A Publish/Subscribe-based Programming Language for Sensor Networks

BiaoDong *
School of Computer & Software
Nanjing Institute of Industry Technology
Nanjing, China
dongb@niit.edu.cn
* Corresponding Author

Jinhui Chen
School of Computer & Software
Nanjing University of Information Science & Technology
Nanjing, China
cjh@nuist.edu.cn

Abstract—Sensor networks constitute an efficient solution for gathering data on such events and feeding the evacuation simulation. Sensor networks software(SNS) is considered as a new software paradigm in the sensor networks environment, with the characteristics of cooperation and flexibility. SNS brings challenges to programming technologies in terms of model, language and platform. This paper presented a new programming language, called PSPL, for modeling and implementing the architecture of sensor networks applications using publish/subscribe paradigm. Considering bridging the gap between sensor nodes and enterprise applications where sensor data is used, PSPL was defined by integrates organization abstract and event-condition-action(ECA) technology. PSPL was composed of group, role, and behavior agent. Rule management and matching is designed through two aspects, such as subscription management and rule matching. A runtime environment, namely PSPL_RTE, was developed for development, compile, deployment and running of PSPL programs. Simulation experiments imply that PSPL is simplicity, while ensuring good flexibility and co-operability.

Keywords-programming language; publish/subscribe; sensor networks; model

I. INTRODUCTION

Programming language for sensor networks is a software layer between operating system and various distributed applications. SNS make use of Sensor networks and the related embedded operating system. SNS is a kind of middleware that provides the desired services such as reusable code services, system abstractions, as well as resource services for sensor networks applications. Publish/subscribe is an asynchronous communication paradigm that supports many-to-many interactions between a set of clients. Publish/subscribe protocol aids re-configurability in dynamic environments where clients and their roles can change frequently, and this paradigm is a suitable paradigm for SNS.

Mottola used a taxonomy to provide an exhaustive classification of existing approaches, and mapped existing approaches back to the application requirements, therefore providing useful insights for selecting the programming abstraction most appropriate to the application[1].Hughes described LooCI, LooCI components use an event-based binding model that allows developers to model component interactions, while providing support for run-time reconfiguration, reflection, and policy-based management [2]. Chen described PS-QUASAR, a middleware for wireless sensor and actor networks, offers a high level simple programming model based on the publish/subscribe paradigm. PS-QUASAR also handles QoS and supports a many-to-many exchange of messages between nodes in a fully distributed way by means of multicasting techniques[3]. Gámez presented a family of configurable middleware with a really flexible architecture, instead of building a single version of a middleware with a rigid structure, and presented the architecture of middleware that can be configured, following a software product line approach, in order to be instantiated in a particular device fulfilling specific application requirements[4]. Nicola proposed a set of programming abstractions that permit to represent behaviors, knowledge and aggregations according to specific policies, and to support programming context-awareness, self-awareness and adaptation[5]. Seeger presented an event-driven middleware for on-body and ambient sensor networks that allows multiple applications to define information types of their interest in a publish/subscribe manner[6]. Morales introduced mechanisms that allow mobile brokers to distribute their subscribers and coordinate the notification of events between fixed and mobile brokers, and proposed a hot-topic algorithm that marks the event popularity[7]. Bakillah presented a publish/subscribe system based on event calculus to support real-time multi-agent evacuation simulations. The publish/subscribe system acts as a middleware between the sensor data publishers and the multi-agent evacuation simulation through a sensor processing service that infers the impact of events on the characteristics of the road network[8]. Fortino presented signal processing in node environment, an open-source programming framework, designed to support rapid and flexible prototyping and management of body sensor networks applications[9]. Mottola presented a programming model called team-level programming that can express collaborative sensing tasks without exposing the complexity of managing multiple drones, such as concurrent programming, and scaling[10]. Marques developed the networked vehicles' language for coordinated control of unmanned vehicle networks. A single program expresses an on-the-fly selection of multiple vehicles and their allocation to cooperative tasks, subject to time, precedence, and concurrency constraints[11]. Doroodgar concentrated on Seluge, one of the existing over-the-air programming schemes, and proposed an improved version of it, named Seluge++, which complies with the security model requirements[12].

This paper presents the programming model and system design that tailor PSPL to SNS environment. Specifically, it makes the following primary contributions. Firstly, we propose PSPL based on publish/subscribe, and define its PSPL model, role-based Interaction, and runtime environment. And secondly, we design and implement rule management and matching, such as registration, publishing, subscription management and rule matching.

II. PUBLISH/SUBSCRIBE-BASED PROGRAMMING LANGUAGE

A. PSPL Model

PSPL programming model integrates organization abstract and ECA technology, and group, role, and behavior agent are constructed as first order entities. Behavior agent is the basic running entity of SNS system. Group encapsulates and describes the behavior of the operating context, including the social environment and the physical environment of the behavior. In a specific group, the structure and behavior of SNS are abstracted and described by the roles. Namely, the behavior rules, the interaction patterns, and the accessible resources are described by the roles. By playing a role, SNS gets the executable behaviors and the available resources. That is, a behavior agent holds all the rules, the interaction patterns, and the accessible resources defined by the role. There are a many to many, dynamic relationship between behavior agent and role. During the operation, behavior agent can be in multiple groups, plays multiple roles, and dynamically changes in their life cycle. A role in a group can be played by multiple behavior agents. In this section, we propose a reference framework, shown in Fig. 1, to describe the programming model of PSPL integrated organization and ECA technology.

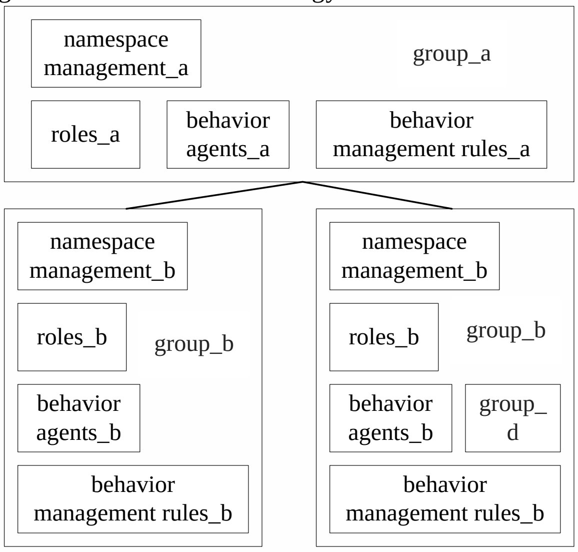

Figure 1. Reference model of PSPL

A SNS system is abstracted as a group, which has a hierarchical structure, and can be nested. A group consists of a set of roles, behavior agents, an optional sub-groups and behavior management rules. A group provides a namespace for the role and behavior agents that it contains. The behavior management rule defines how the organization can dynamically adjust its structure based on the runtime environment. This is primarily achieved by adjusting the number of its members, that is, based on an operating environment, the group dynamically creates new agents, remove idle or error behavior agents.

Role defines the behavior rules, the interaction patterns, and the accessible resources. A role includes a set of behaviors and properties. Behavior is a description of its behavior rules and interaction patterns, runs autonomously, and abides by ECA model. Thus, a behavior is composed of a precondition and one or more actions. An action may be related to a method or function call, message sending or receiving. It can be seen that PSPL actually provides a higher level of abstraction than object technology. So, PSPL can be considered as the extension and evolution of object model.

Agent behavior is the basic operating entity of SNS. In its life cycle, it can dynamically join or leave different groups, dynamically plays different roles or removes role-playing. So SNS can be regarded as a dynamic role set. A birth group of a behavior agent is called an owner group of the behavior agent. Similarly, a behavior agent, which was born in the group, is called a local agent of the group. On the contrary, is known as a foreign agent.

B. Role-based Interaction

In SNS, behavior agent may only care about the services they need without regard to specific service providers. In traditional message passing mechanisms, both object-oriented programming and agent-oriented programming have assumed that they have already known the specific interactive objects. Their cooperative mechanisms are fixed to class or agent templates by means of method call or message transmission. Since the number of the entities in SNS dynamically changes, behavior agent must be able to dynamically discover interactive objects, and establish the interaction between the interactive objects. Although event mechanism provides dynamic interaction mechanism, the mechanism achieves inversion of control. The autonomy of agent behavior is limited, that is, behavior agent can only passively by event driven, and can't take the initiative to request services. PSPL provides role based interaction, and allows agent behavior to dynamically discover the needs of the service provider, and dynamically create interaction. Here, the role name can be seen as a service provided by behavior agent. As role player is constantly changing, this enables dynamic interaction between behavior agents. PSPL provides two kinds of role-based interactive modes. In one-to-one way, a message is transmitted to any one of role-players. In one-to-many way, a message is transmitted to all the players in a role. From the receiver, the role-based interaction also provides role-based access control. In object-oriented programming, methods or properties of an object can only be declared as private or public. For all of the objects, their access rights are the same. However, for the role based interaction, agent behavior can automatically limit the sources of its acceptable message. Thus, agent behavior limits the access to services, and provides a more fine-grained access control mode.

C. PSPL Runtime Environment: PSPL_RTE

In order to support the running of PSPL program, we design and implement a PSPL runtime environment(PSPL_RTE). In two aspects of developing and running, PSPL_RTE supports PSPL programming, debugging, compiling and running.

In the developing layer, PSPL_RTE provides a PSPL editor, debugger, and reusable PSPL library to simplify programming and debugging program. PSPL compiler is one of the PSPL_RTE core modules. Based on JavaCC, the compiler transforms PSPL codes into Java codes that can run on the PSPL operating platform.

In the running layer, considering the basic requirements of behavior agent management, message transmission, rules management and yellow pages service, PSPL_RTE provides a code loader, message distributor, organization manager and rule manager. The code loader is used to achieve dynamic role-playing, that is, behavior agent dynamically loads properties and behavior. It is a core component to implement role transition. The message distributor is used to implement role-based interaction. The realization of this interaction can be divided into two processes. Firstly, the active players in the current role are acquired, then, the communication messages are forwarded according to the request. Organization management is an extension of the behavior agent management system. It enhances the management of the life cycle of groups and roles, modifies the life cycle model of the behavior agent management system, and provides basic services, such as namespace services, to support the management of the entities in PSPL program. Rule manager is used to implement rule-based matching. By abstracting the management and processing strategies that should be originally written into an application, rule manager realizes the separation of data and knowledge. Rule manager includes three functions, such as registration Function, publishing and subscription. Rule is a method of combining event trigger, object oriented technology and event driven environment.

III. RULE MANAGEMENT AND MATCHING

A. Registration and Publishing

Registration module manages behavior agent registration requests. It includes three sub-modules such as registration composition, Qos selection, and semantic analysis. The semantic analysis sub-module resolves registration requests sent by behavior agent providers, and analyses request information sent by behavior agent requestors, it stores the related information into a registry in order to facilitate subsequent calls. The service composition sub-module selects and loads the appropriate discovery and combination algorithm, the candidate services, which are obtained by the algorithm, are combined into a composite service to satisfy the functional requests. The Qos selection sub-module validates the discovered services.

Publishing module completes event publishing. Its function processes are summarized as follows. First, according to the specific SNS standards, behavior agent as provider creates practical events, and establishes event library. Second, the providers need registration in a registry center to ensure the routing can normally navigate to the target registry, and then, upload event library files to registration center while the routing saves a copy of the library files. Finally, the provider publishes events, and launch specific registration requests to the routing. According to the registration information, the routing navigates to an appropriate registration center, and manages the services through the registration center.

B. Subscription Management

In subscription management, subscriptions are organized as a subscription forest according to cover relationship. Every tree in the forest is called a subscription tree. In a subscription tree, each node represents a subscription, and all nodes satisfy the following two constraints. A subscription of parent node is covered with that of child nodes; and between nodes in the same level, there may or may not exist cover relationship.

Because there is a cover relationship between different levels of subscriptions in a subscription tree, the subscription tree has two characteristics in the process of matching events.

- If a node matches an event, then all ancestor nodes of this node must also match the event.
- If a node does not match the event, then all descendants of the node are not matched with this event.

Subscription manager uses the second property to accelerate the matching process. Whether the subscription manager enters a sub-tree rooted at a node to search, it depends on whether the event is successfully matched with the node. A sub-tree takes a node as a root. Whether the subscription manager enters the sub-tree to search, it depends on whether the event is successfully matched with the node.

When a user adds a subscription sub1, subscription manager determines whether there is the same subscription with sub1 in the system. If there is sub1, subscription manager just inserts the new subscriber into a reusable subscription set. Otherwise, sub1 is added to the forest. In other words, in the forest, subscription manager find a node node1 which covers sub1, but all children of node1 don’t cover sub1. The node1 is used as the parent node of sub1.

When a user cancels a subscription sub2, subscription manager deletes sub2 for each subscription tree in the forest. In every subscription tree, if there are nodes which are covered by sub2, the nodes denoted as n2. We delete the sub2’s subscribers from the subscriber sets of all the nodes which constitute sub-trees whose roots are the nodes in n2. After deleting sub2, if a subscriber set of a node is empty, then this node is deleted from the subscription tree and all of its children are linked to their grandparent.

C. Rule Matching

In publish/subscribe, we use rules to describe subscriptions. Rule matching mechanism includes the following parts, such as data updating interface, subscription management, event management, subscription set management and subscription storage management.

The framework is shown in Fig. 2. Subscription manager parses rules into atomic predicates. These predicates can be represented as a set of three tuple (TagName, Operator, Value), each of which corresponds to an index. When the system captures an event, the subscription predicate bit vector management uses the subscription predicate index to locate the position associated with the point in the subscription predicate bit vector table, and calculates whether various atomic predicates are satisfied. And then, the manager determines whether a rule is triggered according to the event index.

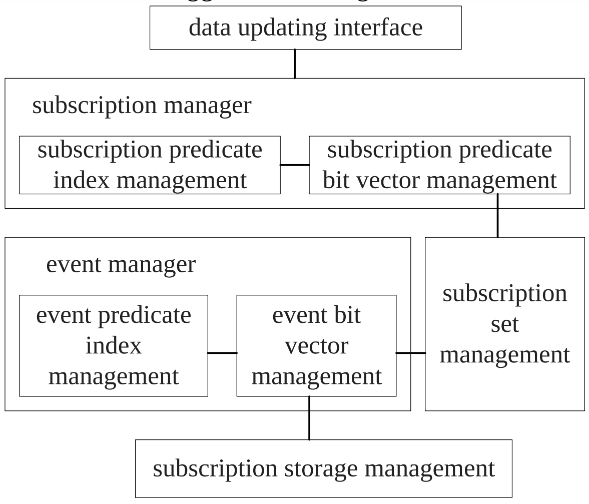
data updating interface
subscription manager
subscription predicate index management
subscription predicate bit vector management
event manager
event predicate index management
event bit vector management
subscription set management
subscription storage management

Figure 2. Framework of rule matching mechanism

Subscription manager parses rules into atomic predicates. These predicates can be represented as a set of three tuple (TagName, Operator, Value), each of which corresponds to an index. When the system captures an event, the subscription predicate bit vector management uses the subscription predicate index to locate the position associated with the point in the subscription predicate bit vector table, and calculates whether various atomic predicates are satisfied. And then, the manager determines whether a rule is triggered according to the event index.

IV. TEST AND ANALYSIS

In order to show how to use this method to solve practical problems, we take a temperature collecting system as an example. The temperature sensor collects temperature data once every one second. And then, the temperature data, together with the sensor node number, are sent to the base station. The parent node forwards the data packet of its child node. When the temperature exceeds a certain threshold, an alert message from the temperature sensor is published. The codes of the system are as follows.

- Code fragments of collecting_org

```
organization temp_collecting_org {
internal role monitor() plays master {
    rule(rulename rule_temp_collector){
        On: TTimer.OnTimer
        Condition: Aowner=app;
        Act: Insert(Temperature);
        behavior_agent{
            TApplication app;TTimer t;TReceive r;
            app=new TApplication(app,0);
            t=new TTimer(app, sys_Initial, 1000,
sys_Infinity);
            r= new TReceiveNew(app);
            while app.Enabled do {
                Application.ProcessMessages;
                app.free;}
            }
        }
external role alert(float temp) plays alerter{
    rule(rulename rule_temp_alerter){
        //role transfer behaviors
        loop receive{
            temp_forwarder? alert(ID, temp): deact alert;}
        }
```

- Code fragments of temp_collecting_org

```
within temp_collecting_org;
employ temp_collecting _org;
organization temp_sensing_org{
internal role sensor(float temp) plays temp_provider{
    rule(rulename rule_temp_provider){
        On: TTimer.OnTimer
        Condition: Aowner=app;
        Act: Insert(Temperature);
    }
internal role forwarder(int parent-ID, int ID, float temp)
plays temp_forwarder{
    rule (rulename rule_temp_forwarder){
        On: TReceive.Get
        Condition: Aowner=app;
        Act: Send(Parent-ID, ID, Temp);}
        // rule_temp_forwarder transfer behaviors
        loop receive{
        Send(parent-ID, ID, temp)?: Alerter g; enact
g.alert(ID, temp); }
    }
}
```

As can be seen from the example, there are two groups, such as temp_collecting_org, and temp_sensing_org. Temperature sensors are responsible for collecting and forwarding temperature data, they are the event publisher. Temperature sensors subscribe the rules, such as rule_temp_provider, and rule_temp_forwarder.

V. CONCLUSIONS

In this paper, we propose PSPL which is a publish/subscribe architecture for SNS. PSPL model is defined in sensor networks environment. We illustrate the rule design through a temperature collecting system. The results show that PSPL program can easily be constructed, and bridge the gap between sensor nodes and SNS applications.

ACKNOWLEDGMENT

This work was sponsored by Qing Lan project 2012 (Jiangsu province, china) and open fund project of Jiangsu provincial research and development center of intelligent sensor network engineering technology, China (ZK13-02-03, Software technology, platform and application for sensor network).

REFERENCES

[1] L. Mottola, and G. P. Picco, “Programming wireless sensor networks: Fundamental concepts and state of the art”, ACM Computing Surveys (CSUR), vol. 43(3), 2011, pp. 19.

[2] D. Hughes, K. Thoelen, W. Horré, et al, “Building wireless sensor network applications with LooCI”, Advancing the Next-Generation of Mobile Computing: Emerging Technologies: Emerging Technologies, vol. 61, 2012.

[3] J. Chen, M. Díaz, B. Rubio, et al, "PS-QUASAR: A publish/subscribe QoS aware middleware for Wireless Sensor and Actor Networks", Journal of Systems and Software, vol. 86(6), 2013, pp. 1650-1662.

[4] N. Gámez, and L. Fuentes, "FamiWare: A family of event-based middleware for ambient intelligence", Personal and Ubiquitous Computing, vol. 15(4), 2011, pp. 329-339.

[5] R. D. Nicola, M. Loreti, R. Pugliese, et a, "A formal approach to autonomic systems programming: The SCEL language:, ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 9(2), 2014, pp. 7.

[6] C. Seeger, K. Van Laerhoven, J. Sauer, et al, "A Publish/Subscribe Middleware for Body and Ambient Sensor Networks that Mediates between Sensors and Applications", Healthcare Informatics (ICHI), IEEE, vol. 2013, pp. 199-208.

[7] A. Morales, T. Robles, R. Alcarria, et al, "A Hot-topic based Distribution and Notification of Events in Pub/Sub Mobile Brokers", Network Protocols and Algorithms, vol. 5(1), 2013, pp. 90-110.

[8] M. Bakillah, A. Zipf, and S. H. L. Liang, "Publish/subscribe system based on event calculus to support real-time multi-agent evacuation simulation", Geographic Information Science at the Heart of Europe, Springer International Publishing, 2013, pp. 337-352.

[9] G. Fortino, R. Giannantonio, R. Gravina, et al, "Enabling effective programming and flexible management of efficient body sensor network applications", Human-Machine Systems, IEEE Transactions, vol. 43(1), 2013, pp. 115-133.

[10] L. Mottola, M. Moretta, K. Whitehouse, et al, "Team-level programming of drone sensor networks", Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems, ACM, 2014, pp. 177-190.

[11] E. R. B. Marques, M. Ribeiro, J. Pinto, et al, "NVL: a coordination language for unmanned vehicle networks", Rendezvous, vol. 28(29):30, 2015.

[12] F. Doroodgar, M. A. Razzaque, I. F. Isnin, "Seluge++: A Secure Over-the-Air Programming Scheme in Wireless Sensor Networks", Sensors, vol. 14(3), 2014, pp. 5004-5040.