# Background

The publish/subscribe (pub/sub) is a communication paradigm which has been well studied in distributed systems. Conceptually, a pub/sub system model can be illustrated as follows:



Figure : Pub/Sub System Model [FACES\_PUBSUB]

As we can see, a pub/sub system has the following entities including subscribers, publishers and brokers. The broker provides event services including data storage and subscription management. Subscribers register their interest in events by using a subscribe() operation to the brokers. A publisher uses a publish() operation to publish events to the brokers. When the published events match any subscription, the broker will use notify() operation to inform all the subscribers involved.

Compared with other communication paradigms such as message passing, RPC [RPC], notification [OBSERVER\_DESIGN], shared spaces [LINDA] and message queuing [MOM], pub/sub has the following three unique decoupling features [FACES\_PUBSUB].

* **Space decoupling**: the interacting parties do not need to know each other. The publishers publish events through an event service and the subscribers get these events indirectly from the event service. The publishers do not usually have any reference to these subscribers; neither do they know how many of these subscribers are participating in the interaction. Similarly, subscribers do usually not have any reference to the publishers; neither do they know how many of these publishers are participating in the interaction.
* **Time decoupling**: the interacting parties do not need to be actively participating in the interaction at the same time. In particular, the publisher might publish some events while the subscriber is disconnected, and conversely, the subscriber might get notified about the occurrence of some event while the original publisher of the event is disconnected.
* **Flow decoupling**: publishers are not blocked while producing events and subscribers can get notified about the occurrence of some event while performing some concurrent activity (through a callback), i.e., subscribers do not need to pull for events in a synchronous manner. In short, message production and consumption do not happen in the main flow of control of the publisher or subscriber.

Because of these features pub/sub has attracted a lot of interest in distributed computing. A few pub/sub systems such as Siena [SIENA], Elvin [ELVIN], Gryphon [GRYPHON] and Jedi [JEDI] have already been implemented in distributed networks.

However, the design and implementation of pub/sub middleware for WSN has not been adequately addressed. There are a few works on pub/sub based systems for WSNs [LOW\_LEVEL\_NAMING] [DIFFUSION\_FILTERS] [MIRES], but those works can only support primitive event, which contains single topic and the filters are mostly threshold-based. However, many applications over WSNs need handle more complex events. For example, the collision between two cars needs to be described by the speeds of two cars and the distance between the cars. Thus, the applications need to use composite events but not just primitive events. A composite event is composed of a set of primitive events which have spatial and temporal relationships among them. However, supporting composite events in a pub/sub middleware is not trivial mainly due to the various constraints of WSN.

# System Design

We develop PSWare [PSWARE], a pub/sub middleware for WSN which addresses the constraints of WSN and is able to support composite events. The overall architecture of our system is shown in the figure below:

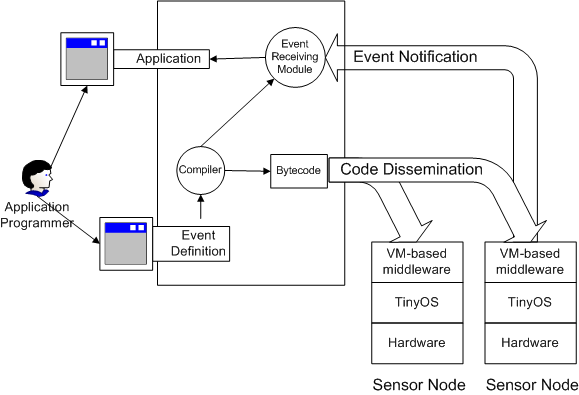


Figure 6: System architecture

The major components in the system include:

1. Event Definition Language (EDL) and its compiler can be used to define composite events and the compiler can translate the high-level event definition into low-level byte codes which can be further executed by individual sensor nodes.
2. VM runtime environment which can be used to execute the byte codes compiled from the event definition. The VM runtime environment can also be used to implement distributed event detection protocols.

Our event definition language is a combination of SQL and object-oriented language. Application programmers start by defining a list of events and a subscription statement. Then, the subscription statement is used to choose the events that need to be delivered. The programming model for EDL is illustrated in the following Figure:

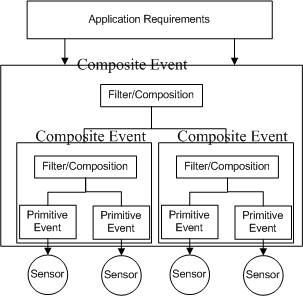


Figure 2: event programming model

The application programmer first defines the events to be subscribed in EDL. The compiler will compile the program into two parts. The first part is the bytecodes which will later be disseminated and executed on individual sensor nodes. The second part is the event receiving module which is a Java program running on the terminal. It will interpret the data sent by the sensor nodes and notify the application programmer about the incoming events. The application programmer will just need to write an application which uses the interface provided by the Event Receiving Module.

The structure of the EDL compiler is shown in the following Figure:

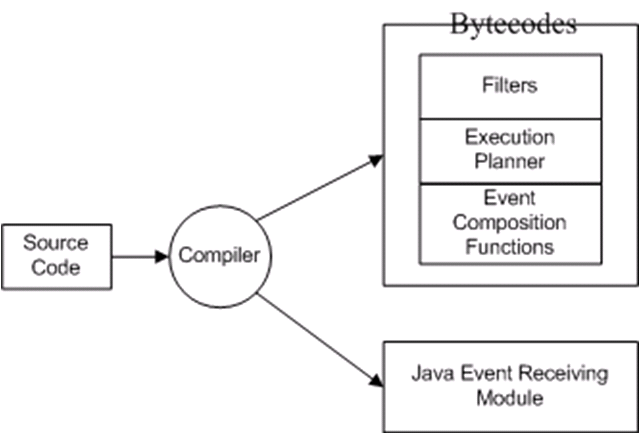


Figure 7: EDL compiler

The bytecodes include some program images where each program image consists of some instructions. In order to help the event detection protocol, the following program images will be produced:

* *Filters* are the instructions generated from the event filters and they are used to let each sensor knows whether it is capable of detecting an event.
* *Main* is the execution planner.
* *Detection codes* are a group of program images which will start execution when event composition takes place.

Our VM-based runtime environment is based on Mate [MATE]. It uses stack-based instructions which allow compact code size. We choose a VM-based runtime environment because it is more flexible in terms of interfacing with the EDL compiler and introducing new modules. The runtime environment is responsible for executing the program image produced by the EDL compiler. Its structure of is shown as follows:

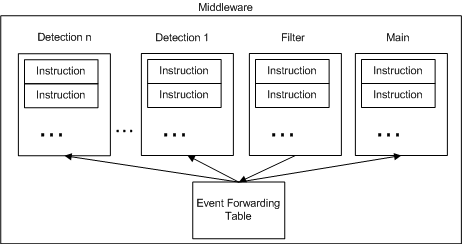


Figure 8: VM-based runtime environment

The VM has several contexts. Each context will be used to execute the corresponding program image produced by the compiler. Upon the detection of an event, the sensor node will execute an instruction called ‘send’ with the event ID as the operand. The ‘send’ instruction will then invoke the routines related to event detection protocol.

# References

[RPC] AD Birrel, BJ Nelson, "Implementing Remote Procedure Calls", ACM Transactions on Computer Systems, pages 39-59, Feb, 1984

[OBSERVER\_DESIGN] E Gamma, R Helm, R Johnson, J Vlissides, "Design Patterns, Elements of Reusable Object-Oriented Software", Addison Wesley, 1995

[MOM] G Banavar, T Chandra, R Strom, D Sturman, "A case for message oriented middleware", Proceedings of 13th International Symposium on Distributed Computing, DISC'99, Bratislava, Slovak Republic, September 1999

[FACES\_PUBSUB] PTH EUGSTER, PA FELBER, R GUERRAOUI, AM KERMARREC, "The Many Faces of Publish/Subscribe", ACM Computing Surveys, 2003

[LINDA] N Carriero, D Gelernter, "The S/Net's Linda kernel", ACM Transactions on Computer Systems (TOCS), 1986

[ELVIN] B Segall, D Arnold, "Elvin has left the building: A publish/subscribe notification service with quenching", Proceedings of AUUG97, 1997

[GRYPHON] G Banavar, T Chandra, B Mukherjee, J Nagarajarao, RE Strom, DC Sturman, "An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems", Proceedings of 19th IEEE International Conference on Distributed Computing Systems (ICDCS'99), 1999

[JEDI] G Cugola, E Di Nitto, A Fuggetta, "Exploiting an Event-Based Infrastructure to Develop Complex Distributed Systems", Proceedings of 20th International Conference on Software Engineering (ICSE'98), 1998

[LOW\_LEVEL\_NAMING] J Heidemann, F Silva, C Intanagonwiwat, R Govindan, D Estrin, D Ganesan, "Building efficient wireless sensor networks with low-level naming", Proceedings of the eighteenth ACM symposium on Operating systems principles, 2001

[DIFFUSION\_FILTERS] J Heidemann, F Silva, Y Yu, D Estrin, P Haldar, "Diffusion filters as a flexible architecture for event notification in wireless sensor networks", USC/ISI Tech. Report, 2002

[MIRES] E Souto, G Guimaraes, G Vasconcelos, M Vieira, N Rosa, C Ferraz, J Kelner, "Mires: a publish/subscribe middleware for sensor networks", Personal and Ubiquitous Computing, Volume 10, Issue 1, Dec 2005

[MATE] P Levis, D Culler, "Maté: a tiny virtual machine for sensor networks", ACM SIGOPS Operating Systems Review, 2002

[PSWARE] Steven Yi Lai, Jiannong Cao, Yuan Zheng, "PSWare: a Publish / Subscribe Middleware Supporting Composite Event in Wireless Sensor Networks", 5th IEEE PerCom International Workshop on Sensor Networks and Systems for Pervasive Computing (PerSeNS’09), March 9 – 13, 2009. Galveston, Texas, USA.