such as a smart office building.

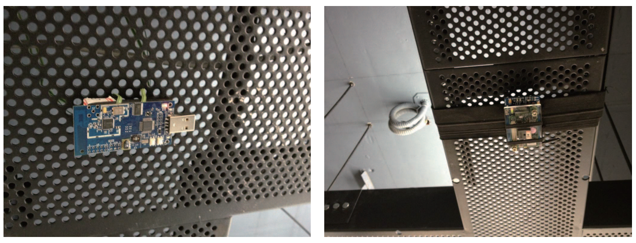


Fig.14. Examples of sensors. (a) TelosB. (b) IRIS.

Requirements of Smart Meeting Room Applications. The users of these applications are the attendees of the meetings. Application requirements are organized with activity types of the attendees. For example, an academic report in the meeting room has multiple phases such as reporters entering the room, audience entering the room, reporting, discussing after the report, and attendees leaving the room. These phases are recognized by the system and then described as different activities of different users. Developers specify the constraints of the activities. The system supports the performing of these activities.

6.2 Development with AocML

Fig.15 shows the AocML-based environment specification of a smart meeting room. This system defines six features: LightIntensity, Temperature, ProjectorOn, PosterOn, CurtainUp, and MusicOn. A meeting room room810 is defined as a location. A TelosB sensor is deployed in the meeting room in order to sense the light intensity and temperature. It is encapsulated into two probes p101 and p102. A lamp a201 will

```
envPackage demoEnv
{
 feature LightIntensity{
  owner_type Location
   probe_type LightIntensityProbe
    actuator_type Lamp
     operator <= >= inRange
 }
 feature Temperature{
  owner_type Location
   probe_type TemperatureProbe
    actuator_type AirConditioner
     operator <= >= inRange
 }
 feature ProjectorOn{
  owner_type Location
   probe_type ProjectorProbe
    actuator_type Projector
     operator =
 }
 feature PosterOn{
  owner_type Location
   probe_type PosterProbe
    actuator_type Poster
     operator =
 }
 feature CurtainUp{
  owner_type Location
   probe_type CurtainProbe
    actuator_type Curtain
     operator =
 }
 feature MusicOn{
  owner_type Location
   probe_type MusicProbe
    actuator_type MusicPlayer
     operator =
 }
 owner room810:Location {
  probe p101:LightIntensityProbe
  probe p102:TemperatureProbe
  probe p103:ProjectorProbe
  probe p104:lightIntensityProbe
  probe p105:CurtainProbe
  probe p106:MusicProbe
  actuator a201:Lamp
  actuator a202:AirConditioner
  actuator a203:Projector
  actuator a204:Poster
  actuator a205:Curtain
  actuator a206:MusicPlayer
 }
}
```

Fig.15. Environment specification of a smart meeting room.

⑤http://www.modbus.org/, July 2018.

influence the light intensity. An air conditioner a202 will influence the temperature. There are four more devices that can be controlled: a projector a203, a poster a204, a curtain a205, and a music player a206. Their statuses are acquired by virtual probes p103, p104, p105, and p106 respectively.

Fig.16 shows the AocML-based application specification of a smart meeting room. We define five activity types and the following default constraints.

• *ReporterEntering*. The light intensity and temperature should be suitable; the curtain should be rolled up; the projector should be turned on.

• *AudienceEntering*. The music player and the poster should be turned on to welcome the audience.

• *Reporting*. The music player should be turned off; the environment should be darker for the sharpness of the projector.

• *Discussing*. The environment should be brighter for the discussing.

• *Leaving*. The devices such as projector and poster should be turned off.

Apart from these default constraints, we also give an example of personalized constraints.

```
appPackage demoApp
{
    activity ReporterEntering constraint consRepEnter:
        assertion LightIntensity, >=, 500
        assertion CurtainUp, =, 1
        assertion Temperature, inRange, [22,28]
        assertion ProjectorOn, =, 1
    activity AudienceEntering constraint consAudEnter:
        assertion MusicOn, =, 1
        assertion PosterOn, =, 1
    activity Reporting constraint consReport:
        assertion MusicOn, =, 0
        assertion LightIntensity, inRange, [100,300]
    activity Discussing constraint consDiscuss:
        assertion LightIntensity, >=, 500
    activity Leaving constraint consLeave:
        assertion CurtainUp, =, 0
        assertion MusicOn, =, 0
        assertion PosterOn, =, 0
        assertion ProjectorOn, =, 0
    person Bob activity AudienceEntering constraint
        consBobEnter:
        assertion Temperature, >=, 25
}
```

Fig.16. Application specification of a smart meeting room.

6.3 Discussion

In Subsection 6.2, we present a case study of design and implementation of AOCA applications. We further discuss about some notices and suggestions of the development.

1) One of the key design steps is the definition of features. Features contain not only the properties such as light intensity but also the status of devices such as projectors. We suggest that it is better to define features as the basic information which can be acquired directly. The requirements related to more than one feature can be specified as the combination of constraints. However, infrastructure developers can also define complex features in specific applications. Probes for the fusion of features can be developed.

2) In an application, we may use different ways to define the activity types. In the smart meeting room case, we define two activity types "a reporter is entering" (ReporterEntering) and "an audience is entering" (AudienceEntering). In the runtime, these activity types are separated in the activity recognition, i.e., when a user is entering the meeting room, the system sets one of the two activity types on the basis of her/his identity. An alternative way is to define an activity type Entering. Then personalized constraints for users with different identities can be specified.

3) The controlling of a device may be triggered by different features. In the smart meeting room case, the lamp can be triggered by the constraints related to light intensity. However, we can also define a feature to describe the status of the lamp (e.g., LampOn). Then we can define constraints related to this feature. The issue about which way is more appropriate depends on the scenario. Generally, if the application focuses on a property of the space (especially when multiple devices can influence the property, e.g., light intensity), the better way is to define the property as a feature directly. The infrastructure will decide how to influence the feature. In contrast, if there are requests for a device status, the better way is to define the constraints about the status. In the example above, if we want to ensure the lamp is turned off when the attendees are leaving, we can define a feature LampOn to describe the status of the lamp. Then we can define constraints about this feature instead of using constraints related to light intensity.

7 Evaluation

In this section, we evaluate the proposed approach to demonstrate that AocML facilitates the development of AOCA applications.

As mentioned in Section 2, the purpose of introducing DSL in AOCA applications is to reduce the difficulty of learning the approach and improve the productivity. Therefore, we need to answer the following research questions.

*RQ*1. Is AocML easier to read and understand?

*RQ*2. Does AocML improve the productivity?

7.1 Evaluation for RQ1

In comparison with GPLs, AocML provides keywords which are easier for domain developers to understand. Furthermore, the structure of the development artifacts is also specific for AOCA applications.

There are some widely used metrics[29-30] to measure the complexity of a language. In order to evaluate the readability and understandability, we choose the following metrics:

- TERM: number of grammar terminals;
- VAR: number of grammar non-terminals.

We evaluate these values of AocML and show the results in Table 1. It is compared with two GPLs Java 1.5 and ANSI C (the results is shown in [30]), together with one DSL MLContext[25], which is designed for a different domain of pervasive computing.

Table 1. Results of Comparison in Grammar Metrics

Metric	Java 1.5	ANSI C	MLContext	AocML
TERM	102	83	24	33
VAR	129	66	8	21

7.2 Evaluation for RQ2

The improvement in productivity is often evaluated by lines of code (LOC). We take the smart meeting room scenario (in Section 6) as an example. In this paragraph, LOC of the development with AocML is compared with that of PAOC API[3]. We also use the open source version of the classical tool Context Toolkit⑥ to develop the smart meeting room infrastructure and applications by widgets[7] as an example of general context-aware development methods. The results are shown in Table 2.

Table 2. Results of Comparison in LOC

Metric	Context Toolkit	PAOC API	AocML
LOC	423	138	76

7.3 Summary

Based on the results of the two evaluations, we answer the research questions as follows.

*RQ*1. AocML is easier to read and understand in comparison with GPLs such as Java and ANSI C. This benefit is similar to most DSLs. Although some other DSLs (such as MLContext) may have less grammar terminals and non-terminals than AocML, they are not designed for AOCA applications. AocML is a more understandable approach for AOCA applications. The domain experts can be involved in the development directly.

*RQ*2. AocML improves the productivity in comparison with general context-aware development methods as well as the PAOC API for AOCA applications. A few lines of AocML code are used to replace more GPL code. Although PAOC API also encapsulates the knowledge of AOCA applications, AocML provides more benefits such as specialized syntax and syntax error checking in the development environment.

8 Related Work

Pervasive and context-aware computing is an emerging research area. A considerable amount of research efforts focus on the development method of such applications. Some approaches, such as Context Toolkit[7] and Solar[31], propose APIs supporting the development with GPLs. There is also some existing work focusing on the model-driven development (MDD) of these applications or designing DSLs for the development. In this section, we compare AocML with these approaches in terms of the following properties.

- *Application Domain.* Which domain is the MDD method or the DSL tailored to?
- *Tool for Development.* What kind of tools is designed and implemented for the developers of domain applications?
- *Transformation.* Does the approach support model transformation or code generation? If so, what kind of model or code will be generated?
- *Scalability.* How does the approach support the change of the development artifacts? How can a developer reuse the existing artifacts?

This comparison is shown in Table 3. We learn from the comparison that the most significant difference between AocML and other approaches is the application domain. Besides PAOC[3], some other existing researches (e.g., [32-33]) also consider the development based on activities. Furthermore, some efforts (e.g., Egospace[34]) separate the development of infrastructure and application requirements. However, a suitable DSL for AOCA applications has not been proposed so far.

Apart from the application domain, we summarize the other three properties of AocML as follows.

⑥http://contexttoolkit.sourceforge.net/, July 2018.

Table 3. Comparison of the DSLs for Model-Driven Development of Pervasive and Context-Aware Computing

Approach	Application Domain	Tool for Development	Transformation	Scalability
PervML[24]	Services in pervasive computing	UML profile and a general constraint language	Generation of Java code and ontology	Rewrite UML profile; modify constraints
MLContext[25]	Context modeling	Specific textual DSL for entities and context sources	Generation of Java code and ontology	Application-independent, code for the reuse among applications
ContextUML[37]	Context-aware web services	UML profile	No	Rewrite UML profile
WebML[38]	Context-aware web applications	UML profile	Application-independent generation of Java code	Rewrite UML profile
Ayed *et al.*[39]	General context collection and adaptation	UML profile	PIM to PSM	Rewrite UML profile
CML[40]	General context-aware applications	Graph-based model; textual DSL	Generation of database scripts	Reuse the unchanged code; modify the changed parts
RAPPT[41]	Mobile applications	Specific textual DSL for the scaffolding of a mobile application	Generation of Android projects	Modify the changed description code
PerLa[42]	Pervasive information systems	An SQL-like language for data management in pervasive systems	No	Modify the changed SQL-like code
Ctrl-F[43]	Self-adaptive component-based architecture	Specific textual DSL for control policies	Translation to finite state automata models	Rewrite the changed component controllers
Kulkarni *et al.*[44]	Context-aware CSCW (computer-supported cooperative work) applications	Specific textual DSL for environment specification and policies	Generation of an application's execution environment	Reuse the unchanged environment and policies, modify the changed parts
Mobicon[45]	Mobile context-monitoring platform	Specific declarative query language for context monitoring query (CMQ)	No	Modify the changed CMQ
AocML	AOCA applications	Specific textual DSL for infrastructure and application specification	Generation of Java code and ontology	Reuse the unchanged code; add or modify the changed parts as lightweight plug-ins

1) *Tool for Development.* A number of existing studies use UML profile or UML extension[35] as a modeling language. However, UML is a GPL. The extension of domain concepts is restricted[25,36]. In this work, we propose a textual DSL which is beneficial to describe domain elements.

2) *Transformation.* We implement the automatic generation of Java code and ontology from the AocML model. The reduction of manual code will promote development efficiency and reduce the number of bugs.

3) *Scalability.* The fragments specified by AocML can be used as lightweight plug-ins for the reuse among multiple applications. This approach gives the benefit of productivity.

9 Conclusions

In this paper, we proposed a DSL for model-driven development of AOCA applications in order to facilitate the development. We analyzed the concept model of such applications, and designed the AocML abstract syntax and concrete syntax. We also implemented a tool for supporting AocML-based development, including the automatic generation of Java code and ontology. The case study and evaluation demonstrated that this approach provides the developers a suitable and practical tool for developing AOCA applications.

As future work, we plan to combine the AocML approach with formal methods. A possible way is to analyze the domain model by a formal method and transform the artifacts into a formal system. This formal system can be used to verify the applications in order to enhance the reliability. Furthermore, we plan to make attempts on the DSLs of related application domains in pervasive computing.

Acknowledgment The authors thank the anonymous reviewers for their helpful feedback.

References

[1] Weiser M. The computer for the 21st century. *Scientific American*, 1991, 265(3): 94-104.

[2] Satyanarayanan M. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 2001, 8(4): 10-17.

[3] Li X, Tao X, Lu J. Towards a programming framework for activity-oriented context-aware applications. *Frontiers of Computer Science*, 2017, 11(6): 987-1006.

[4] Gu T, Pung H, Zhang D. Toward an OSGi-based infrastructure for context-aware applications. *IEEE Pervasive Computing*, 2004, 3(4): 66-74.

[5] Li J, Bu Y, Chen S, Tao X, Lu J. FollowMe: On research of pluggable infrastructure for context-awareness. In *Proc. the 20th International Conference on Advanced Information Networking and Applications, Volume 1*, April 2006, pp.199-204.

[6] Arcelus A, Jones M H, Goubran R, Knoefel F. Integration of smart home technologies in a health monitoring system for the elderly. In *Proc. the 21st Int. Conf. Advanced Information Networking and Applications*, *Volume 2*, May 2007, pp.820-825.

[7] Dey A, Abowd G, Salber D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction* 2001, 16(2): 97-166.

[8] Gu T, Pung H K, Zhang D Q. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 2005, 28(1): 1-18.

[9] Voelter M, Benz S, Dietrich C, Engelmann B, Helander M, Kats L C, Visser E, Wachsmuth G. DSL Engineering: Designing, Implementing and Using Domain-Specific Languages. CreateSpace Independent Publishing Platform, 2013.

[10] Kamma D, Sasi K G. Effect of model based software development on productivity of enhancement tasks — An industrial study. In *Proc. the 21st Asia-Pacific Software Engineering Conference*, December 2014, pp.71-77.

[11] Mellegard N, Ferwerda A, Lind K, Heldal R, Chaudron M. Impact of introducing domain-specific modelling in software maintenance: An industrial case study. *IEEE Transactions Software Engineering*, 2016, 42(3): 245-260.

[12] Engeström Y, Miettinen R, Punamäki R L. Perspectives on Activity Theory. Cambridge University Press, 1999.

[13] Baldauf M, Dustdar S, Rosenberg F. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2007, 2(4): 263-277.

[14] Selic B. Personal reflections on automation, programming culture, and model-based software engineering. *Automated Software Engineering*, 2008, 15(3/4): 379-391.

[15] Kleppe A. Software Language Engineering: Creating Domain-Specific Languages Using Metamodels (1st edition). Addison-Wesley Professional, 2008.

[16] Freudenthal M. Domain specific languages in a customs information system. *IEEE Software*, 2010, 27(2): 65-71.

[17] Spinellis D. Notable design patterns for domain-specific languages. *Journal of Systems and Software*, 2001, 56(1): 91-99.

[18] Hürsch W L, Lopes C V. Separation of concerns. Technical Report NU-CCS-95-03, Northeastern University, 1995. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=0D0343A1A144A43A5687C675AF2766C0?doi=10.1.1.125.2723&rep=rep1&type=pdf, July 2018.

[19] Mernik M, Heering J, Sloane A M. When and how to develop domain-specific languages. *ACM Computing Surveys*, 2005, 37(4): 316-344.

[20] Wang L, Gu T, Tao X, Chen H, Lu J. Recognizing multi-user activities using wearable sensors in a smart home. *Pervasive and Mobile Computing*, 2011, 7(3): 287-298.

[21] Dourish P. What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 2004, 8(1): 19-30.

[22] Eysholdt M, Behrens H. Xtext: Implement your language faster than the quick and dirty way. In *Proc. the 25th ACM International Conference Companion on Object-Oriented Programming, Systems, Languages, and Applications Companion*, October 2010, pp.307-309.

[23] Kleppe A G, Warmer J, Bast W. MDA Explained: The Model Driven Architecture: Practice and Promise (1st edition). Addison-Wesley Professional, 2003.

[24] Serral E, Valderas P, Pelechano V. Towards the model driven development of context-aware pervasive systems. *Pervasive and Mobile Computing*, 2010, 6(2): 254-280.

[25] Hoyos J R, García-Molina J, Botía J A. A domain-specific language for context modeling in context-aware systems. *Journal of Systems and Software*, 2013, 86(11): 2890-2905.

[26] Gamma E, Helm R, Johnson R, Vlissides J, Booch G. Design Patterns: Elements of Reusable Object-Oriented Software (1st edition). Addison-Wesley Professional, 1994.

[27] Gruber T R. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 1993, 5(2): 199-220.

[28] Bechhofer S. OWL: Web ontology language. In *Encyclopedia of Database Systems*, Liu L, Özsu M T (eds.), Springer, 2009, pp.90-154

[29] Power J F, Malloy B A. A metrics suite for grammar-based software. *Journal of Software Evolution and Process*, 2004, 16(6): 405-426.

[30] Crepinsek M, Kosar T, Mernik M, Cervelle J, Forax R, Roussel G. On automata and language based grammar metrics. *Computer Science and Information Systems*, 2010, 7(2): 309-329.

[31] Chen G. Solar: Building a context fusion network for pervasive computing [Ph.D. Thesis]. Dartmouth College, Hanover, New Hampshire, 2004.

[32] Wischweh J, Bade D. Activity-oriented context adaptation in mobile applications. In *Proc. the 8th International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, Dec. 2011, pp.298-313.

[33] Rehman K, Stajano F, Coulouris G. An architecture for interactive context-aware applications. *IEEE Pervasive Computing*, 2007, 6(1): 73-80.

[34] Julien C, Roman G C. EgoSpaces: Facilitating rapid development of context-aware mobile applications. *IEEE Transactions on Software Engineering* 2006, 32(5): 281-298.

[35] Sindico A, Grassi V. Model driven development of context aware software systems. In *Proc. International Workshop on Context-Oriented Programming*, July 2009, Article No. 7.

[36] Kelly S, Pohjonen R. Worst practices for domain-specific modeling. *IEEE Software*, 2009, 26(4): 22-29.

[37] Sheng Q Z, Benatallah B. ContextUML: A UML-based modeling language for model-driven development of context-aware Web services. In *Proc. the 4th International Conference on Mobile Business*, July 2005, pp.206-212.

[38] Ceri S, Daniel F, Matera M, Facca F M. Model-driven development of context-aware Web applications. *ACM Transactions on Internet Technology*, 2007, 7(1): Article No. 2.

[39] Ayed D, Delanote D, Berbers Y. MDD approach for the development of context-aware applications. In *Proc. the 6th International and Interdisciplinary Conference on Modeling and Using Context*, August 2007, pp.15-28.

[40] Henricksen K, Indulska J. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2006, 2(1): 37-64.

[41] Barnett S, Vasa R, Grundy J. Bootstrapping mobile app development. In *Proc. the 37th International Conference on Software Engineering, Volume 2*, May 2015, pp.657-660.

[42] Schreiber F A, Camplani R, Fortunato M, Marelli M, Rota G. PerLa: A language and middleware architecture for data management and integration in pervasive information systems. *IEEE Trans. Software Engineering*, 2012, 38(2): 478-496.

[43] Alvares F, Rutten E, Seinturier L. A domain-specific language for the control of self-adaptive component-based architecture. *Journal of Systems and Software*, 2017, 130: 94-112.

[44] Kulkarni D, Ahmed T, Tripathi A. A generative programming framework for context-aware CSCW applications. *ACM Trans. Software Engineering and Methodology*, 2012, 21(2): Article No. 11.

[45] Lee Y, Iyengar S, Min C, Ju Y, Kang S, Park T, Lee J, Rhee Y, Song J. Mobicon: A mobile context-monitoring platform. *Communications of the ACM*, 2012, 55(3): 54-65.

Xuan-Song Li received his B.Sc. and Ph.D. degrees in computer science from Nanjing University, Nanjing, in 2007 and 2016, respectively. He is currently an assistant professor in the School of Computer Science and Engineering at Nanjing University of Science and Technology, Nanjing. His research interests include software methodology, pervasive computing, and formal methods. He is a member of CCF.

Xian-Ping Tao received his M.Sc. and Ph.D. degrees in computer science from Nanjing University, Nanjing, in 1994 and 2001, respectively. He is currently a professor in the Department of Computer Science, and State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing. His research interests include software agents, middleware systems, Internetware methodology, and pervasive computing. He is a senior member of CCF and a member of IEEE.

Wei Song received his Ph.D. degree in computer science from Nanjing University, Nanjing, in 2010. He is currently an associate professor in the School of Computer Science and Engineering at Nanjing University of Science and Technology, Nanjing, and was a visiting scholar at Technische Universität München, Germany. His research interests include software engineering, program analysis, services computing, and process mining. He was invited to the Schloss Dagstuhl Seminar "Integrating Process-Oriented and Event-Based Systems" held in August, 2016. He is a senior member of CCF and a member of IEEE.

Kai Dong received his Ph.D. degree in computer science from Nanjing University, Nanjing, in 2014. He is currently an associate professor in the School of Computer Science and Engineering at Southeast University, Nanjing. His research interests include security, privacy, localization, and social networks.