

BeePlex™ Whitepaper

HVAC – The Old-fashioned Way

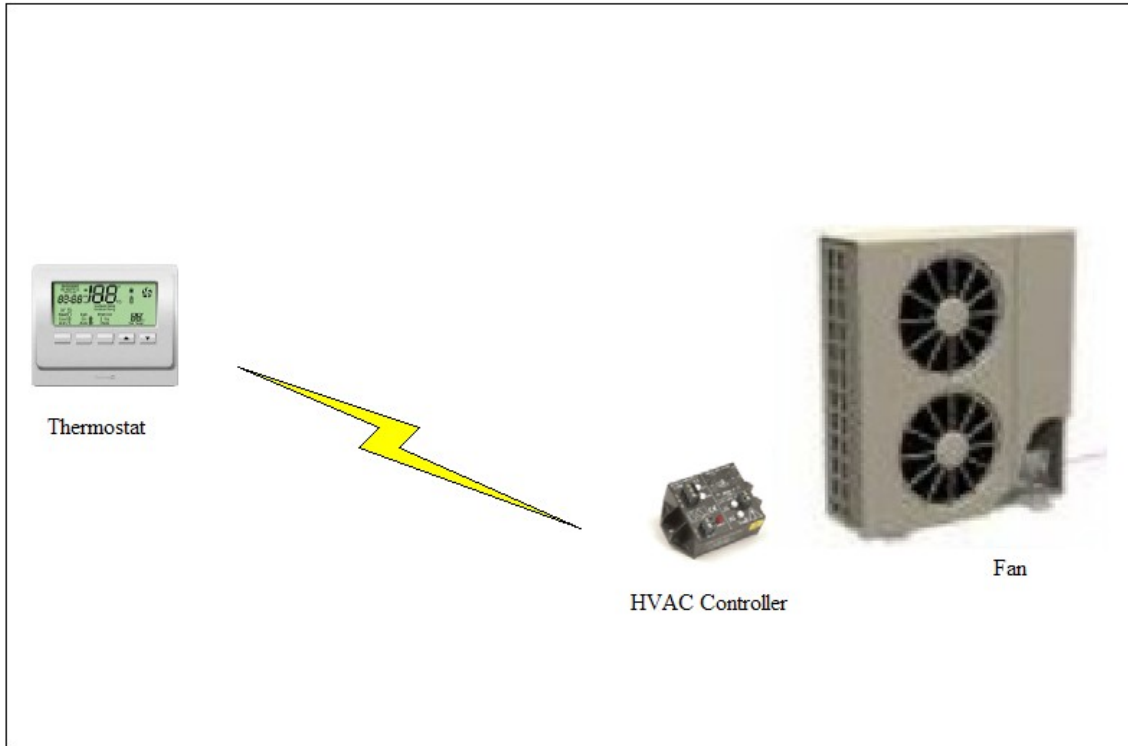


Figure 1. HVAC - The Old-fashioned Way

In the example above a hypothetical thermostat is either connected by wire or by wireless RF to an HVAC Controller module that regulates a fan. If an RF connection is used then the protocol is likely to be proprietary.

Note the following:

- The system is logically “close-coupled” – i.e. the thermostat has a dedicated connection to its associated HVAC Controller in a one-to-one relationship. The two units are said to be “aware” of each other because they co-exist as master and slave. You set the temperature at the thermostat and it controls the fan.
- The system is a “closed” architecture since it does not readily lend itself to expansion. For example, you can’t easily add another HVAC Controller and fan and have it controlled by the same thermostat if you find that a single fan cannot handle the load requirement. Likewise, you can’t easily add another thermostat if perhaps you want the fan to respond to temperature changes in more than one locality.
- There’s an inherent lack of dynamic flexibility in this implementation. If you decide that you want, say, other machinery in the local area to turn on or off if the temperature reaches a pre-defined limit, or for the HVAC system to respond to external events (from other devices or remote

human control or even a time-schedule) then that can't easily be done. The system is controlled at the thermostat and that's pretty well it.

HVAC – The BeePlex™ Way

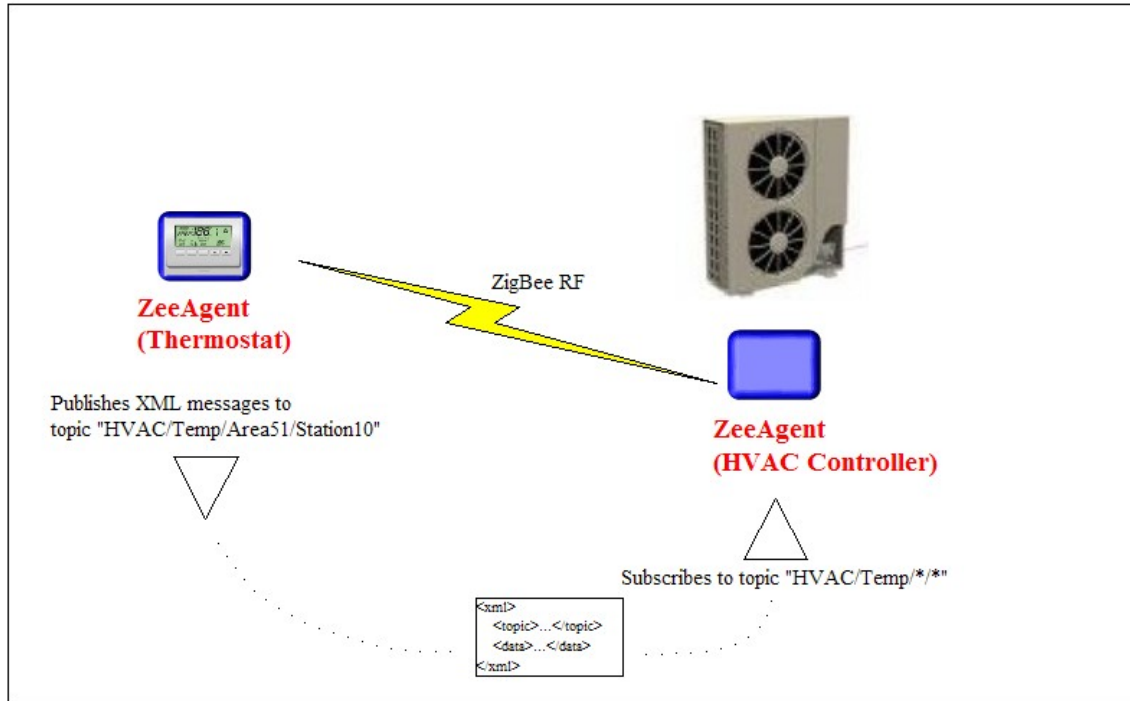


Figure 2. HVAC - The BeePlex Way

BeePlex decouples the HVAC example above by implementing the thermostat and HVAC Controller as BeePlex devices called ZeeAgents.

ZeeAgents are autonomous sensors, monitors and control devices that perform a variety of tasks depending on the type of ZeeAgent device. What makes a ZeeAgent a ZeeAgent is merely the fact that it communicates using open-standard [ZigBee RF](#), can participate in a ZigBee wireless [mesh-network](#), and emits or consumes message packets in a documented [XML](#) schema that includes a “topic” element. ZeeAgents can be self-powered or externally powered depending on their function and specification. Because ZeeAgents are wireless transceivers they eliminate the need for costly wiring runs, core drilling, or conduit.

In the HVAC example above a ZeeAgent device dedicated to temperature sensing is used to monitor temperature in a controlled area. Co-incidentally, a ZeeAgent of the HVAC Controller variety is used to control an HVAC fan unit. The important thing to note here is that the two ZeeAgent devices are logically (and physically) “de-coupled”. What that means is that on an intermittent basis the thermostat blindly emits temperature data messages (in XML format) and has no knowledge of who or what may be consuming or acting on the data it is sending. Likewise, the HVAC controller ZeeAgent picks up temperature data messages via the ZigBee RF mesh-network and responds to them, but it has no awareness of where (or from what device) those messages might have originated.

How do you organize a system like this and stop HVAC controllers responding to temperature data that has nothing to do with them, perhaps in the next room? The answer is that all data in a BeePlex is strictly arranged according to type and categorized using a logical hierarchy of abstract *topic names*. Topics are analogous to subject lines in e-mail messages but are far more powerful because they can be specified as a hierarchical structure like the directory path in a computer file system. It's up to the customer to decide exactly what topic names to set for a device, but once set, the device will associate all data it sends with that topic. In essence the data is "labeled" so that any device or logical entity can receive messages with a particular type of label simply by registering its interest in a particular type of "label" with the BeePlex network.

Sending data with a given topic name is called "publishing" data to a "topic". Publishing data is a fire-and-forget activity (from the device point of view) and the routing and preservation of the message is the responsibility of the underlying BeePlex message transport system.

In order to receive messages a device is said to "subscribe" to messages on a given "topic". A device can subscribe to more than one topic (a subscription list). The interesting thing is that subscribers can use wildcards when they specify a topic name. So, for example, the HVAC controller in our example could specify a subscription topic name of "HVAC/Temp/Area51/Station10" in order to pick up only the temperature data from a thermostat that publishes on topic "HVAC/Temp/Area51/Station10", or it could specify "HVAC/Temp/Area51/*" or "HVAC/Temp/Area51/Station*" to pick up *all* temperature data from *all* temperature sensors in hypothetical "Area51". This makes for a very flexible system because the HVAC controller's behavior can be radically altered by simply specifying a new subscription topic (or list of topics). More than this, because the HVAC controller is consuming XML messages and responding to the temperature data in a specific XML temperature data tag, while ignoring other tagged data it may not understand, it is capable of responding to messages sent by ZeeAgent devices that are not necessarily dedicated to only sensing ambient temperature data in the environment. For example, it could respond to a single temperature XML data element in a message that is actually a status message from an arbitrarily chosen machine within its control area that only incidentally contains temperature data along with other device information. The HVAC could then easily be changed to respond to the core temperature of a particular machine inside "Area51", rather than only the room temperature, simply by changing its subscription topic so that it can pick up messages published by the machine (or an adapter on the machine).

A BeePlex system is inherently scalable because if, for example, you wanted to add another fan to your infrastructure you simply add a new fan and ZeeAgent HVAC controller and set its subscription to the same topic as the first unit. The beauty of BeePlex communication is that all messages are *broadcast* and so are logically available to all ZeeAgents (and other ZeePlex entities) in a network. A BeePlex network can securely span the globe, since the network can bridge the public Internet.

With minimal extra cost other BeePlex components can be deployed in order to extend your BeePlex infrastructure and allow an even richer set of dynamic capabilities and a greater degree of intelligence.

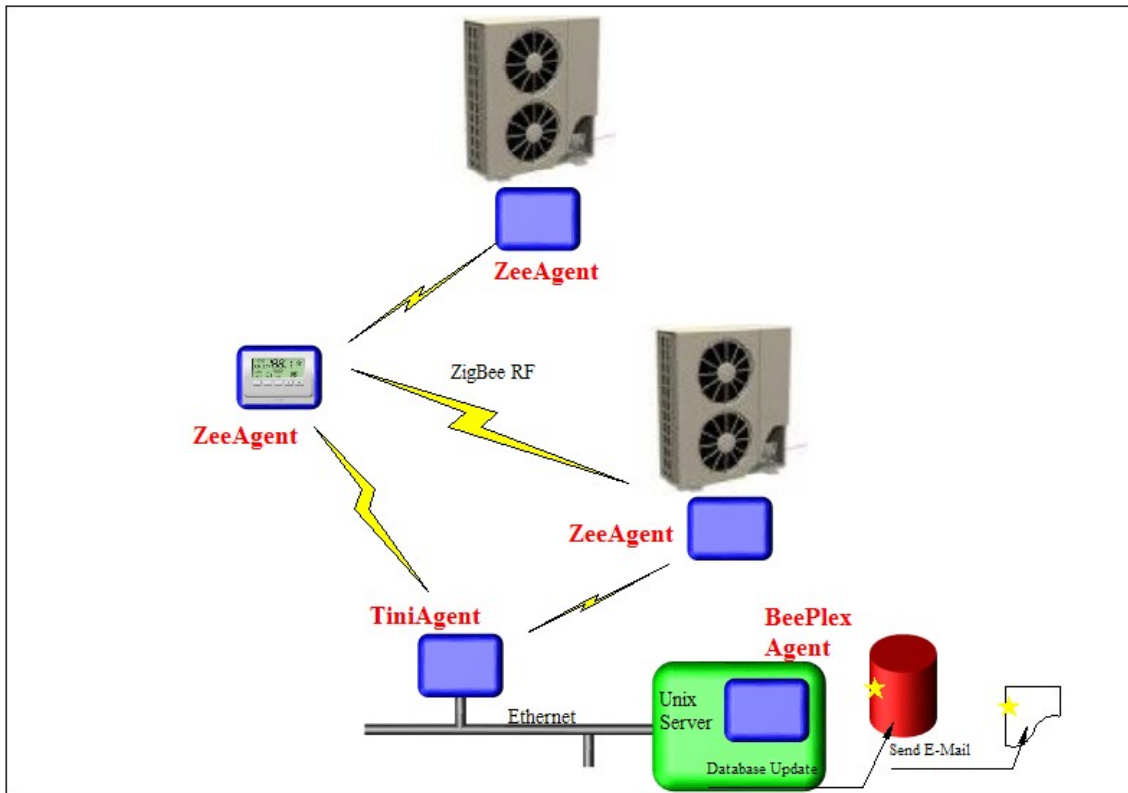


Figure 3. A BeePlex Network

A device called a TiniAgent connects your ZigBee RF mesh-network to an intranet or to the public Internet. TiniAgents are small, low-power devices that provide a gateway into the sophisticated BeePlex [Event Driven Architecture \(EDA\)](#). The BeePlex EDA consists of BeePlex applications that are hosted on one or more Windows or Unix servers called BeePlex Agents. Agents are virtual containers that you can use to quickly and easily create workflows, automated actions, or rules-based programming. Agents host virtual services, which can be configured using a centralized graphical console to perform tasks ranging from updating a database or sending e-mails or text messages.

Agents use the same publication and subscription (pub/sub) protocol as ZeeAgents. In essence they are virtual implementations of ZeeAgents but use their own highly reliable TCP/IP messaging backbone to communicate rather than ZigBee RF.

Using Agents you can create sophisticated rules and sequences of operations without resorting to a software programming language (though you can do that too, if you want to). So, for example, it's a trivial matter to configure a system that starts to switch off heat-creating machinery (either to protect the machines or to reduce the heat) as the temperature rises to critical thresholds. By using ZeeAgent power-switches between a machine and its power source you can implement an emergency shut-off, perhaps. A system like that could be done using an Agent that subscribes to the thermostat ZeeAgent temperature data and applies a rule you create (using the BeePlex graphical console) which could cause the Agent to publish "shut-off"/"switch-on" XML events on a topic that the ZeeAgent power-switch subscribes to. At the same time, it would not take much more than five minutes to implement a BeePlex Service within the same Agent that subscribes to the "shut-off" event and sends a e-mail to a mobile device to warn an operator, or, just as quickly, implement a Service that logs the temperature history and all "shut-off"/"switch-on" events to a database.

Not to labor the point, but you probably noticed that once you have a ZeeAgent power switch, to turn a machine on and off, any entity within the BeePlex network could be set up to publish “on” or “off” events. Using a product-provided HTTP Service in an Agent on a remote host machine, you might choose to switch the machine on and off from a mobile device (like your WAP cell phone browser) or you could implement a Timer Service in the Agent to also control the machine on a clock or calendar schedule.

BeePlex Architecture

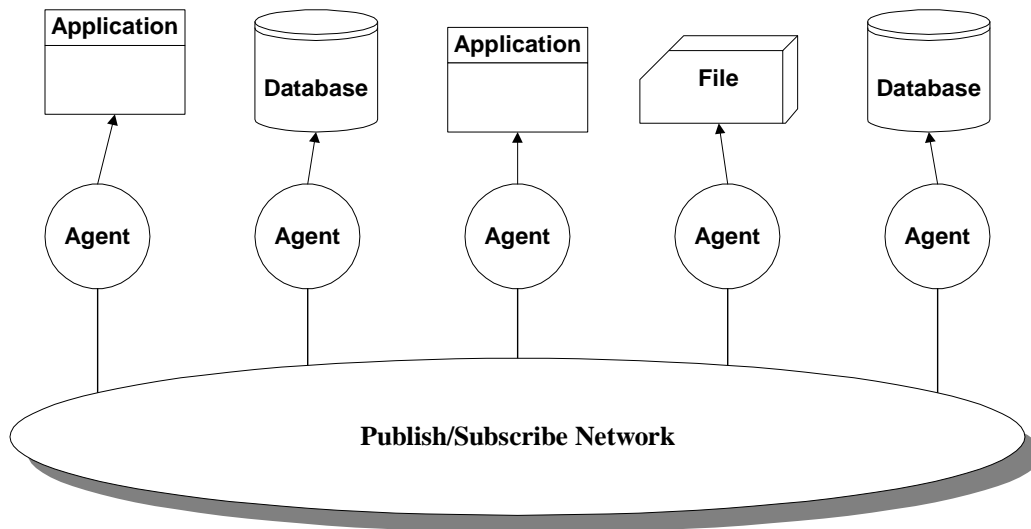


Figure 4. Modular Integration Architecture

BeePlex is process-driven, rather than data driven. BeePlex operates using a network of distributed agents that communicate using the Java Messaging Specification (JMS). It eliminates the complexity of point-to-point integration and allows for rapid deployment with a moderate learning curve.

The primary features of BeePlex are:

- ◆ **Flexibility** – the solution is cheaply and easily customized or re-organized
- ◆ **Modularity** – a successful solution cannot be monolithic, therefore BeePlex has replaceable, discrete components that plug-in, interact and intercommunicate
- ◆ **Asynchronosity** – your integration solution must not depend on the immediate availability of distributed components (brokers, remote servers etc.) in order to function. This is achieved with the use of a robust messaging system that has two-phase commit and store-and-forward messaging capabilities
- ◆ **Extensibility** – BeePlex has a natural, well-defined growth path. As the use of any system grows, and the number of tasks it performs multiplies, the internal complexity of the system increases. BeePlex is able to absorb new requirements with the minimum of impact on the existing infrastructure and to accommodate more tasks with relative ease
- ◆ **Scalability** – as the solution is extended (or the volume of usage increases) the impact to the BeePlex system is linear in fashion, without bottlenecks or exponential use of resources
- ◆ **Reliability** – BeePlex is robust. It has no single points-of-failure and functional fail-over can be easily implemented
- ◆ **Performance** – BeePlex promotes the highest data velocity possible, with the minimum of CPU, memory, storage, and network overhead. The system as a whole tends towards zero-latency.
- ◆ **Simplicity** – the BeePlex solution is uncomplicated and elegant