COPAL: An Adaptive Approach to Context Provisioning

Fei Li, Sanjin Sehic, Schahram Dustdar
Distributed Systems Group (DSG)
Information Systems Institute
Vienna University of Technology, Austria
A-1040 Wien, Argentinierstrasse 8/184-1
Email: {li, ssehic, dustdar}@infosys.tuwien.ac.at

***Abstract*—Context-aware services need to acquire context information from heterogeneous context sources. The diversity of service requirements posts challenges on context provisioning systems as well as their programming models.**

This paper proposes COPAL (COntext Provisioning for ALl) — an adaptive approach to context provisioning. COPAL is at first a runtime middleware, which provides loose-coupling between context and its processing. The component architecture of COPAL ensures that new context processing functions can be added dynamically. A set of context processing patterns are proposed to customize context attributes and compose context provisioning schemes. The COPAL components and models are reflected in a Domain Specific Language (DSL), which can further reduce the development efforts of context provisioning using automatic code generation. A motivating scenario is used throughout the paper to illustrate COPAL approach.

I. Introduction

Context-awareness is one of the cornerstones of mobile and ubiquitous computing [1][2]. One major problem in providing context-aware services is how to bridge the vast information gap between context sources and context-aware services. Context information is generated by heterogeneous and lower-level devices, which are unaware of application requirements and information models. Services need to understand complex context information to achieve context-awareness. *Context provisioning* refers to the approach of gathering, transferring and processing context in order to raise context-awareness of ubiquitous services.

In ubiquitous environments, new devices are incrementally added to meet new requirements. Thus context provisioning should integrate new context sources and customize context information. The same context could come from different sources, with different source properties and information quality, and some of these properties are not explicitly provided by sources. Different services could have different requirements for the same type of context. Context-aware services could be interested in aggregative results of more than one type of context for a certain time span. All these diversities of service requirements post significant challenges not only on the architecture of context provisioning systems, but also on the programming model of context provisioning, which has rarely been addressed in the literature.

This paper presents COPAL (COntext Provisioning for ALl)[1]. COPAL is intended to provide a runtime middleware as well as a new programming model for context provisioning. COPAL features a loosely-coupled and modularized architecture for integrating new context sources, creating new information models and supporting various information processing requirements of context-aware services. Essential context attributes are supported but services can enrich the attributes and add corresponding processing actions. Five context processing patterns are summarized to assist the design of context provisioning schemes. Based on the COPAL architecture and context provisioning models, a Domain-Specific Language (DSL) [3] is proposed to facilitate the development of context provisioning schemes. COPAL-DSL has a concise, text-based grammar to define component and deployment models. Code skeletons for COPAL components and deployment artifacts can be generated from the DSL automatically.

The paper is structured as follows: Section 2 describes a scenario to motivate the COPAL approach. The COPAL framework and key concepts are introduced in Section 3. Section 4 proposes context attributes in COPAL and the context processing patterns to construct context provisioning plans. Section 5 presents COPAL-DSL to facilitate the development of context provisioning plans. The related work is surveyed and compared in Section 6, and Section 7 concludes the paper with future work.

II. Motivating scenario

Frida lives in a smart home equipped with sensors, control devices and a context-aware service platform. A basic requirement of context-aware services is to keep a comfortable ambience for Frida. This requirement needs environmental *luminance* and *temperature* to decide the suitable lighting and the setting of air conditioner respectively. In addition, Frida's presence in certain area is essential for managing the status of devices.

Furthermore, the town where Frida lives is involved in an experimental deployment of Smart Meters and Smart Grids [4], which implement a brand new pattern of power service and encourages residential energy saving. Some smart

[1]COPAL development information and tutorial are available at: http://www.infosys.tuwien.ac.at/m2projects/sm4all/copal/

meters are deployed in Frida's house to monitor and control lights and air conditioner, which are two of the major sources of residential power consumption [5]. A price indicator is connected to smart grid to get real-time price information from power market [6]. Frida has a context-aware service platform and she wants to reduce her power consumption while keeping her home environment comfortable. A context-aware Cozy&Green (C&G) service will be designed to meet her requirements.

From the perspective of context provisioning, C&G should know where luminance and temperature are measured. Frida's *presence* is detected by RFID (Radio Frequency Identification), which might give false-positive readings occasionally. The smart meters and price indicator provide two types of context respectively: the real-time *power consumption* and real-time *power price*. The smart meters are not very precise, thus the quality of information has to be considered before using it. The price information is reliable and straightforward. For scheduling energy usage of her home, total power consumption and total power cost are also of interest to Frida.

This scenario is meant to illustrate the requirements to flexible context provisioning but not to be over-complicated. The example C&G service should exploit multiple context types provided by many sources—luminance sensor, thermometer, presence sensors and smart meters and price indicators. The preliminary sensor readings has to be processed (e.g., to evaluate the quality of information) before being used by C&G. Some indirect context information (e.g., total price of consumed power) has to be provided. The goal of context provisioning is to provide sufficient information for services to make decisions and adapt itself to the changing context, thus we leave the logic inside C&G service to Frida, e.g. when to activate air conditioner and what is the target temperature. This paper will illustrate the architectural elements of COPAL and our approach to context provisioning.

III. The COPAL framework

COPAL is situated in the overall architecture of smart home middleware developed in SM4ALL (Smart hoMes for ALL) project[2]. The important system services, COPAL components and general process of context provisioning are illustrated in Fig. 1[3] (on the next page). The system parts outside COPAL are simplified for the sake of clarity.

1) Device services: Device services refer to devices and sensors in the environment. The heterogeneity of hardware and communication protocols is hidden by device-specific *Wrappers (W)*, which implement access protocols and expose devices as web services to upper-layer. The service descriptions comply to UPnP (Universal Plug and Play[4]) standard, which is extensible, and has already provided a broad spectrum of service descriptions. The sensors that have very simple functions and limited resources are not able to act as wrappers. They are connected to and accessed via wrappers that implement their communication protocols. *Device manager* stores hardware information, maintains a device catalog, and monitors device status. From COPAL point of view, the device services serve as context **Publishers**.

2) Context-aware (CA) services: Context provisioning provides interested context to CA services. From COPAL point of view, CA services are merely context **Listeners** which COPAL has to notify when corresponding queries are met. Each service can have multiple listeners to collect context of different types, from different sources, or with different criteria.

3) COPAL: The relationship between COPAL components are illustrated in Fig. 2. It is worth noting that understanding the relationship between the components is required for developing context provisioning schemes using COPAL-DSL.

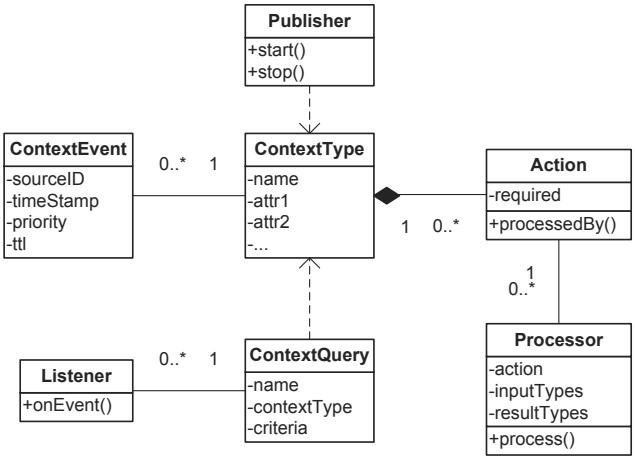

Fig. 2. COPAL components

Context Type (CT) is central in COPAL components. A context type has a unique name and a set of attributes. A publisher can only publish a certain type of context. If the type is unknown to COPAL it will be registered to *Context Type Registry*. Publishers register themselves to *Publisher Registry*. A publisher can leave and join COPAL anytime, but when it leaves COPAL, the registered context type will be kept in context type registry. Thus, context types are incrementally added to COPAL.

Context Query (CQ) is used for continuously selecting **Context Events**. A query is identified by its name, and the query statement contains a context type and a set of criteria. Each listener is associated with a context query, which can be reused by multiple listeners. If there is no query suitable for the service, it can create a new query using *Query Factory*.

Context Processor (CP) is the key concept by which a wide range of context operations are carried out. Each processor holds a name of the action that it is able to perform. A set of input context types and a set of result context types define the function of a processor. Correspondingly, each context type has a set of default **Actions** that add, modify or remove some of its attributes. Each action can be done by one or more processors,

[2]http://www.sm4all-project.eu

[3]In this section, we denote system *services* in *italic* and COPAL **components** in **bold**.

[4]http://www.upnp.org/

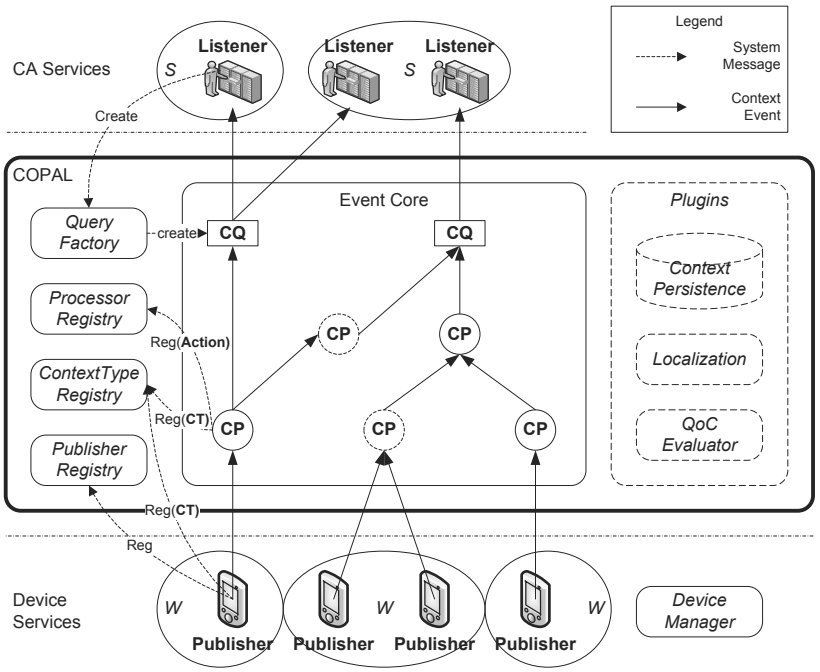

Fig. 1. COPAL Architecture

which register themselves to *Processor Registry*. Processors and context are only coupled by action name dynamically at runtime. Actions can be optional or required. The required actions must be performed by at least one processor, otherwise an exception is thrown. Events are self-contained, that means each event provides sufficient information for COPAL to process it properly. The actions are represented as a list in the event and are processed in order. The context exchange between publishers, listeners and processors is supported by *Event Core*. The event core service is implemented on top of Esper[5] — a widely used open source event processing system. Events in COPAL are XML documents that conform to context types defined in XML schema. COPAL queries will eventually be transformed to EPL (Event Processing Language), which is an SQL-like query language provided by Esper. The high performance of Esper[6] ensures the efficiency of COPAL at run-time.

*Plugin*s are optional services to be used in the context provisioning process. They are not required by the core COPAL architecture but they can extend the functions of COPAL. For instance, to add attributes of source location and QoC (Quality of Context)[7], *Localization* and *QoC evaluator* are developed. Beside continuous queries, COPAL also provides *Context persistence* plugin to support query of historical data. COPAL architecture is based on OSGi[7] framework to achieve modularization and dynamic deployment of components and plugins.

[5]http://esper.codehaus.org
[6]http://esper.codehaus.org/esper/performance/performance.html
[7]http://www.osgi.org

IV. Context processing

Context in COPAL is delivered by events. This section introduces the event attributes applied to context, and the patterns of COPAL context processing to build adaptive and customizable context provisioning processes.

A. Context attributes

1) Required attributes: The four essential context attributes are: **SourceID, Timestamp, Priority and Time To Live (TTL)**. SourceID and Timestamp are inserted by context publishers when an event is created. Default Priority and TTL are decided by the context type of the event.

- **Priority**: When processing a context event, the processor is aware of its significance from its Priority. Priority can assure emergency events as security related information to be processed first.
- **TTL**: TTL is strongly related to event type and the specific source that provides the event. For example, the location of car might be valid for less than 3 seconds, while the temperature of living room could be valid for half an hour. COPAL will discard any context event that is out of TTL.

2) Optional attributes: COPAL provides three optional attributes to all context types: **SourceLocation, QoC (Quality of Context)**, and **Authorization**. The processing of these attributes is not in COPAL core functions, but supported by processors and plugins.

SourceLocation: Location is a well-defined concept in the literature. As an attribute of context event, it denotes where the event is generated. For the complexity and various choices

of localization [8], we provide two options to present Source-Location attributes. One is basic coordinates. The difficulty to apply this approach is that the accuracy of coordinates is largely limited by location detection technologies, especially for indoor mobile context sources. Another is conceptual location with concepts like ”living room”, ”car”, etc. Conceptual location is easier to detect with higher confidence due to the coarse resolution of information. Conceptual location is sufficient for most applications in smart home environments. However, it is worth noting that location detection is not in the scope of this paper.

QoC: QoC is defined as ”any inherent information that describes context information and can be used to determine the worthiness of information for a specific application”[7]. Several QoC metrics have been proposed by Buchholz, T et al. [9] and Krause, M et al. [7]. COPAL supports the most important metrics by using the QoC evaluation approach developed by Manzoor, A et al. [10].

- Freshness: Freshness (also known as ”Up-to-dateness”) is an important attribute to capture the merit of a context event. It is evaluated using TTL and timestamp of a context event. The requirements for freshness are highly dependent on context-aware applications.
- Trust-worthiness: This quality measure indicates the belief in the correctness of information in a context event.
- Precision: ”Precision describes how exactly the provided context information mirrors the reality” [9]. Precision is decided by the capability of context source itself.

Authorization: Context information is related to user environments or user activities, so the access to some sensitive context should be limited to certain authorized services. Authorization defines which service is entitled to access a certain type of context. Services that can access a context event are enumerated by this attribute. The authorization attributes can be further combined with other properties to achieve flexible access control, as changing access with time, location and other criteria.

B. Processing patterns

Processing patterns define the abstract relationships between input events and output events of a processor in COPAL. Understanding the processing pattern can help design and compose processors for constructing complex context provisioning schemes. Five patterns are summarized in this section, namely **Filter**, **Abstraction**, **Differentiation**, **Enrichment** and **Peeling**. These patterns are inspired by the work in complex event processing [11] and event processing networks [12], but they are adapted and elaborated specifically for context provisioning.

- **Filter** Filter excludes the ”unqualified” context information from further processing in COPAL. It is a basic pattern that every processor applies on TTL attribute of each event. More complex combinations of criteria can be applied on context types, values and attributes, e.g., $temperature > 10°C, location = livingRoom$.

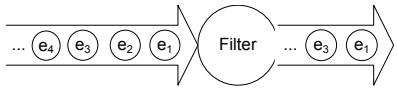

Fig. 3. Filter

- **Abstraction** Abstraction creates events to indicate the context which can only be derived from the occurrence or missing of more than one event.

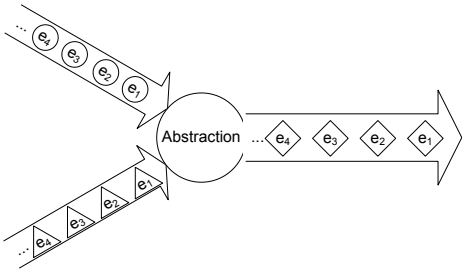

Fig. 4. Abstraction

Abstraction can be further classified to **Summarization** and **Aggregation**. Summarization gathers same type of events that have occurred during a certain time span and publishes a summarizing event, e.g. the power usage of a fridge for each hour. Aggregation gathers different types of events and publishes a new type of context, e.g. ”door open” and ”lights on” together can indicate the presence of person in a certain room. The hybrid of summarization and abstraction can help build more sophisticated and efficient context processing schemes. For example, in our smart home environment, we apply several sensors to monitor physical status of users. Heartbeat frequency, blood pressure and breath frequency are periodically collected by COPAL. Most of the time, user is stable and healthy, so it is unnecessary to invoke any other services or take any further action. COPAL uses an abstraction processor to gather all the latest physical status, and periodically publish only one type of Context—”UserHealthy”.
When applying abstraction pattern, the attributes of output context should be set properly according to various service requirements. Table I summarizes the aggregation of attributes for the health monitoring example.

TABLE I
ATTRIBUTES AGGREGATION

Attribute	Aggregation
Priority	The highest of input events
TTL	The shortest of input events
Location	Delete if any
QoC	The lowest of input events

- **Differentiation** Differentiation is applied when services have different processing requirements for one type of context. Using differentiation, same context can be forked

to multiple processors and eventually feed different services.

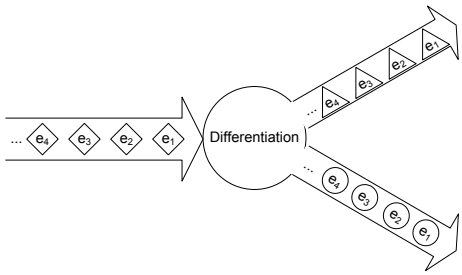

Fig. 5. Differentiation

In our smart home environment, user presence is essential for many services. Security and common environmental control (e.g., Cozy&Green service) both need presence information, but with largely different requirements of quality and frequency. For Cozy&Green the resolution at room level is enough, but for emergency monitoring, the highest resolution and frequency that can be achieved by the devices are always favorable. Therefore, the original user presence information is differentiated into two types: ”RoomOfUser” which only reports when user moves to another room, and ”DetailedLocation” including the original coordinates, user moving direction and room ID. The ”DetailedLocation” is further protected with authorization attribute.

- **Enrichment** Enrichment adds additional information into original events provided by context sources.

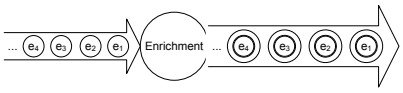

Fig. 6. Enrichment

The enrichments of Location, QoC or Authorization can be applied as the first step of processing. The enrichment is composable, which means two or three types of enrichment can be added to one context event according to service requirements. In certain cases dependencies exist between enriched attributes. For example, if QoC evaluation is based on location, then location attributes are enriched before it. Beside these basic attributes, various types of context may add their own attributes to convey information.

- **Peeling** When the context information has been properly processed and is to be delivered to services, some attributes of context may not be required anymore. A processor can peel off the attributes and only relay the useful information to services.

C. Solution to Cozy&Green service

With the knowledge of context attributes and processing patterns, the context provisioning scheme of C&G is designed as illustrated in Fig. 8 (on the next page).

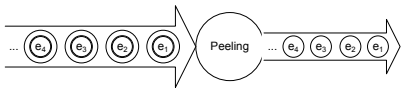

Fig. 7. Peeling

A user could be moving constantly, so the *Presence* of user will be evaluated by a QoC processor to indicate the confidence and freshness of information. *Temperature* information from different places have the same context type. Thus, a localization processor is used for adding location information to each event, and for *Luminance* likewise. The three types of context events are selected by Q1, Q2 and Q3 respectively.

We have multiple smart meters that provide *Power* context measured in Watt-hours (Wh) for lights and air conditioner. The quality of this context type has to be evaluated by meterQoC to provide C&G assistance to decide on usefulness of the information. C&G selects the readings of these meters by Q4 to know current energy consumption of each device.

These readings are also the input for powerSum processor to get a total energy consumption for a certain period. The output of powerSum processor is again Power but for an extended time period. Frida is only interested when the total power consumption for last hour exceeds a certain threshold, so C&G listens to the output of powerSum when it reaches a certain threshold (Q5).

Frida is also interested in a total cost of power consumption for last hour, therefore C&G provides another processor priceSum that calculates the total cost. The input of priceSum processor is total power consumption from powerSum and current energy price. Every price change is notified to C&G by Q7 for analyzing the price fluctuation patterns. Finally, C&G listens on results of priceSum processor only when it exceeds the threshold defined in Q6.

V. COPAL-DSL

”Domain-specific languages (DSLs) are languages tailored to a specific application domain. They offer substantial gains in expressiveness and ease of use compared with general-purpose programming languages in their domain of application” [3]. For these advantages, COPAL provides a DSL to further facilitate the development of application-specific context provisioning schemes.

COPAL-DSL, on one hand, is a description of COPAL components and their relationships as defined in Fig. 2. On the other hand, it hides COPAL system services to help developers focus on context provisioning logics. The models defined by COPAL-DSL can be used to generate code skeletons and deployment artifacts in order to minimize implementation efforts and maximize automation of context-aware service development. This section goes through COPAL-DSL grammar and code generation by developing the context provisioning process of C&G service described in Fig. 8.

COPAL-DSL has a concise, text-based grammar. The DSL encompasses *Component Models* and *Deployment Models*. Each model belongs to a model type, e.g., Listener, Manifest,

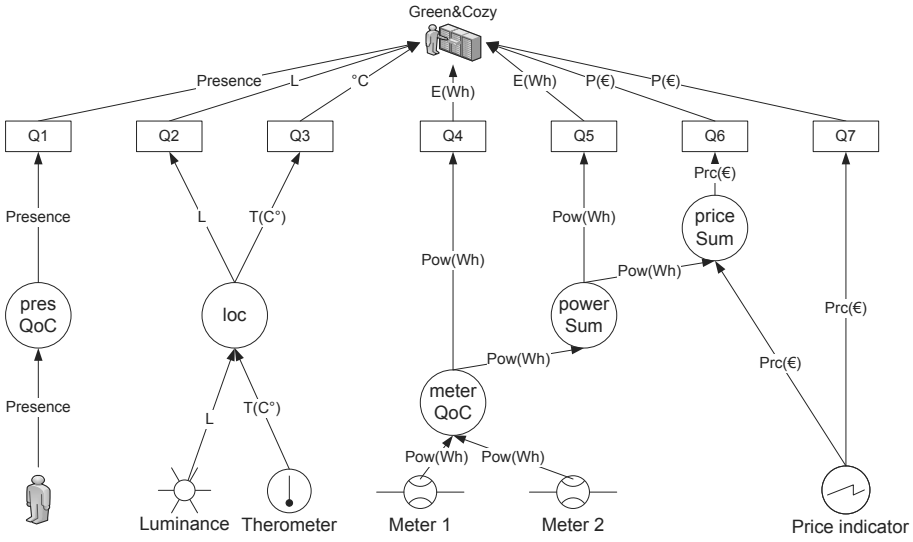

Fig. 8. COPAL solution for Cozy&Green Service

(in **bold** in following code snippets) and is identified by a unique name. Inside each model is a set of required and optional properties (in *italic* in following code snippets). COPAL-DSL grammar is defined in Xtext[8]. Correspondingly, code generator is implemented by Xpand[9]. These two tools are based on EMF (Eclipse Modeling Framework)[10].

A. Component model

- **Publisher and ContextType** Each publisher has to refer to one and only one context type when implementing the publisher class. Each context type is defined by a set of attributes. The common attributes introduced in previous section are implicitly injected to events by COPAL, so they will not be specified in model definition. In the following code, Meter1 publisher publishes Power. This Power type has three attributes. For example, in ”metric:String!”, ”metric” is the name of attribute, ”String” is the datatype, and ”!” indicates that it is a required attribute. The *defaultAction* specifies an action that can be applied on this context type. ”!” indicates that ”meterQoC” is a required action. The code for other context types is skipped to save space.

```
Publisher Meter1{
  contextType=Power
}
```

```
ContextType Power{
  attribute=metric:String!
  attribute=value:Float!
  attribute=duration:Int
  defaultAction=meterQoC!
  defaultAction=powerSum
  defaultAction=priceSum
}
```

[8]http://www.eclipse.org/Xtext/
[9]http://wiki.eclipse.org/Xpand
[10]http://www.eclipse.org/modeling/emf/

- **Listener and ContextQuery** Listener component specifies in which events it is interested using queries. A query is defined by a context type and the logical criteria on attributes. A listener is associated with a query when it registers itself to COPAL. In the code snippets below, query Q5 selects the event notifying if the consumption is greater than 1000 Wh.

```
Listener listener5{
  query=q5
}
```

```
ContextQuery q5{
  contextType=Power
  criteria="sourceID='powerSum' and
        unit='Wh' and value>1000"
}
```

- **Processor** Each processor handles a certain action, which is indicated by the *action* field in processor definition. The input and output events are defined by *inputTypes* and *resultTypes* respectively. The code snippet below defines the priceSum processor. Power and Price are input for priceSum processor and its output is Price. priceSum processor uses previously introduced Abstraction pattern.

```
Processor priceSum{
  action=priceSum
  inputTypes=[Power Price]
  resultTypes=[Price]
}
```

Fig. 9 (on the next page) illustrates the component model elements and generated artifacts. Context type definition will generate an XML Schema containing the attributes and a configuration file that specifies the default actions. Each publisher, processor and listener definition generates separate code skeletons that require their specific methods (represented **bold** in the figure) to be implemented by developer.

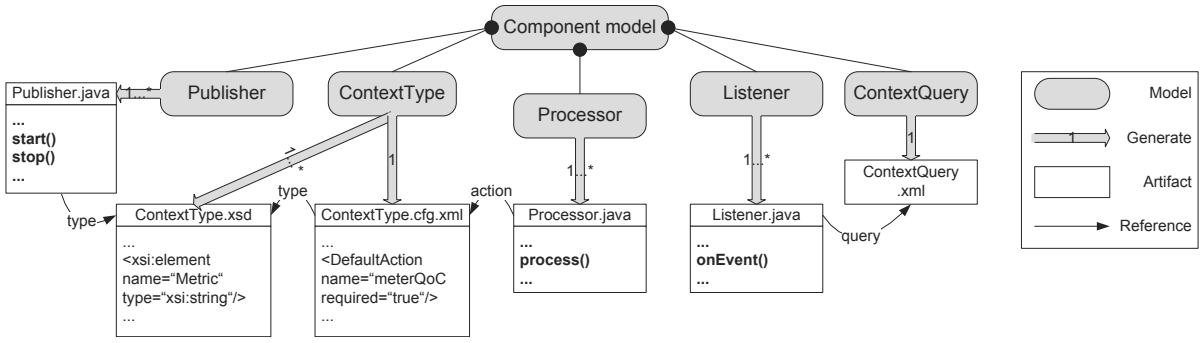

Fig. 9. DSL component model and artifacts

B. Deployment model

The deployment model specifies how the artifacts generated from component model are deployed as an OSGi bundle. The general bundle information and the main activator name are described in manifest model. Artifacts model enumerates the *publishers*, *listeners* and *processors* to be deployed.

```
Manifest CGManifest {
  BundleName='sm4all.copal.cozygreen'
  BundleSymbolicName='copal-cozygreen'
  BundleVersion='0.1'
  Activator='CGActivator'
}
```

```
Artifacts CGComponents {
  publishers=[Meter1 Meter2 PriceIndicator]
  listeners=[listener1 listener2
             listener3 listener4
             listener5 listener6 listener7]
  processors=[meterQoC powerSum priceSum]
}
```

Fig. 10 illustrates the generation of deployment artifacts. The generation of OSGi activators is based on the components deployed in a bundle. Publishers, processors and listeners are registered in corresponding activators. A main activator, named in manifest model, activates and deactivates other activators. The generated deployment artifacts require no further implementation efforts.

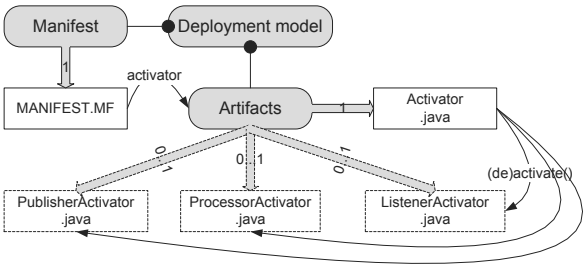

Fig. 10. DSL deployment model and artifacts

VI. Related Work

Context provisioning has been intensively investigated in the past years, but COPAL approach distinguishes itself substantially.

To the best of our knowledge, there has not been an effort to define a DSL for context provisioning in the literature. We offer service engineers a DSL with source code generation capability to facilitate development of context provisioning schemes. COPAL-DSL is an abstraction of COPAL architecture and COPAL context event model. With concise text-based grammar, COPAL-DSL helps engineers to define and utilize context in a very intuitive way. One complementary work to COPAL-DSL is *Habitation* DSL [13]. Habitation is a result of applying Model-Driven Engineering (MDE) in home automation domain. Its focus is on controlling home devices. The graphical expression of Habitation is appealing, but it does not support automatic code generation. Conceptually, COPAL resembles the service development approach proposed in SPICE (Service Platform for Innovative Communication Environment) project [14][15]. The SPICE service development approach employs MDE concepts. A domain-specific UML dialect called SPATEL is developed to provide abstractions of mobile service interfaces. Code generation to different service execution environments is supported. But SPATEL stresses on mobile service description rather than context utilization.

Chen et al. [16][17][18] proposed a context sharing middleware—Solar, which is a P2P network based on application layer multicasting and Distributed Hash Table (DHT). Distributed operators are proposed as the abstract basic units of context data processing in Solar, and Filter-Pipe is the core pattern to compose peers. COPAL enriches the idea of Solar in two respects. At first, COPAL extends the programming model to five context processing patterns in order to accommodate complex application requirements and context attributes. Second, COPAL decouples context processing action and processor implementation, which offers more flexibility when developing context-aware systems because third-party processors can be utilized in COPAL easily. In addition, COPAL is intended to hide the lower-level hardware information from application developers. In contrast, Solar is directly implemented on sensors.

Conan et al. [19] and Taherkordi et al. [20] proposed to use composable components for context processing. We recognize the significance of composability, but our approach to composition is largely different. COPAL uses context type to couple processors, and event processing to manipulate context, thus

COPAL is able to maintain and operate on structural context information with composition of processors.

Broker-based architecture [21][22][23] is popular in distributed context-aware systems. In the sense of a context provisioning system situated between context providers and context-aware services, COPAL is also a broker. Knappmeyer et al. [23] proposed a broker-based context provisioning system that supports context presentation, publish/subscribe and data acquisition from devices. The configuration and management of context is supported by Context Meta Language (ContextML). COPAL offers more adaptive and customizable context data processing approach rather than publish/subscribe to bridging context providers and services. Furthermore, COPAL-DSL is tailored to be more expressive and concise than XML-based ContextML.

VII. Conclusions and future work

This paper introduced COPAL (COntext Provisioning for ALl) framework, which provides a runtime middleware as well as a programming approach to context provisioning. The COPAL architecture is designed following the principle of loose-coupling and modularization. New context provisioning schemes can be deployed to COPAL dynamically. Important context attributes as QoC and source location are analyzed and incorporated into COPAL. The context processors can compose with each other via input and output events. The abstract functions of processors are summarized to five processing patterns. The usage of context attributes and processing patterns is demonstrated by designing Cozy&Green service in our scenario. The context provisioning scheme for C&G service is further specified by COPAL-DSL, which largely reduces the development efforts because of its concise grammar and code generation capability.

COPAL is our first attempt to a new programming model of context-aware systems. We plan to further improve COPAL in several respects. First, we will extend COPAL framework to distributed architecture that can be deployed on multiple hosts and share context events among them. A context provisioning plan can be executed by the cooperation of multiple COPAL nodes. Correspondingly, the COPAL-DSL will be updated to include distributed deployment model. Second, context provisioning plans that are provided by developers need to be tested and validated before deployment. Further more, we will investigate graphical DSL and eventually offer a development environment to accommodate COPAL programming model.

Acknowledgment

This work is supported by EU FP7 STREP Project SM4ALL (Smart hoMes for ALL), under Grant No. 224332.

References

[1] M. Baldauf, S. Dustdar, and F. Rosenberg, “A survey on context-aware systems,” *International Journal of Ad-Hoc and Ubiquitous Computing*, Jan 2006.

[2] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, “Towards a better understanding of context and context-awareness,” in *HUC ’99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. London, UK: Springer-Verlag, 1999, pp. 304–307.

[3] M. Mernik, J. Heering, and A. M. Sloane, “When and how to develop domain-specific languages,” *ACM Comput. Surv.*, vol. 37, no. 4, pp. 316–344, 2005.

[4] M. Chebbo, “EU SmartGrids framework ”Electricity networks of the future 2020 and beyond”,” in *2007 IEEE Power Engineering Society General Meeting*, Tampa, FL, USA, 2007, pp. 1–8.

[5] Residential Energy Efficiency—DNR, http://www.dnr.mo.gov/energy/residential/residential.htm.

[6] C. Wang, M. de Groot, and P. Marendy, “A Service-Oriented system for optimizing residential energy use,” in *Web Services, 2009. ICWS 2009. IEEE International Conference on*, 2009, pp. 735–742.

[7] M. Krause and I. Hochstatter, “Challenges in modelling and using quality of context (qoc),” *Mobility Aware Technologies and Applications*, pp. 324–333, 2005.

[8] V. Zeimpekis, G. Giaglis, and G. Lekakos, “A taxonomy of indoor and outdoor positioning techniques for mobile location services,” *ACM SIGecom Exchanges*, vol. 3, no. 4, pp. 19–27, 2002.

[9] T. Buchholz, A. Kupper, and M. Schiffers, “Quality of context: What it is and why we need it,” in *Proceedings of the workshop of the HP OpenView University Association*, 2003.

[10] A. Manzoor, H. Truong, and S. Dustdar, “On the Evaluation of Quality of Context,” in *Proceedings of the 3rd European Conference on Smart Sensing and Context*. Springer-Verlag Berlin, Heidelberg, 2008, pp. 140–153.

[11] D. Luckham, “The power of events: an introduction to complex event processing in distributed enterprise systems,” *Rule Representation, Interchange and Reasoning on the Web*, pp. 3–3, 2008.

[12] G. Sharon and O. Etzion, “Event-processing network model and implementation,” *IBM Syst. J.*, vol. 47, no. 2, pp. 321–334, 2008.

[13] M. Jimenez, F. Rosique, P. Sanchez, B. Alvarez, and A. Iborra, “Habitation: A Domain-Specific language for home automation,” *IEEE Software.*, vol. 26, no. 4, pp. 30–38, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1592157

[14] O. Droegehorn, I. Konig, G. Le-Jeune, J. Cupillard, M. Belaunde, and E. Kovacs, “Professional and end-user-driven service creation in the SPICE platform,” in *World of Wireless, Mobile and Multimedia Networks, 2008. WoWMoM 2008. 2008 International Symposium on a*, 2008, pp. 1–8.

[15] M. Belaunde and P. Falcarin, “Realizing an MDA and SOA marriage for the development of mobile services,” in *Model Driven Architecture Foundations and Applications*, 2010, pp. 393–405.

[16] G. Chen and D. Kotz, “Context aggregation and dissemination in ubiquitous computing systems,” in *Mobile Computing Systems and Applications, 2002. Proceedings Fourth IEEE Workshop on*, 2002, pp. 105–114.

[17] G. Chen, M. Li, and D. Kotz, “Design and implementation of a large-scale context fusion network,” in *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (Mobiquitous)*, 2004, pp. 246–255.

[18] Chen, M. Li, and D. Kotz, “Data-centric middleware for context-aware pervasive computing,” *Pervasive and Mobile Computing*, vol. 4, no. 2, pp. 216–253, 2008.

[19] D. Conan, R. Rouvoy, and L. Seinturier, “Scalable processing of context information with COSMOS,” in *Distributed Applications and Interoperable Systems*, 2007, pp. 210–224.

[20] A. Taherkordi, R. Rouvoy, Q. Le-Trung, and F. Eliassen, “A self-adaptive context processing framework for wireless sensor networks,” in *Proceedings of the 3rd international workshop on Middleware for sensor networks*. Leuven, Belgium: ACM, 2008, pp. 7–12.

[21] H. Chen, T. Finin, and A. Joshi, “Semantic web in the context broker architecture,” in *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom’04)*. IEEE Computer Society, 2004, p. 277. [Online]. Available: http://portal.acm.org/citation.cfm?id=978667

[22] M. van Sinderen, A. van Halteren, M. Wegdam, H. Meeuwissen, and E. Eertink, “Supporting context-aware mobile applications: an infrastructure approach,” *Communications Magazine, IEEE*, vol. 44, no. 9, pp. 96–104, 2006.

[23] M. Knappmeyer, N. Baker, S. Liaquat, and R. Tnjes, “A context provisioning framework to support pervasive and ubiquitous applications,” in *Smart Sensing and Context*, 2009, pp. 93–106.