IoTBPM Server Design Documentation

IoT BPM Drools-BPM Design Architecture

Arduino Tron ESP8266 MQTT Telemetry Transport IoT / (M2M) Machine-to-Machine

Version: 1.10
Published: 06/10/2019

FOR MORE INFORMATION CONTACT:

Steven Woodward
Telephone: (770) 998-3900
Email: info@iotbpm.com
Arduino Tron AI-IoTBPM – Artificial Intelligent Internet of Things http://www.iotbpm.com

Custom Software Development – Executive Order Corporation

www.executiveordercorp.com

Executive Order Corporation - We make Things Smart

Executive Order Corp. is a leading provider of technology that helps global companies design, develop, deploy, and integrate software applications

Media Contact
Company Name: Executive Order Corporation
Contact Person: Steven Woodward
Email: info@iotbpm.com
Phone: (770) 998-3900
Website: www.iotbpm.com

IoTBPM Server - Internet of Things Drools - jBPM Development

DOCUMENT REVISION HISTORY

List changes between each template release.
Increment Release Number by 1 is only between published versions.
(Update the Page Heading when published)

Version #	Date	Revised By	Description of Changes
0.5	3/04/2016	Steven Woodward	First Draft of AI-IoTBPM Internet of Things.
1.02	9/03/2018	Steven Woodward	Arduino Tron ESP8266 MQTT Telemetry Transport (M2M) Machine-to-Machine.
1.10	06/10/2019	Steven Woodward	AI-IoTBPM Applications using Drools-jBPM Expert System Engine Applications.

SUPPORTING DOCUMENTS

Document Number	Document Name Description
1	Business Requirements Document – Executive Order Document
2	SRS (System Requirements & Use-Cases) Document – EO Document
3	High-level Design Document – Executive Order Document
4	IF-SPECS Documents – Executive Order Document
5	Low Level Design – Application Development – Executive Order Document

About Executive Order Corporation

We make Things Smart

Executive Order Corp. provides custom software built by software professionals. We specialize in IoTBPM (Internet of Things), desktop and web-enabled IT solutions for small and large business enterprises. Our professional offerings span business and technology consulting, business application development, mobile messaging solutions, custom web design, E-commerce development, web maintenance, website re-engineering, website optimization for search engine submission, internet marketing hosting solutions for enterprises, GPS, IoTBPM (Internet of Things),remote sensing services and development program architecture of AI-Drools and jBPM (Business Process Management).

TABLE OF CONTENTS

1 IoTBPM Server-Internet of Things

1.1 What is IoTBPM Server?

1.1.1 IoTBPM Server definition

- IoTBPM Server is a Business Process Management Engine for IoT Device Orchestration.

1.1.2 When IoT meets BPM

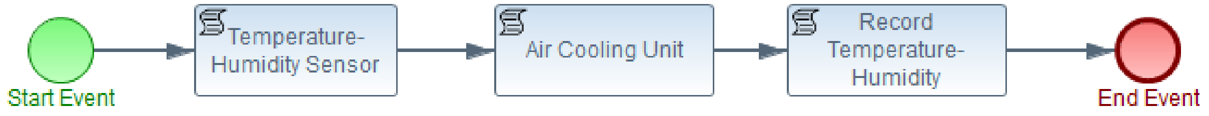
This IoT BPM Server documentation will help you understand exactly what IoTBPM Server is and how it is relevant to IoT Device Orchestration and IoT Device Ontology (AI-IoT Device awareness, state of being, knowledge of real-world objects, events, situations and abstract concepts).

In the context of IoT Devices when we say BPM, all we mean is "a sequence of tasks that allows us to achieve a goal." In IoT BPM orchestrated each task-goal can be carried out by a different IoT Device, enterprise RESTful server, human task, or any other integrated service.

The IoTBPM Server ensures that once started, the BPM is carried out fully and retries any steps in case of failures. The IoTBPM Server maintains an audit log, so that the progress of BPM can be monitored. The IoTBPM Server is fault tolerant and scales seamlessly to handle growing transaction volumes.

1.1.3 IoTBPM Server Introduction

A company's end-to-end IoT BPM workflow will almost always span more than one IoT Device or enterprise servers. These IoT Device and Services Integration into the enterprise can be mission critical to the business and is rarely modeled and monitored. Often, the message flow of events through different IoT Devices is expressed only implicitly in code.



These cross-microservice workflows can be a company's core operation drivers, often they are rarely modeled and monitored; and the flow of events through different IoT Devices is usually expressed only in low-level Interface Flow specifications and rarely depict Business Enterprise goals.

In that case, enterprises often ask; how can we ensure visibility of workflows and provide status and error monitoring? How do we guarantee that overall flows always complete, even if single IoT Devices fail? Or how do we at least recognize that a flow is stuck so we can go in and fix it?

IoTBPM Server is an open-source BPM workflow engine for IoT Devices orchestration that provides:

1. **Visibility** into the state of a company's end-to-end IoT BPM workflows.
2. **BPM Orchestration** based on the current state of a BPM; IoTBPM Server publishes "commands" as events that can be consumed by one or more IoT Devices, ensuring that BPMs progress according to their definition.
3. **Monitoring for timeouts** or other BPM errors with the ability to path, such as stateful retries or escalation to teams that can resolve an issue manually.

IoTBPM Server was designed to operate at a very large scale and to achieve this, it provides:

- **Horizontal scalability** and no dependence on an external database; IoTBPM Server writes data directly to the filesystem on the same servers where it's deployed. IoTBPM Server makes it simple to distribute processing across a cluster of machines to deliver high throughput.
- **A message-driven architecture** where all BPM-relevant events are written to an append-only log, providing an audit trail and a history of the state of a BPM.

- **A publish interaction model**, which enables IoT Devices that connect to IoTBPM Server to maintain a high degree of control and autonomy, including control over BPM sessions. These properties make IoTBPM Server resilient, scalable, and reactive.
- **Visual BPMs modeled in ISO-standard BPMN 2.0** so that technical and nontechnical stakeholders can collaborate on BPM design in a widely-used modeling language.
- **A language-agnostic client model**, making it possible to build an IoTBPM Server client in just about any programming language that an organization uses to build IoT Devices.
- **Operational ease-of-use** IoTBPM Server does not require a cluster coordinator. Because all nodes in an IoTBPM Server cluster are equal, it's easy to scale, and it plays nicely with modern resource managers and container orchestrators.

1.1.4 IoTBPM Server IoT Device solution

IoT Devices architectures have become increasingly popular in recent years for providing new functionality and services; butr, the very principles that make IoT Devices architectures so appealing; (i.e., loose coupling, independent deployment, and new data points) also present significant challenges.

A core point of IoT Devices architecture is that each IoT Device is responsible for one and only one business capability. Each IoT Device exists to contribute to a broader workflow. The company goal succeeds only when the end-to-end workflow is successful, so ensuring workflow quality is critical.

In an IoT Devices architecture, where each IoT Device is responsible for only one carefully-scoped business capability, who’s responsible for the end-to-end workflow?

By default, no one. In fact, the end-to-end workflow might not even be formally documented within the company, with the flow of events from IoT Device to enterprise service expressed only implicitly in code.

Many IoT Devices architectures rely on a pure choreography pattern for communication, where IoT Devices collaborate by publishing events to and consuming events from a messaging platform without a central controller.

Typical implementation of choreography communications does not provide:

1. **Visibility** into the current state of the business: How many end-to-end workflow instances are in progress, and what’s their state? How many workflow instances did not complete successfully? What’s the average amount of time that it takes to complete a workflow instance or a specific step in a workflow?
2. **Failure handling** to ensure workflow completion even when errors occur: If a service that’s a part of a workflow fails, who’s responsible for handling the failure? What’s the retry logic of the workflow? What are our rules for escalating?

Note: when we say, “workflow instance”, we mean “a single occurrence of our BPM workflow.”

Current IoT Device architectures concentrate on producing good IoT software (at the code service-level) but fail to go beyond this, to the business outcomes level. The bigger picture is the success of the business workflow is what ultimately makes or breaks the business solution process.

The IoTBPM Server provides the solution with the benefits of loose coupling IoT Devices architecture while ensuring robust end-to-end business enterprise BPM workflows success.

With the IoTBPM Server platform, we go beyond loose coupling IoT Devices architecture with Drools providing AI to our IoT Device BPM workflow. The IoTBPM Server uses the concepts of Drools AI applied to the BPM workflow to make smarter decisions by our IoT Devices.

1.1.5 IoTBPM Server workflow solution

IoTBPM Server is a workflow engine that manages your end-to-end IoT Device business processes solution. It monitors the state of activities in a workflow and determines the status of each BPM activity according to defined processes.

IoTBPM Server provides high throughput, low latency, and horizontal scalability. The IoTBPM Server does not require a central database component and instead leverages event sourcing, meaning that all changes to a workflow's state are captured as events and stored. In the IoTBPM Server this state is written directly to the filesystem on the server where the IoTBPM Server is running, and the current state of a workflow can be derived from the events stored on the server.

IoTBPM Server uses a client / server messaging model. The IoTBPM Server is a remote engine that runs in its own program space on a Java Virtual Machine. Either in the cloud or on your own servers.

The IoTBPM Server end-to-end workflow enables users to:

- Explicitly define and model workflows that span multiple IoT Devices
- Gain detailed visibility into workflow performance and an understanding of any problems
- Orchestrate IoT Devices that fulfill a defined workflow to ensure that all workflow instances are completed according to business requirements

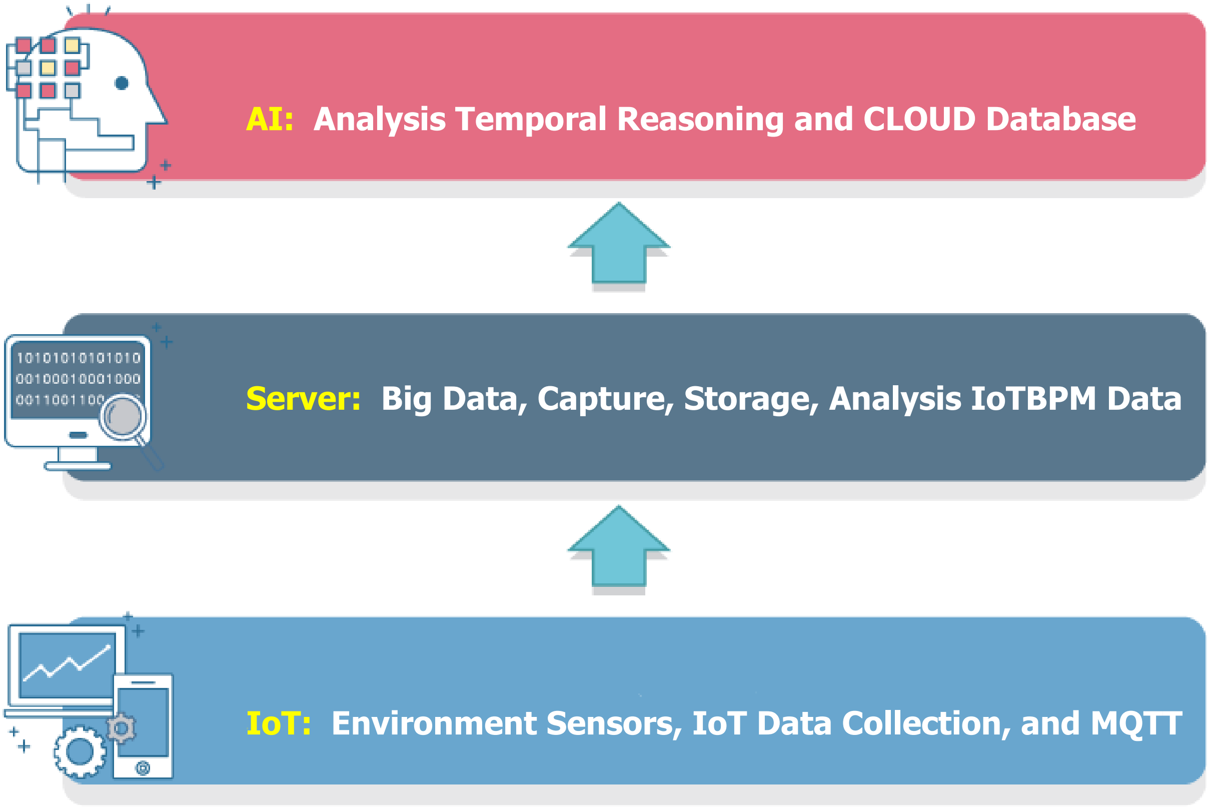
1.1.6 IoTBPM Server workflow aware event monitoring

The IoTBPM Server workflow-aware event monitoring is defined as.

- Your business relies on the successful completion of one or more workflows
- These workflows are carried out by independently-developed and independently-deployed IoT Devices that need to communicate with a central control mechanism
- You need insight into the performance of a given IoT services and visibility into the end-to-end health of your business workflows.

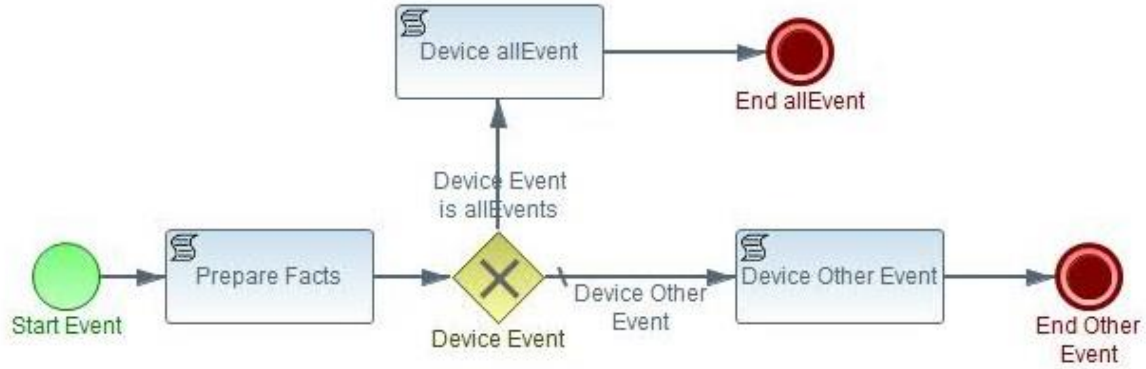
IoTBPM Server can work alongside the components you already use in your event-driven architecture without requiring you to replace or remove any existing systems to provide workflow visibility.

Here's a simple diagram showing how IoTBPM Server can be used for visibility into workflows that span IoT Devices:


AI: Analysis Temporal Reasoning and CLOUD Database
Server: Big Data, Capture, Storage, Analysis IoTBPM Data
IoT: Environment Sensors, IoT Data Collection, and MQTT

In this example, the IoTBPM Server receives IoT Device event messages that are published to your messaging platform and correlates them to your predefined workflow, which has been modeled visually in BPMN 2.0 and deployed to the IoTBPM Server.

The workflow-relevant events processed by IoTBPM Server can be used to power dashboards, to build workflow service ques instances to have technicians to fix that needs attention.



In this implementation, IoTBPM Server is monitoring the health of an individual service and gives visibility into:

- The current state of a business: How many service workflows are currently in flight?
- Are there in-flight processes that are "stuck" due to an (outage) issue?
- What is the average end-to-end process duration?

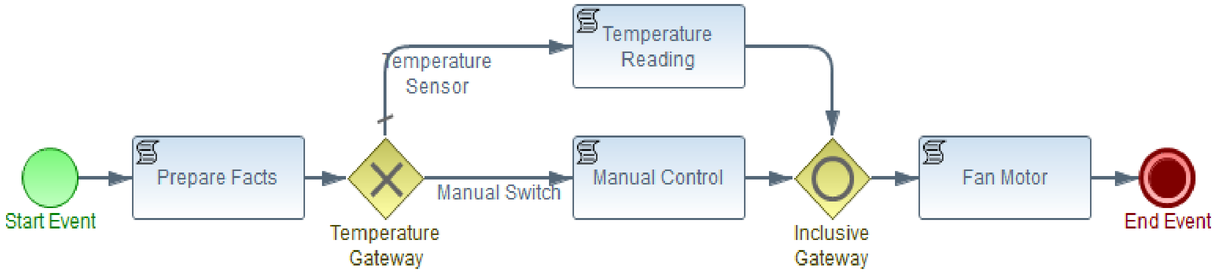
In this example, IoTBPM Server interacts with the IoT Devices that participate in a workflow and provides matrix information. Now, let's look at how to extend this "visibility" solution to take advantage of IoTBPM Server's orchestration capabilities.

1.1.7 IoTBPM Server workflow IoT Device Orchestration

IoTBPM Server workflow and IoT Devices orchestration makes it the watchdog to failure handling and problem retries that we process out end-to-end business workflow.

IoT Devices orchestration is sometimes considered to be at odds with core IoT Devices principles such as loose coupling and independent deployment. IoT Devices orchestration can be implemented in a way that aligns with these principles, and IoTBPM Server was designed accordingly.

A diagram showing how IoTBPM Server can be used for IoT Devices orchestration:



- Temperature Sensor - an IoT device transmitting a DHT11 digital ambient temperature and humidity Sensor Reading.
- Pushbutton Switch - an IoT device transmitting a HTTP Fan Motor On / Off Command to the IoTBPM Server.

This architecture is similar to the "IoTBPM Server for visibility" architecture described above. A notable difference is that in our diagram, we have removed the messaging platform layer, and IoTBPM Server communicates directly with the IoT Devices that participate in a workflow.

It is still possible to use IoTBPM Server for IoT Devices orchestration without removing your existing messaging platform–in addition to subscribing to workflow-relevant events as illustrated in the "visibility" solution, the IoTBPM Server would simply publish events to the messaging platform, as well.

You can think of IoTBPM Server's approach to workflow orchestration as a state machine, that uses BPM Service Tasks. A Service Task is a Task that uses some sort of service, which could be a Web service or an automated application, or IoT Device. When a workflow instance progresses to a Service Task, the IoTBPM Server sends a message to notify the responsible worker service then waits for the worker to complete that Service Task.

Once a Service Task is completed, the worker service notifies the IoTBPM Server and the flow continues with the next step. If the worker fails to complete the Service Task, the workflow remains at the current step, potentially retrying the task until it eventually succeeds or escalating to a different team if human intervention is required.

The IoTBPM Server decouples the creation of task notification messages from the actual performing of the work, meaning that IoTBPM Server can send task notification messages at the maximum possible rate regardless of whether there is a worker service available to take on the work.

This IoT Devices orchestration approach provides a full level of visibility into workflows and workflow instances while also ensuring that workflows are completed according to their definition, even when there are failures along the way.

- **IoTBPM Servers scales horizontally**

The ability to scale to handle high-throughput workloads is critical to IoTBPM Server's role in IoT Devices orchestration. As event volume grows, it may become necessary to distribute the IoTBPM Server across a cluster of machines in order to meet throughput requirements. The IoTBPM Server uses isolation to provide horizontal scalability.

- **IoTBPM Server is fault tolerant and highly available**

The IoTBPM Server allows users to configure and define a replication and failover strategy through TCP/IP routing. The IoTBPM Server provides fault tolerance and high availability without the need for an external database, storing data directly on the filesystems in the servers where it's deployed. The IoTBPM Server is completely self-contained and self-sufficient.

- **IoTBPM Server allows workflows to be defined visually**

ISO-standard BPMN 2.0 is the default modeling language for defining workflows in IoTBPM Server. Workflows are defined visually and with full participation of both technical and non-technical stakeholders, along with vendors.

- **IoTBPM Server is fully message-driven**

The IoTBPM Server and IoT clients communicate entirely by HTTP message protocol, making it possible to adhere to the principle of loose coupling and to enable asynchronous communication between IoTBPM Server and the IoT Devices that participate in a workflow.

A very commonly used M2M protocol used by IoT devices is the MQTT (Message Queuing Telemetry Transport) protocol. The IoTBPM Server is fully compatible with the MQTT protocol which uses a publish / subscribe communications model and allows for data to be sent and received asynchronously.

- **IoTBPM Server event history and auditing**

The IoTBPM Server logging makes it easy to stream workflow event data into a storage system so that this data can be available indefinitely. This is important for reporting and visibility, and depending on your industry, it might be necessary for regulatory reasons.

1.1.8 IoTBPM Server use case - IoT Devices orchestration

We often talk about the IoTBPM Server in the context of the IoT Devices orchestration use case because it's a problem te IoTBPM Server solves really well, but the IoTBPM Server can be applied to use cases beyond IoT Devices orchestration.

IoTBPM Server is a workflow engine that handles a wide range of high-throughput use cases. Here are some of the general characteristics of the IoTBPM Server:

- IoTBPM Server is designed for large-scale workflows. IoTBPM Server does not rely on an external database and instead stores data in the form of an immutable log directly on the servers where the IoTBPM Server is deployed; this architecture is key to IoTBPM Server's ability to handle high throughput and to scale horizontally.
- Communication between the IoTBPM Server and IoT clients is handled by HTTP message protocol, making it possible to adhere to the principle of loose coupling and to enable asynchronous communication between IoTBPM Server and the IoT Devices and services that participate in an IoT BPM workflow.
- IoTBPM Server currently covers all of the BPMN 2.0 BPM symbols. The IoTBPM Server adds support for new symbols and the IoTBPM Server offers complete coverage of BPMN 2.0 symbols for workflow automation.

1.1.9 IoTBPM Server summary

IoT BPM Service is well suited for IoT Devices orchestration. It does not rely on a relational database to manage the state of active workflow instances, this is an inherent limit to its scalability in terms of throughput (as measured, for example, by workflow instances started per second).

The Internet of Things (IoT) refers to a network of connected devices collecting and exchanging data and processes over the internet. IoT promises to provide "smart" environments and smart products. While this data is useful, there is still "a disconnect" in integrating these IoT devices with mission-critical business processes and corporate cloud data awareness.

The task of moving IoT devices beyond "connected" to "smart" is daunting. Moving beyond collecting IoT data and transitioning, to leveraging this new wealth of IoT data, to improving the smart decision-making process is the key to automation. Artificial Intelligence (**AI**) will help these IoT devices, environments and products to self-monitor, self-diagnose and eventually, self-direct. This is what we said in several of our books, "If a machine thinks, then a machine can do."

However, one of the key concepts in enabling this transition from connected to smart is the ability to perform **AI Decisions**. The traditional analytic models of pulling all data into a centralized source such as a data warehouse or analytic sandbox is going to be less useful. We are not trying just to analyze complex IoT data; we are trying to make "smart decisions" and take actions based on our AI Analytics of IoT devices.

***IoT Definition** – IoT is the integration of computer-based systems into our physical-world.*

Our world is increasingly linked through the number of already connected IoT devices. IoT components are equipped with sensors and actuators that enable sensing, acting, collecting and exchange data via various communication networks including the internet. These IoT devices such as wearable, GPS, smartphones, connected cars, vending machines, smart homes, and automated offices are used in areas such as supply chain management, intelligent transport systems, robotics, and remote healthcare. Businesses can rapidly gain a competitive edge by using the information and functionalities of IoT devices (sensors and actuators). Business processes can use IoT information to incorporate real-world data, to make informed decisions, optimize their execution, and adapt itself to context changes.

This increase in processing power of IoT devices and enables it to take part in the execution of the business logic. This way IoT devices can aggregate and filter data and make decisions globally by executing parts of the business logic where central control is required, reducing both the amount of exchanged data and adding the benefit of central processing involvement. This gives us an "IoT Hive" and not just isolated, disparate devices, executing in a vacuum without coronation and corporation.

2 IoTBPM Server Drools-jBPM Architecture

2.1 IoT Internet of Things jBPM Solution Architecture

2.1.1 IoTBPM Server jBPM Design Methodology

The following key design methodologies are utilized in the AI-IoTBPM Server architecture.

jBPM is a flexible Business Process Management (BPM) Suite. A business process allows you to model your business goals by describing the steps that need to be executed to achieve those goals, and the order of those goals are depicted using a BPM flow chart. **jBPM -** Flexible Business Process Management suite allowing you to model your business goals by describing the steps that need to be executed to achieve those goals.

- **jBPM –** Allows you to model business goals by describing the steps that need to be executed to achieve those goals and the order of those goals are depicted using a flow chart diagram.
  - jBPM is a flexible Business Process Management (BPM) Suite. It allows you to model, execute and monitor business processes throughout their life cycle through GUI.
  - Business Process Models are expressed in a given process modeling language. OMG commissioned the development of the BPMN2.

The BRMS is making the Enterprise Business decisions and the BPM is executing the Business Goal.

2.1.2 Design Use Case Methodology

The power of the IoT device increases greatly when business process (jBPM) can use them to provide information about our real-world as well as execute IoT device actions as part of our business process. The jBPM-BPMN modular allows us to define both the business processes and IoT devices behavior at the same time using one (BPM) diagram. In our examples, we will be adding Drools-jBPM to IoT devices. Making **"Things Smart"** is the application of AI to IoT platform via Drools-Rules Inference Reasoning, jBPM and ES-Expert Systems Architecture.

With the use of AI Drools-jBPM analysis and reasoning in IoT Devices, we can orchestrate dissimilar devices that normally have no awareness of each other. This creates opportunities for direct integration of computer-based into the physical-world that has never been available before. This results in improved efficiency, accuracy and economic benefits by increased automation - reduced intervention. This IoT BPM orchestration of IoT devices gives us the ability for action after our AI Drools decision.

2.1.3 IoTBPM Server Architecture

There are two main components in IoTBPM Server's architecture:

- **IoT Device Client**

  IoT Device Clients is the SOC embed application, the IoT Device that executes your business logic to connect to the IoTBPM Server. Clients have two primary uses:

  - Carrying out business logic (starting BPM instances, publishing messages, working on tasks and taking action in our physical world)
  - Handling operational issues (updating BPM instance payloads, resolving incidents)
  - Clients connect to the IoTBPM Server via HTTP / MQTT protocol base transport.

- **IoTBPM Server**

  The IoTBPM Server is the distributed BPM engine that keeps state of active BPM instances.

The IoTBPM Server can be partitioned for horizontal scalability and replicated for fault tolerance. An IoTBPM Server deployment will often consist of more than one server.

It's important to note that the application business logic, the BPM lives in the IoTBPM Server. The IoTBPM Server's responsibilities are:

1. Storing and managing the state of active BPM instances
2. Making AI decisions for the BPM process instance
3. Distributing work items to IoT Device clients

2.1.4 BPMN Quick Primer

For those who are new to BPM, it is a good idea to present a quick primer overview. This will provide better context and understanding for the next chapters. There are excellent training documents available for both BPM and Drools online.

Business Process Model and Notation 2.0 (BPMN) is an industry standard for BPM modeling and execution. A BPMN, BPM is an XML document that has a visual representation.

This duality makes BPMN very powerful. The XML document contains all the necessary information to be interpreted by BPM engines. At the same time, the visual representation contains information to be understood by humans, even when they are non-technical people. The BPMN model is source code and documentation in one artifact.

The following is an introduction to BPMN 2.0, its elements and their execution semantics. It tries to briefly provide an intuitive understanding of BPMN's power. It does not cover the entire feature BPM set. For more exhaustive BPMN resources, see the references on the www.iotbpm.com website.

2.1.5 BPMN 2.0 Elements

- **Sequence Flow: Controlling the Flow of Execution**

A core concept of BPMN is sequence flow that defines the order in which steps in the BPM happen. In BPMN's visual representation, a sequence flow is an arrow connecting two elements. The direction of the arrow indicates their order of execution.

You can think of BPM execution as tokens running through the BPM model. When a BPM is started, a token is spawned at the beginning of the model. It advances with every completed step. When the token reaches the end of the BPM, it is consumed and the BPM instance ends. IoTBPM Server's task is to drive the token and to make sure that the IoT job workers are invoked whenever necessary.

- **Tasks: Units of Work**

The basic elements of BPMN BPMs are tasks, atomic units of work that are composed to create a meaningful result. Whenever a token reaches a task, the token stops and IoTBPM Server creates a job and notifies a registered worker to perform work. When that handler signals completion, then the token continues on the outgoing sequence flow.

Choosing the granularity of a task is up to the person modeling the BPM. For example, the activity of processing an order can be modeled as a single Process Order task or as three individual tasks Collect Money, Fetch Items, Ship Parcel. If you use IoTBPM Server to orchestrate IoT Devices, one task can represent one IoT Device invocation.

- **Gateways: Steering Flow**

Gateways are elements that route tokens in more complex patterns than plain sequence flow.

BPMN's exclusive gateway chooses one sequence flow out of many based-on data items in the BPM. BPMN's parallel gateway generates new tokens by activating multiple sequence flows in parallel.

- **Events: Waiting for Something to Happen**

Events in BPMN represent things that happen. A BPM can react to events (catching event) as well as emit events (throwing event). The circle with the envelope symbol is a catching message event.

It makes the token continue as soon as a message is received. The XML representation of the BPM contains the criteria for which kind of message triggers continuation.

Events can be added to the BPM in various ways. Not only can they be used to make a token wait at a certain point, but also for interrupting a token's progress.

- **Sub Processes: Grouping Elements**

Sub Processes are element containers that allow us to define common functionality. For example, we can attach an event to a sub process's border. When the event is triggered, the sub process is interrupted regardless which of its elements is currently active.

- **Parameter Mapping**

A BPMN BPM instance can have data associated with it, through the BPM Start Process Map parameter instance. The Data Map parameter carries contextual data of the BPM instance that is required by job workers to do their work. It can be provided when a BPM instance is created.

The Start Process Map parameter is the link between BPM instance context and Java POJO classes. The Start Process Map param Mapping allows copying the value of process variables to parameters of the work item. Each type of work can define result parameters that will (potentially) be returned after the work item has been completed. A result mapping can be used to copy the value of the given result parameter to the given variable in this process and then read it in Java.

Parameter mappings are pairs of data expressions. Every mapping has a source and a target expression. The source expression describes the path in the source document from which to copy data. The target expression describes the path in the new document that the data should be copied to.

- **Events**

Events in BPMN can be thrown (i.e. sent), or caught (i.e. received), respectively referred to as throw or catch events, e.g. message throw event, timer catch event.

Additionally, a distinction is made between start, intermediate, and end events:

> **Start events** (catch events, as they can only react to something) are used to denote the beginning of a process or sub-process.
>
> **End events** (throw events, as they indicate something has happened) are used to denote the end of a particular sequence flow.
>
> **Intermediate events** can be used to indicate that something has happened (i.e. intermediate throw events), or to wait and react to certain events (i.e. intermediate catch events).

- **Intermediate Events**

Intermediate catch events can be inserted into your process in two different contexts: normal flow, or attached to an activity, and are called boundary events. In normal flow, an intermediate throw event will execute its event (e.g. send a message) once the token has reached it, and once done the token will continue to all outgoing sequence flows.

An intermediate catch event, however, will stop the token, and wait until the event it is waiting for happens, at which execution will resume, and the token will move on.

- **Boundary Events**

Boundary events provide a way to model what should happen if an event occurs while an activity is currently active. For example, if a process is waiting on a user task to happen which is taking too long, an intermediate timer catch event can be attached to the task, with an outgoing sequence flow to notification task, allowing the modeler to automate sending a reminder email to the user. A boundary event must be an intermediate catch event and can be either interrupting or non-interrupting. Interrupting, in this case means that once triggered, before taking any outgoing sequence flow, the activity the event is attached to will be terminated. This allows modeling timeouts where we want to prune certain execution paths if something happens, e.g. the process takes too long.

- **Message Events**

A message can be referenced by one or more message events. It holds the information which is used for the message correlation.

The correlation key is specified as JSON Path expression. It is evaluated when the message event is entered and extracts the value from the BPM instance payload. The value must be either a string or a number. If the correlation key can't be resolved or is not a string or number, then an incident is created.

- **Message Start Events**

A message start event allows creating a BPM instance by publishing a named message. A BPM can have more than one message start event, each with a unique message name. We can choose the right start event from a set of start events using the message name.

When deploying a BPM, the following conditions apply:

- The message name must be unique across all start events in the BPM definition.
- When a new version of the BPM is deployed, subscriptions to the message start events of the old version will be canceled. This is true even if the new version has different start events.
- Currently, a BPM that has message start events cannot have a none start event.

The following behavior applies to published messages:

- A message is correlated to a message start event if the message name matches. The correlation key of the message is ignored.
- A message is not correlated to a message start event if it was published before the subscription was created, i.e. before the BPM was deployed. This is because the message could have been already correlated to the previous version of the BPM.

- **Boundary Events**

When attached to the boundary of an activity, a message catch event behaves in two ways:

If it is non-interrupting, it will spawn a new token which will take the outgoing sequence flow, applying any defined output mapping with the message payload. It will not terminate the activity that it's attached to. If it is interrupting, it will terminate the activity before spawning the token.

- **Timer Boundary Events**

As boundary events, timer catch events can be marked as non-interrupting; as a simple duration; however, a non-interrupting timer event isn't particularly useful. As such, it is possible to define repeating timers, that is, timers that are red every X amount of time, where X is a duration.

On the other hand, in the case of an interrupting boundary event, a cycle is not particularly useful, and here it makes sense to use a simple duration; this allows you, for example, to model timeout logic associated to a task.

- **Timer Start Events**

A timer start event can be used to periodically create an instance of a BPM. A BPM can have multiple timer start events along with other types of start events. The interval is expressed in the same way as in timer boundary events.

- **Embedded Sub Process**

An embedded sub process can be used to group BPM elements. It must have a single none start event. When activated, execution starts at that start event. The sub process only completes when all contained paths of execution have ended.

- **BRMS - BPM workflows**

BPMs are flow-chart blueprints that denote the orchestration of Business Process tasks (a goal).

Every task represents a piece of business logic such that their ordered execution produces a meaningful result. A BPM is your implementation of the business logic of a task, the goal.

- **Sequences**

The simplest kind of BPM is an ordered sequence of tasks. Whenever BPM execution reaches a task, the IoTBPM Server creates a job and triggers the invocation of a job worker.

You can think of IoTBPM Server's BPM orchestration as a state machine. Orchestration reaches a task, triggers the worker and then waits for the worker to complete its work. Once the work is completed, the flow continues with the next step. If the worker fails to complete the work, the BPM remains at the current step, potentially retrying the job until it eventually succeeds.

- **Data Flows**

As IoTBPM Server progresses from one task to the next in a BPM, it can move custom data in the form of variables along. This data is called the BPM payload and is created whenever a BPM is started.

Every BPM can read the current variables and modify them when completing a job so that data can be shared between different tasks in a BPM. A BPM model can contain simple, yet powerful payload transformation instructions to keep workers decoupled from each other.

- **Databased Conditions**

Some BPMs do not always execute the same tasks but need to choose different tasks based on variable payload and conditions:

    - The diamond shape with the "X" in the middle is a special step marking that the BPM decides to take one or the other path.

Conditions used in the BPM are extract properties and values from the current payload document.

- **Events**

Events represent things that happen. A BPM can react to events (catching event) as well as emit events (throwing event). For example, there are different types of events like message or timer.

- **Fork / Join Concurrency**

In many cases, it is also useful to perform multiple tasks in parallel. This can be achieved with Fork / Join concurrency:

    - The diamond shape with the "+" marker means that all outgoing paths are activated, and all incoming paths are merged.

Concurrency can also be based on data, meaning that a task is performed for every data item:

    - Local variable list for BPM processing.

- **BPMN 2.0**

The IoTBPM Server uses BPMN 2.0 for representing BPMs. BPMN is an industry standard which is widely supported by different vendors and implementations.

Using BPMN 2.0 ensures that BPMs can be interchanged between the IoTBPM Server and other types of BPM systems, even events across vendors, standalone and cloud base BPM systems.

- **State Machine**

The IoTBPM Server manages stateful entities: Jobs, BPMs, etc. Internally, these entities are implemented as State Machines managed by a stream processor.

The concept of the state machine pattern is simple: An instance of a state machine is always in one of several logical states. From each state, a set of transitions defines the next possible states.

Transitioning into a next state is the progression flow of the BPM to reach a goal.

- **Events and Commands**

Every state change in a state machine is called an event. IoTBPM Server publishes every event as a record on the stream.

State changes can be requested by submitting a command. An IoTBPM Server receives commands from two sources:

1. Clients send commands remotely. Examples: Deploying BPMs, starting BPM instances, creating and completing jobs, etc.
2. The IoTBPM Server itself generates commands. Examples: Locking a job for exclusive processing by a worker, etc.

Once received, a command is published as a record on the addressed stream. This creates transitioning into a new state that may produce outputs or variable changes of the events.

- **Stateful Stream Processing**

A stream processor reads the record stream sequentially and interprets the commands with respect to the addressed BPM entity's lifecycle. More specifically, a stream processor repeatedly performs the following steps:

1. Consume the next command from the stream.
2. Determine whether the command is applicable based on the state lifecycle and the entity's current state.
3. If the command is applicable: Apply it to the state machine. If the command was sent by a client, send a HTTP or MQTT reply/response.
4. If the command is not applicable: Rejected; if it was sent by a client, send an error reply/response.
5. Publish an event reporting the entity's new state.

For example, processing the Create Job command produces the event Job Created.

- **Command Triggers**

A state change which occurred in one entity can automatically trigger a command for another entity. Example: When a job is completed, the corresponding BPM instance shall continue with the next step. Thus, the Event Job Completed triggers the command Complete Activity.

- **Function as a Service (FaaS)**

The IoTBPM Server computing execution model runs in stateless compute containers that are event-triggered, ephemeral (may last for one invocation). This application architecture is designed to incorporate third-party “Backend as a Service” (BaaS) and/or custom code to run in managed, ephemeral containers on a “Functions as a Service” (FaaS) platform.

The Functions as a Service (FaaS) architectures, where engineers can deploy an individual function or a piece of business logic, is similar to the functions you’re used to writing in programming languages, small, separate, units of logic that take input arguments, operate on the input and return the result. This independent, server-side, logical function messaging architecture works well with our IoT Devices that will send specific messages indicating a condition. Software Developers and Business Analysis can leverage this to deploy an individual “function”, action, or piece of business logic form an IoT Device.

- **Ephemeral**

FaaS provides an Ephemeral architecture since FaaS are designed to spin up quickly, do their work and then shut down again. The IoTBPM Server is Ephemeral since it is event-triggered and can be invoked directly by IoT Devices or by events from other cloud services such as HTTP requests or inbound message notifications. This FaaS Ephemeral architecture is the glue between your IoT Device and services in your enterprise environment.

2.2 IoT Internet of Things Drools Solution Architecture

2.2.1 IoTBPM Server Drools Design Methodology

The following key design methodologies are utilized in the AI-IoTBPM Drools Server architecture.

Drools is a flexible Business Rules Management System (BRMS) Logic Integration Platform which provides a unified and integrated platform for **Rules, Workflow,** and **Event Processing**. **Drools –** BRMS system with a forward-chaining and backward-chaining inference-based rules engine, allowing fast and reliable evaluation of business rules and complex event processing.

- **Drools –** Allows fast and reliable evaluation of business rules and complex event processing.
    - A rule engine is a fundamental building block to create an expert system which, in artificial intelligence terms is an AI / ES (artificial intelligence / expert system).
    - Drools can reason to a conclusion (infer) beyond what we currently know.
    - An inference model provides a conclusion reached on the basis of evidence and reasoning.
        - We say that "Drools emulates the decision-making ability of a human expert."

2.2.2 Why use a Rule Engine with jBPM for IoT Internet of Things?

Some frequently asked questions:

- When should you use a Rule Engine in jBPM?
- What advantage does Rule Engine and jBPM have over hand-coded "if...then" approaches?
- Why should you use Rules Engine / jBPM instead of an imperative programming language?

2.2.3 Advantages of a Rule Engine

Below summarizes some key business benefits for using BRMS - BPM, a comprehensive platform for business rules management, business resource optimization and Complex Event Processing (CEP).

- **Declarative Programming**

A Rule Engine allows you to define "**What to Do**" and not "**How to do it**."

The key advantage of this point is that using rules can make it easy to express solutions to difficult problems and consequently have those solutions verified. Rules are much easier to read than code.

Rule systems are capable of solving very hard problems, providing an explanation of how the solution was arrived at and why each "decision" along the way was made, not so easy with other Artificial Intelligent (AI) Systems like neural networks or the human brain.

- **Logic and Data Separation**

Your data is in your domain objects; the logic is in the rules. This is fundamentally breaking the OO coupling of data and logic, which can be an advantage or a disadvantage depending on your point of view. The upshot is that the logic can be much easier to maintain when there are changes in the future, as the logic is all laid out in rules.

This can be especially true if the logic is cross-domain or multi-domain logic. Instead of the logic being spread across many domain objects or controllers, it can all be organized in one or more very distinct rules files co-located across IT infrastructure. With IoT devices by default, they are separated in a Hyper Computing environment. This given by architecture design creates the separation of logic and data.

- **Speed and Scalability**

The Rete algorithm, the Leaps algorithm, and their descendants such as Drools' ReteOO provide very efficient ways of matching rule patterns to your domain object data. These are especially efficient when you have datasets that change in small portions as the rule engine can remember past matches. These algorithms are battle-proven.

- **Centralization of Knowledge**

By using rules, you create a repository of knowledge (a knowledge base) which is executable. This means it's a single point of truth, for business policy, for instance. Ideally, rules are so readable that they can also serve as documentation.

- **Tool Integration**

Tools such as Eclipse provide ways to edit and manage rules and get immediate feedback, validation and content assistance. Auditing and debugging tools are also available.

- **Explanation Facility**

Rule systems effectively provide an "explanation facility" by being able to log the decisions made by the rule engine along with why the decisions were made.

- **Understandable Rules**

By creating object models and, optionally, domain-specific languages that model your problem domain, you can set yourself up to write rules that are very close to natural language. They lend themselves to logic that is understandable to possibly nontechnical and domain experts.

2.2.4 Business Rules Engine Methodology

Provides a mechanism to define, deploy, execute, monitor and maintain the variety and complexity of decision logic that is used by operational systems within an organization. There are two methods of execution for a rule system: Forward-Chaining and Backward-Chaining; systems that implement both are called Hybrid Chaining Systems. Drools provide seamless Hybrid chaining for our IoT systems.

Forward Chaining

- Forward Chaining is "data-driven" and thus reactionary, with facts being asserted into working memory, which results in one or more rules being concurrently true and scheduled for execution by the Agenda Engine. In short, we start with a fact, it propagates, and we end in a conclusion.

In Forward Chaining, the system starts from a set of facts, and applies a set of rules, and tries to find a way of using those rules and facts to deduce a conclusion or come up with a suitable course of action.

Usually, when a rule is triggered, it is then fired, which means its conclusion is added to the rules facts database. If the conclusion of the rule that has fired is an action or a recommendation, then the system may cause that action to take place or the recommendation to be made.

Backwards Chaining

- Backward Chaining is "goal-driven," meaning that we start with a conclusion which the engine tries to satisfy. It searches for conclusions that it can satisfy; as sub-goals; these then help satisfy some unknown part of the current goal. It continues this process until either the initial conclusion is proven or there are no more sub-goals.

In backward chaining, we start from a conclusion, which is the hypothesis we wish to prove, and we aim to show how that conclusion can be reached from the rules and facts in the database. The conclusion we are aiming to prove is called a goal, and so reasoning in this way is known as a derivation query or goal-driven reasoning. Prolog is AI-ES, an example of a Backward Chaining engine.

It is difficult to talk specifics without details about each use case, but the general idea is that the more rules and the more complex the rules are, the more benefits the Drools Rules engine gives you. This happens because increasing the number of rules usually has little to negligible impact on the response.

2.2.5 The IoT STREAM Rules System

Most of the rules used by an IoT STREAM System are event declaration. To declare a FACT type as an "event" all that is required is to assign the @role metadata tag to the fact type.

The @role metadata tag accepts two possible values:

- **Fact**: (this is the default) Declares that the type is to be handled as a regular fact. This would be a typical CLOUD type of rule.
- **Event**: Declares that the type is to be handled as an event. Every event has an associated timestamp assigned to it.

By default, the timestamp for a given event is read from the session clock and assigned to the event at the time the event is inserted into the working memory. All facts are static and stored in the IoT BPM system production memory as rules or events.

The facts that the inference engine matches against are kept in the working memory. Using the CEP pattern of processing the event-driven architecture from the transaction database to the JBoss Drools/Fusion Engine (CEP), the facts are inserted into the IoT BPM working memory when the data collection / transaction occurs.

- **Note:** Defining terms is not the goal of this guide and as so, let's adopt a loose definition that, although not formal, allows us to proceed with a common understanding. So, in the scope of this guide: Event is a record of a significant change of state in the application domain at a given point in time. Events are immutable and can be embellished. A transaction is an event.

2.2.6 Event-Driven Architectures

An Event-Driven Architecture (EDA) is a software architecture pattern promoting the production, detection, consumption of and reaction to events. Building applications and systems around an event-driven architecture allow these applications and systems to be constructed in a manner that facilitates more responsiveness because event-driven systems are, by design, more normalized to unpredictable and asynchronous environments.

Event processing is a method of tracking and analyzing (processing) streams of information (data) about things that happen (events) and deriving a conclusion from them. Complex event processing, or CEP, is event processing that combines data from multiple sources to infer events or patterns that suggest more complicated circumstances. The goal of Complex Event Processing is to identify meaningful events (such as opportunities or threats) and respond to them as quickly as possible.

The core benefits of this architecture approach are that it provides loose coupling of the components, (an optional) IoT BPM transaction database or MQTT message queue, and JBoss Drools/Fusion AI-IoTBPM Server system.

A component publishes events about actions that it is executing and transaction database subscribes/listens to these events.

The transaction database subscriber listens for events, stores and inserts the facts into the AI-IoTBPM Server working memory. An orchestration layer then handles the actual inserting into working memory of only events we are interested in analyzing by the rules inference engine.

These events are notices happening across the various layers of the organization. An event may also be defined as a "change of state," when a measurement exceeds a predefined threshold of time, response, or other value. IoT Complex Event Processing analysts give the organizations a new way to analyze patterns in real-time and help the business side communicate better with IT and service departments.

After inserted, new events into Drools Working Memory from the data collection ingestion the Event Stream Processing, or ESP passes-off to our complex event processing (CEP), which takes precedence.

ESP-(Event Stream Processing) is a set of technologies which include event visualization, event databases, event-driven middleware, and our Drools AI event processing language.

2.2.7 CEP - Complex Event Processing

CEP is primarily an event processing concept that deals with the task of processing multiple events with the goal of identifying the meaningful events within the event cloud. CEP employs techniques such as detection of complex patterns of many events, event correlation and abstraction, event hierarchies, and relationships between events such as causality, membership, and timing, and event-driven processes.

CEP deals with the task of processing streams of event data with the goal of identifying the meaningful pattern within those streams, employing techniques such as detection of relationships between multiple events, event correlation, event hierarchies, and other aspects such as causality, consequence, and timing. These are inserted in the Rules Engine and synthetic facts or events that are conclusions of facts. This is the sequencing of synthetic events from actual events/faces and the conclusions drawn.

CEP allows patterns of simple and ordinary events to be considered to infer that a complex event has occurred. Complex event processing evaluates a confluence of events and then takes action. The events (notable or ordinary) may cross event types and occur over a long period of time. Events across multiple Drools BRMS engine sessions. The event correlation may be causal, temporal, or spatial.

CEP requires the employment of sophisticated event interpreters, event pattern definition and matching, and correlation techniques from the Drools BRMS engine. CEP is commonly used to detect and respond to business anomalies, threats, locations, and opportunities and is well suited for our IoTBPM Server and IoT device domain.

2.2.8 Drools Quick Primer

For those who are new to Drools, it is a good idea to present a quick primer overview. This will provide better context and understanding for the next chapters. There are excellent training documents available for both BPM and Drools online.

The following is an introduction to Drools, its elements and their execution semantics. It tries to briefly provide an intuitive understanding of Drools AI power. It does not cover the entire feature set of Drools. For more exhaustive BRMS resources, see the references on the www.iotbpm.com website.

2.2.9 Drools BRMS Elements

Drools allows fast and reliable evaluation of business rules and complex event processing using rule engine. A rule engine is a fundamental building block to create an expert system; an AI / ES System. Drools can reason to a conclusion (infer) beyond what we currently know, and we can use this conclusion in BPM to process to our goal.

Drools uses an inference model provides a conclusion reached on the basis of evidence and reasoning. We say that; Drools emulates the decision-making ability of a human expert. The Drools engine makes it very suitable for writing business logic in a central location. This makes it easier to maintain and usually less like spaghetti code. This makes it easy to separate each achievement as a separate rule, without impacting the flow of the application.

Drools is a **BRMS (Business Rules Management System)**. It is written in Java and is open source. Drools Implements and extends the Rete Algorithm. The Drools Rete implementation is called ReteOO, signifying that Drools has an enhanced and optimized implementation of the Rete algorithm.

Drools is a collection of tools that allow us to separate and reason over logic and data found within business processes. The two important keywords to notice here are **Logic (Rules)** and **Data (Facts)**.

2.2.10 Drools Knowledge Base

In Drools we use Java files to load the Drools and execute the rules environment.

- Drools Rule Engine

Drools is a **Rule Engine** or a **Production Rule System** that uses the rule-based approach to implement and Expert System. Expert Systems are knowledge-based systems that use knowledge representation to process acquired knowledge into a knowledge base that can be used for reasoning.

The brain of a Production Rules System is an **Inference Engine** that can scale to a large number of Rules and Facts. The Inference Engine Matches Facts against Rules.

- Knowledge Base

A Drools Knowledge Base is an interface that manages a collection of rules, processes, and internal types. In Drools, these are commonly referred to as **knowledge definitions** or **knowledge packages**. The Knowledge definitions are rules that can reason over facts (data) in the working memory. The Knowledge Base provides methods for creating knowledge sessions.

- Knowledge Session

The knowledge session is retrieved from the knowledge base. It is the main interface for interacting with the Drools Engine. The knowledge session can be of two types: A Stateless Knowledge Session and a Stateful Knowledge Session.

- Stateless Knowledge Session

Stateless Knowledge Session is a stateless session that forms the simplest use case, not utilizing inference. A stateless session can be called like a function, passing it some data and then receiving some results back. Common examples of a stateless session include:

  - **Validation -** Is this person eligible for door access?
  - **Calculation -** Compute room temperature and humidity sensor change.
  - **Routing and Filtering -** Filter incoming messages, such as emails, into folders, or Send incoming messages to an IoT device destination.

- Stateful Knowledge Session

Stateful sessions are longer lived and allow iterative changes over time. Some common use cases for stateful sessions include:

  - **Monitoring –** Device monitoring and analysis for automatic alerts.
  - **Diagnostics -** Fault finding, medical diagnostics.
  - **Logistics -** Parcel tracking and delivery GPS provisioning and tracking.

- Knowledge Builder

Writing Drools Rules - Rules are pieces of knowledge often expressed as, "*When* some conditions occur, *then* do some tasks."

- **Package**: Every Rule starts with a package name. The package acts as a namespace for Rules.
- **Import statement**: The facts you want to apply the rule on, those facts need to be imported.
- **Rule Definition**: It consists of the Rule Name, the condition, and the Consequence. Drools keywords are **rule, when, then,** and **end**.

The **when** part is the condition in the rule, and the **then** part is the consequence. A rule specifies that **when** a particular set of conditions occur, **then** do what is specified as an action.

- Rule Consequence Keywords

Rule Consequence Keywords are the keywords used in the "**then**" part of the rule.

- **Modify** – The attributes of the fact can be modified in the **then** part of the Rule.
- **Insert** – Based on some condition, if true, one can insert a new fact into the current session of the Rule Engine.
- **Retract** – If a particular condition is true in a Rule and you don't want to act anything else on that fact, you can retract the particular fact from the Rule Engine.

Note: It is considered very bad practice to have a conditional logic (if statements) within a rule consequence. Most of the times, a new rule should be created.

3 IoT Internet of Things Drools-jBPM Architecture

3.1 Drools Fusion Engine Solution Design

3.1.1 Design Use Case Methodology

Event processing use cases share several requirements, and goals with business rules use cases.

These overlaps happen both on the business side and the technical side.

On the Business side:

- Business rules are frequently defined based on the occurrence of scenarios triggered by events. Examples could be:
    - On an algorithmic trading application: Take action if the security price increases X% compared to the day opening price where price increases are usually denoted by events on a stock trade application.
    - On a monitoring application: Take action if the temperature on the server room increases X degrees in Y minutes where sensor readings are usually denoted by events.
- Both business rules and event processing queries change frequently and require immediate response for the business to adapt itself to new market conditions, new regulations, and new enterprise policies.

From a technical perspective:

- Both require seamless integration with the enterprise infrastructure and applications, especially on autonomous governance, including, but not limited to, lifecycle management, auditing, security, etc.
- Both have functional requirements like pattern matching and non-functional requirements like response time and query/rule explanation.

In this context, Drools Fusion is the module responsible for adding event processing capabilities into the drools rules platform.

Supporting complex event processing, though, is much more than simply understanding what an event is. CEP scenarios share several common and distinguishing characteristics:

- Usually required to process huge volumes of events, but only a small percentage of the events are of real interest.
- Events are usually immutable since they are a record of state change.
- Usually, the rules and queries on events must run in reactive modes, i.e., react to the detection of event patterns.
- Usually, there are strong temporal relationships between related events.
- Individual events are usually not important. The system is concerned about patterns of related events and their relationships.
- Usually, the system is required to perform composition and aggregation of events.

Based on this general common characteristic, Drools Fusion defined a set of goals to be achieved to support complex event processing appropriately:

- Support events, with their proper semantics, as first-class citizens.
- Allow detection, correlation, aggregation, and composition of events.
- Support processing of streams of events.
- Support temporal constraints in order to model the temporal relationships between events.
- Support sliding windows of interesting events.
- Support a session scoped unified clock.
- Support the required volumes of events for CEP use cases.
- Support to (re)active rules.
- Support adapters for event input into the engine (pipeline).

This list of goals is based on the requirements not covered by Drools expert itself since in a unified platform, all features of one module are leveraged by the other modules. Drools Fusion is born with enterprise-grade features like pattern matching that is paramount to a CEP product, but that is already provided by Drools expert. In the same way, all features provided by Drools Fusion are leveraged by Drools flow (and vice-versa) making process management aware of event processing and vice-versa.

Additionally, in some scenarios, you have to discard equal objects (objects of the same type and values) when they are inserted into the working memory to avoid data inconsistency and unnecessary activations. The preferred method of discarding duplicated facts on insertion is an insertion – use of a custom classLoader in a knowledgeAgenda method.

We can have a system of streams where the events are transmitted, and these events can be of the same type but have to be processed in different ways without mixing them in their processing. Drools Fusion can handle this scenario by creating entry points that can be used to integrate these streams with the rules patterns to process the events that are going to arrive from these streams.

3.2 Event Semantics

An *event* is a fact that presents a few distinguishing characteristics:

- **Usually immutable:** Since, by the previously discussed definition, events are a record of a state change in the application domain, i.e., a record of something that already happened, and the past cannot be "changed," events are immutable. This constraint is an important requirement for the development of several optimizations and the specification of the event lifecycle. This does not mean that the Java object representing the event object must be immutable. Quite the contrary. The engine does not enforce immutability of the object model because one of the most common use cases for rules is event data enrichment.
  - **Note**: As a best practice, the application is allowed to populate un-populated event attributes (to enrich the event with inferred data), but already populated attributes should never be changed.
- **Strong temporal constraints:** Rules involving events usually require the correlation of multiple events, especially temporal correlations, where events are said to happen at some point in time relative to other events.
- **Managed lifecycle:** Due to their immutable nature and the temporal constraints, events usually will only match other events and facts during a limited window of time, making it possible for the engine to manage the lifecycle of the events automatically. In other words, once an event is inserted into the working memory, it is possible for the engine to find out when an event can no longer match other facts and automatically delete it, releasing its associated memory and resources.
- **Use of sliding windows:** Since all events have timestamps associated to them it is possible to define and use sliding windows over them, allowing the creation of rules on aggregations of values over a period of time. Example: Average of an event value over 60 minutes.

Drools supports the declaration and usage of events with both semantics: **point-in-time** events and **interval-based** events.

  - **Note:** A simplistic way to understand the unification of the semantics is to consider a *point-in-time* event as an *interval-based* event whose *duration is zero*.

3.3 Event Processing Modes

Rules engines, in general, have a well-known way of processing data and rules, and provides the application with the results. Also, there are not many requirements on how facts should be presented to the rules engine especially because, in general, the processing itself is time independent. That is a good assumption for most scenarios but not for all of them. When the requirements include the processing of real time or near real time events, time becomes an important variable of the reasoning process.

The following sections explain the impact of time on rules reasoning and the two modes provided by Drools for the reasoning process.

3.3.1 Cloud Mode

The CLOUD processing mode is the default processing mode. Users of rules engine are familiar with this mode because it behaves in the same way as any pure forward chaining rules engine, including previous versions of Drools.

When running in CLOUD mode, the engine sees all facts in the working memory, does not matter if they are regular facts or events as a whole. There is no notion of flow of time, although events have a timestamp as usual. In other words, although the engine knows that a given event was created, for instance, on January 1st 2018, at 09:35:40.767, it is not possible for the engine to determine how "old" the event is because there is no concept of "now."

In this mode, the engine applies its usual many-to-many pattern matching algorithm using the rules constraints to find the matching tuples, activate, and fire rules as usual.

This mode does not impose any kind of additional requirements on facts. So, for instance:

- There is no notion of time — no requirements clock synchronization.
- There is no requirement for event ordering. The engine looks at the events as an unordered Cloud against which the engine tries to match rules.

On the other hand, since there are no requirements, some benefits are not available either. For instance, in CLOUD mode, it is not possible to use sliding windows because sliding windows are based on the concept of "now" and there is no concept of "now" in CLOUD mode.

Since there is no ordering requirement for events, it is not possible for the engine to determine when events can no longer match and as so, there is no automatic life-cycle management for events. i.e., the application must explicitly delete events when they are no longer necessary, in the same way, the application does with regular facts.

Cloud mode is the default execution mode for Drools but, in any case, like any other configuration in Drools, it is possible to change this behavior either by setting a system property using configuration property files or using the API. The corresponding property is:

```
KieBaseConfiguration config = KieServices.Factory.get().newKieBaseConfiguration();
config.setOption( EventProcessingOption.CLOUD );
```

The equivalent property is:

```
drools.eventProcessingMode = cloud
```

3.3.2 Stream Mode

The STREAM processing mode is the mode of choice when the application needs to process streams of events. It adds a few common requirements to the regular processing but enables a whole lot of features that make stream event processing a lot simpler.

The main requirements to use STREAM mode are:

- Events in each stream must be time-ordered. i.e., inside a given stream event that happened first must be inserted first into the engine.
- The engine forces synchronization between streams through the use of the session clock, so, although the application does not need to enforce time ordering between streams, the use of non-time-synchronized streams may result in some unexpected results.

Given that the above requirements are met, the application may enable the STREAM mode using the following API:

```
KieBaseConfiguration config = KieServices.Factory.get().newKieBaseConfiguration();
config.setOption( EventProcessingOption.STREAM );
```

Alternatively, the equivalent property:

drools.eventProcessingMode = stream

When using the STREAM, the engine knows the concept of the flow of time and the concept of "now," i.e., the engine understands how old events are based on the current timestamp read from the session clock.

This characteristic allows the engine to provide the following additional features to the application:

- Sliding window support
- Automatic event lifecycle management
- Automatic rule delaying when using negative patterns

With temporal reasoning, IoT sensors provide information that Drools AI is acted on immediately.

All these features are explained in the following sections:

3.3.3 Role of Session Clock in Stream mode

When running the engine in CLOUD mode, the session clock is used only to timestamp the arriving events that don't have a previously defined timestamp attribute. Although in STREAM mode, the session clock assumes an even more important role.

In STREAM mode, the session clock is responsible for keeping the current timestamp and based on it, the engine does all the temporal calculations on event's aging, synchronizes streams from multiple sources, schedules future tasks and so on.

The documentation on the session clock, in this section, shows you how to configure and use different session clock implementations.

3.3.4 Negative Patterns in Stream Mode

Negative patterns behave differently in STREAM mode when compared to CLOUD mode. In CLOUD mode, the engine assumes that all facts and events are known in advance (there is no concept of the flow of time) and so, negative patterns are evaluated immediately.

When running in STREAM mode, negative patterns with temporal constraints may require the engine to wait for a time period before activating a rule. The time period is automatically calculated by the engine in a way that the user does not need to use any tricks to achieve the desired result.

For instance:

Example 3.3.4.1. A rule that activates immediately upon matching

```
rule "Sound the alarm"
when
    $f : FireDetected( )
    not( SprinklerActivated( ) )
then
    // sound the alarm
end
```

The above rule has no temporal constraints that would require delaying the rule. Therefore, the rule activates immediately. The following rule, on the other hand, must wait for 10 seconds before activating since it may take up to 10 seconds for the sprinklers to activate:

Example 3.3.4.2. A rule that automatically delays activation due to temporal constraints

```
rule "Sound the alarm"
when
    $f : FireDetected( )
    not( SprinklerActivated( this after[0s,10s] $f ) )
then
    // sound the alarm
end
```

This behavior allows the engine to keep consistency when dealing with negative patterns and temporal constraints at the same time. The above would be the same as writing the rule as below, but does not burden the user to calculate and explicitly write the appropriate duration parameter:

Example 3.3.4.3. Same rule with explicit duration parameter

```
rule "Sound the alarm"
    duration( 10s )
when
    $f : FireDetected( )
    not( SprinklerActivated( this after[0s,10s] $f ) )
then
    // sound the alarm
end
```

The following rule expects every 10 seconds at least one "Heartbeat" event; if not, the rule fires. The special case in this rule is that we use the same type of object in the first pattern and the negative pattern. The negative pattern has the temporal constraint to wait between 0 to 10 seconds before firing, and it excludes the heartbeat bound to $h. Excluding the bound heartbeat is important since the temporal constraint [0s, ...] does not exclude by itself the bound event $h from being matched again, thus preventing the rule to fire.

Example 3.3.4.4. Excluding bound events in negative patterns

```
rule "Sound the alarm"
when
    $h: Heartbeat( ) from entry-point "MonitoringStream"
    not( Heartbeat( this != $h, this after[0s,10s] $h ) from entry-point
"MonitoringStream" )
then
    // Sound the alarm
end
```

3.3.5 Session Clock

Reasoning over time requires a reference clock. To mention one example, if a rule reasons over the average price of a given stock over the last 60 minutes, how does the engine know what stock price changes happened over the last 60 minutes in order to calculate the average? The obvious response is by comparing the timestamp of the events with the "current time." How does the engine know what **time it is now**? Again, obviously, by querying the session clock.

The session clock implements a strategy pattern allowing different types of clocks to be plugged and used by the engine. This is very important because the engine may be running in an element of different scenarios that may require different clock implementations. The following are a few:

- **Rules testing:** Testing always requires a controlled environment, and when the tests include rules with temporal constraints, it is necessary to not only control the input rules and facts but also the flow of time.
- **Regular execution:** Usually when running rules in production, the application requires a real-time clock that allows the rules engine to react immediately to the time progression.
- **Special environments:** Specific environments may have specific requirements on time control. Cluster environments may require clock synchronization through heartbeats, or JEE environments may require the use of an AppServer provided clock, etc.
- **Rules replay or simulation:** To replay scenarios or simulate scenarios, it is necessary that the application also controls the flow of time.

3.3.6 Available Clock Implementations

Drools provides 2 clock implementations out of the box — the default real-time clock, based on the system clock and an optional pseudo clock, controlled by the application.

3.3.7 Real Time Clock

By default, Drools uses a real-time clock implementation that internally uses the system clock to determine the current timestamp.

To explicitly configure the engine to use the real time clock, just set the session configuration parameter to real time:

```
KieSessionConfiguration config = KieServices.Factory.get().newKieSessionConfiguration();
config.setOption( ClockTypeOption.get("realtime") );
```

3.3.8 Pseudo Clock

Drools also offers out of the box implementation of a clock that is controlled by the application that is called Pseudo Clock. This clock is especially useful for unit testing temporal rules since it can be controlled by the application and so the results become deterministic.

To configure the pseudo session clock, do the following:

```
KieSessionConfiguration config = KieServices.Factory.get().newKieSessionConfiguration();
config.setOption( ClockTypeOption.get("pseudo") );
```

An example of how to control the pseudo session clock:

```
KieSessionConfiguration config = KieServices.Factory.get().newKieSessionConfiguration();
conf.setOption( ClockTypeOption.get( "pseudo" ) );
KieSession session = kbase.newKieSession( conf, null );

SessionPseudoClock clock = session.getSessionClock();

// then, while inserting facts, advance the clock as necessary:

FactHandle handle1 = session.insert( tick1 );
clock.advanceTime( 10, TimeUnit.SECONDS );
FactHandle handle2 = session.insert( tick2 );
clock.advanceTime( 30, TimeUnit.SECONDS );
FactHandle handle3 = session.insert( tick3 );
```

3.3.9 Sliding Windows

Sliding windows are a way to scope the events of interest by defining a window that is constantly moving. The two most common types of sliding window implementations are time-based windows and length based windows.

The next sections detail each of them.

- **Important** Sliding Windows are only available when running the engine in STREAM mode. Check the event processing mode section for details on how the STREAM mode works.
- **Important** Sliding windows start to match immediately and defining a sliding window does not imply that the rule has to wait for the sliding window to be "full" in order to match.

  For instance, a rule that calculates the average of an event property on a window: length(10) will start calculating the average immediately, and it will start at 0 (zero) for no-events and will update the average as events arrive one by one.

3.3.10 Sliding Time Windows

Sliding time windows allows you to write rules that will match events occurring in the last X time units.

For instance, if the user wants to consider only the stock ticks that happened in the last 2 minutes, the pattern would look like this:

```
StockTick() over window:time( 2m )
```

Drools uses the "**over**" keyword to associate windows to patterns.

On a more elaborate example, if the user wants to sound an alarm in case the average temperature over the last 10 minutes read from a sensor is above the threshold value, the rule would look like this:

Example 3.3.10.1. Aggregating values over time windows

```
rule "Sound the alarm in case temperature rises above threshold"
when
    TemperatureThreshold( $max : max )
    Number( doubleValue > $max ) from accumulate(
        SensorReading( $temp : temperature ) over window:time( 10m ),
        average( $temp ) )
then
    // sound the alarm
End
```

The engine automatically disregards any SensorReading older than 10 minutes and keeps the calculated average consistent.

- **Important** Please note that time-based windows are considered when calculating the interval an event remains in the working memory before being expired but an event falling off a sliding window does not mean by itself that the event will be discarded from the working memory, as there might be other rules that depend on that event. The engine will discard events only when no other rules depend on that event, and the expiration policy for that event type is fulfilled.

3.3.11 Sliding Length Windows

Sliding length windows work the same way as time windows but consider events based on the order of their insertion into the session instead of flow of time. For instance, if the user wants to consider only the last 10 RHT company stock ticks independent of how old they are, the pattern would look like this:

StockTick(company == "RHT") over window:length(10)

As you can see, the pattern is similar to the one presented in the previous section, but instead of using window:time to define the sliding window, it uses window:length.

Using a similar example to the one in the previous section, if the user wants to sound an alarm in case the average temperature over the last 100 readings from a sensor is above the threshold value, the rule would look like the following:

Example 3.3.11.1. Aggregating values over length windows

```
rule "Sound the alarm in case temperature rises above threshold"
when
    TemperatureThreshold( $max : max )
    Number( doubleValue > $max ) from accumulate(
        SensorReading( $temp : temperature ) over window:length( 100 ),
        average( $temp ) )
then
    // sound the alarm
End
```

The engine only considers the last 100 readings to calculate the average temperature.

- **Important** Please note that falling off a length based window is not criteria for event expiration in the session. The engine disregards events that fall off a window when calculating that window but does not remove the event from the session based on that condition alone as there might be other rules that depend on that event.
- **Important** Please note that length based windows do not define temporal constraints for event expiration from the session, and the engine will not consider them. If events have no other rules defining temporal constraints and no explicit expiration policy, the engine will keep them in the session indefinitely.

3.4 Event Streams Support

Most CEP use cases have to deal with streams of events. The streams can be provided to the application in various forms, from JMS queues to flat text files, from database tables to raw sockets or even through web service calls. In any case, the streams share a common set of characteristics:

- **Events** in the stream are ordered by a timestamp. The timestamp may have different semantics for different streams, but they are always ordered internally.
- **Volumes** of events are usually high.
- **Atomic events** are rarely useful by themselves. Usually, meaning is extracted from the correlation between multiple events from the stream and also from other sources.
- **Streams** may be homogeneous, i.e., contain a single type of events, or heterogeneous, i.e., contain multiple types of events.

Drools generalized the concept of a stream as an "entry-point" into the engine. An entry point for drools is a gate from which facts come. The facts may be regular facts or special facts like events.

In Drools, facts from one entry point (stream) may join with facts from any other entry point or event with facts from the working memory. Although, they never mix, i.e., they never lose the reference to the entry point through which they entered the engine. This is important because one may have the same type of facts coming into the engine through several entry points, but one fact that is inserted into the engine through entry point A will never match a pattern from an entry point B, for example.

3.4.1 Declaring and Using Entry Points

Entry points are declared implicitly in Drools by directly making use of them in the rule. i.e., referencing an entry point in a rule will make the engine, at compile time, identify and create the proper internal structures to support that entry point for this rule.

So, for instance, let's imagine a banking application where transactions are fed into the system coming from streams. One of the streams contains all the transactions executed in ATMs. So, if one of the rules says a withdrawal is authorized if and only if the account balance is over the requested withdrawal amount, the rule would look like:

Example 3.4.1.1. Example of Stream Usage

```
rule "authorize withdrawal"
when
    WithdrawRequest( $ai : accountId, $am : amount ) from entry-point "ATM
Stream"
    CheckingAccount( accountId == $ai, balance > $am )
then
    // authorize withdrawal
End
```

In the previous example, the engine compiler identifies that the pattern is tied to the entry point "ATM Stream" and will both create all the necessary structures for the rule base to support the "ATM Stream" and only matches WithdrawalRequests coming from the "ATM Stream." In the previous example, the rule is also joining the event from the stream with a fact from the main working memory (CheckingAccount).

Now, let's imagine a second rule that states that a fee of $2 must be applied to any account for which a withdrawal request is placed at a bank branch:

Example 3.4.1.2. Using a different Stream

```
rule "apply fee on withdrawal on branches"
when
    WithdrawRequest( $ai : accountId, processed == true ) from entry-point
"Branch Stream"
```

```
        CheckingAccount( accountId == $ai )
    then
        // apply a $2 fee on the account
    End
```

The previous rule will match events of the exact same type as the first rule (WithdrawalRequest) but from two different streams. So, an event inserted into "ATM Stream" will never be evaluated against the pattern on the second rule because the rule states that it is only interested in patterns coming from the "Branch Stream."

So, entry points besides being a proper abstraction for streams are also a way to scope facts in the working memory and a valuable tool for reducing cross products explosions. However, that is a subject for another time.

Inserting events into an entry point is equally simple. Instead of inserting events directly into the working memory, insert them into the entry point, as shown in the example below:

Example 3.8.1.3. Inserting facts into an entry point

```
// create your rulebase and your session as usual
KieSession session = ...
// get a reference to the entry point
EntryPoint atmStream = session.getEntryPoint("ATM Stream" );
// and start inserting your facts into the entry point
atmStream.insert( aWithdrawRequest );
```

The previous example shows how to manually insert facts into a given entry point. Although, usually the application will use one of the many adapters to plug a stream end point like a JMS queue directly into the engine entry point without coding the inserts manually. The Drools pipeline API has several adapters and helpers to do that as well as examples of how to do it.

3.5 Memory Management for Events

- **Important** The automatic memory management for events is only performed when running the engine in STREAM mode. Check the event processing mode section for details on how the STREAM mode works.

One of the benefits of running the engine in STREAM mode is that the engine can detect when an event can no longer match any rule due to its temporal constraints. When that happens, the engine can safely delete the event from the session without side effects and release any resources used by that event.

There are basically 2 ways for the engine to calculate the matching window for a given event:

- Explicitly, using the expiration policy
- Implicitly, analyzing the temporal constraints on events

3.5.1 Explicit expiration offset

The first way of allowing the engine to calculate the window of interest for a given event type is by explicitly setting it. To do that, use the declare statement and define an expiration for the fact type:

Example 3.5.1.1. Explicitly defining an expiration offset of 30 minutes for StockTick events

```
declare StockTick
    @expires( 30m )
End
```

The above example declares an expiration offset of 30 minutes for StockTick events. After that time, assuming no rule still needs the event, the engine will expire and remove the event from the session.

- **Important** An explicit expiration policy for a given event type overrides any inferred expiration offset for that same type.

3.5.2 Inferred expiration offset

Another way for the engine to calculate the expiration offset for a given event is implicit, by analyzing the temporal constraints in the rules. For instance, given the following rule:

Example 3.5.2.1. Example rule with temporal constraints

```
rule "correlate orders"
when
    $bo : BuyOrderEvent( $id : id )
    $ae : AckEvent( id == $id, this after[0,10s] $bo )
then
    // do something
End
```

Analyzing the above rule the engine automatically calculates that whenever a BuyOrderEvent matches, it needs to store it for up to 10 seconds to wait for matching AckEvent's. So, the implicit expiration offset for BuyOrderEvent will be 10 seconds. AckEvent, on the other hand, can only match existing BuyOrderEvent's and so its expiration offset will be zero seconds.

The engine will make this analysis for the whole rulebase and find the offset for every event type.

- **Important** An explicit expiration policy for a given event type overrides any inferred expiration offset for that same type.

3.6 Temporal Reasoning

Temporal reasoning is another requirement of any CEP system. As discussed previously, one of the distinguishing characteristics of events is their strong temporal relationships in **IoT** (Internet of Things) device computing.

With temporal reasoning, IoT device sensors provide information that **AI-BPM** can act on immediately and conditionally, determined by situational awareness. In Drools AI-IoT, judging the impact avoidances of a vehicle and making course adjustments, is an example of Drools AI-IoT temporal reasoning. The distinction is acting on events and not reasoning over data, as in the Cloud.

Temporal reasoning is a type of stream reasoning and is an extensive field of research from its roots on temporal modal logic to its more practical applications in business systems. There are hundreds of papers and thesis written, and approaches are described for several applications. Drools takes a pragmatic and simple approach based on several sources about maintaining knowledge about temporal intervals which make it an elegant solution for **IoT**.

Drools implements the interval-based time event semantics described by Allen and represents point-in-time events as interval-based events with duration 0 (zero). Events or alerts are raised by the Arduino Tron IoT devices and indicate a change in condition, again elegant for a Drools rules AI-IoT solution.

3.6.1 Temporal Situational Awareness (SA)

As distributed IoT sensors and applications become larger and more complex, the simple processing of raw sensor and actuation data streams becomes impractical. Instead, data streams must be fused into tangible facts, information that is combined with the knowledge to perform meaningful operations.

In current IoT systems, sensing and actuation are mostly done at the bare bones data level, whereas many IoT applications demand higher level situationl awareness of - and reasoning about - the systems' states and the physical environment where they operate to perform intelligent decisions.

Situational Awareness (SA) is the perception of environmental elements by our IoT devices and events with respect to time or space, the comprehension of their meaning, and the projection of their status after some variable has changed, such as time, or some other variable, such as a predetermined event.

This is what we have accomplished with our AI-IoT Drools-BPM implementation. Our Arduino Tron IoT devices are judging and making decisions after cognitive situational reasoning.

3.6.2 Complex Event Processing

Event Stream Processing, or ESP, is a set of technologies which includes event visualization, event databases, event-driven middleware, and our event processing language (Drools). After inserting events, the ESP passes-off to our complex event processing (CEP), which takes precedence.

The manipulations of events are described by CEP rules, which are event-condition-actions that combine continuous query primitives with context operators (e.g., temporal, logical, quantifiers) on received events, checking for correlations among these events, and generating complex (or composite) events that summarize the correlation of the input events. Most CEP systems have the concept of Event Processing Agents (EPAs), which are modules that implement event processing workflows.

3.6.3 IoTBPM - Internet of Things Reasoning

The IoT paradigm aims at connecting billions of IoTBPM devices to the internet. This requires suitable architecture and technologies capable of bridging the vast heterogeneity of the devices to provide meaningful services. To enable this seamless integration of devices to provide sophisticated services, the concept of AI reasoning has to address a number of issues with the growing number of devices e.g., scalability, heterogeneity, and reliability.

3.6.4 Temporal Operators

Drools implements all 13 operators defined by Allen and also their logical complement (negation). This section details some of the operators and their parameters used in IoT.

3.6.5 After

The after evaluator correlates two events and matches when the temporal distance from the current event to the event being correlated belongs to the distance range declared for the operator.

Let's look at an example:

```
$eventA : EventA( this after[ 3m30s, 4m ] $eventB )
```

The previous pattern will match if and only if the temporal distance between the time when $eventB finished and the time when $eventA started is between (3 minutes and 30 seconds) and (4 minutes).

3.6.6 Before

The before evaluator correlates two events and matches when the temporal distance from the event being correlated to the current correlated belongs to the distance range declared for the operator.

Let's look at an example:

```
$eventA : EventA( this before[ 3m30s, 4m ] $eventB )
```

The previous pattern will match if and only if the temporal distance between the time when $eventA finished and the time when $eventB started is between (3 minutes and 30 seconds) and (4 minutes).

3.6.7 Coincides

The coincides evaluator correlates two events and matches when both happen at the same time. Optionally, the evaluator accepts thresholds for the distance between events start and finish timestamps.

Let's look at an example:

```
$eventA : EventA( this coincides $eventB )
```

The previous pattern matches if and only if the start timestamps of both $eventA and $eventB are the same AND the end timestamp of both $eventA and $eventB also are the same.

Optionally, this operator accepts one or two parameters. These parameters are the thresholds for the distance between matching timestamps.

- If only one parameter is given, it is used for both start and end timestamps.

  - If two parameters are given, then the first is used as a threshold for the start timestamp and the second one is used as a threshold for the end timestamp.

3.6.8 During

The during evaluator correlates two events and matches when the current event happens during the occurrence of the event being correlated.

Let’s look at an example:

```
$eventA : EventA( this during $eventB )
```

The previous pattern will match if and only if the $eventA starts after $eventB starts and finishes before $eventB finishes.

3.6.9 Finishes

The finishes evaluator correlates two events and matches when the current events start timestamp happens after the correlated events start timestamp, but both end timestamps occur at the same time.

Let’s look at an example:

```
$eventA : EventA( this finishes $eventB )
```

The previous pattern will match if and only if the $eventA starts after $eventB starts and finishes at the same time $eventB finishes.

3.6.10 Includes

The includes evaluator correlates two events and matches when the event being correlated happens during the current event. It is the symmetrical opposite of during evaluator.

Let’s look at an example:

```
$eventA : EventA( this includes $eventB )
```

The previous pattern will match if and only if the $eventB starts after $eventA starts and finishes before $eventA finishes.

3.6.11 Starts

The starts evaluator correlates two events and matches when the current event end timestamp happens before the correlated event's end timestamp, but both start timestamps occur at the same time.

Let’s look at an example:

```
$eventA : EventA( this starts $eventB )
```

The previous pattern will match if and only if the $eventA finishes before $eventB finishes and starts at the same time $eventB starts.

3.6.12 Started By

The started by evaluator correlates two events and matches when the correlating events end timestamp happens before the current events end timestamp, but both start timestamps occur at the same time. Let’s look at an example:

```
$eventA : EventA( this startedby $eventB )
```

The previous pattern matches if and only if the $eventB finishes before $eventA finishes and starts at the same time $eventB starts.

The startedby evaluator accepts one optional parameter. If it is defined, it determines the maximum distance between the start timestamp of both events in order for the operator to match. Example:

```
$eventA : EventA( this starts[ 5s ] $eventB )
```

4 Rules Writing Performance Memory and Testing

4.1 Drools Writing Rules Best Practice

Example 4.1.1. Drools Rules Practices

Drools performance is based on how Rete trees and nodes are created, how Drools indexes them, and why an increasing number of objects in Drools hardly affects the total time taken to execute it. Rules written intelligently can drastically reduce the number of nodes in the Rete tree, thus, further increasing memory and impacting the performance.

Some Drools Rules writing best practice and executing the rules as fast as possible are:

- Put the most restricting condition on the top
- The conditions that you feel should be on the highest priority - put them on the top
- The then conditions that you use should be diligently prepared
- Use the same order of conditions across your rules
- Do not use eval unless you have to
- Put evals at the bottom of your conditions
- Do not use if statements inside consequences
- Using shortcuts for Booleans cause JIT errors on Drools 5.4 so do use them as House (windowOpen == true) not House (windowOpen)
- Avoid using salience. In most cases, it leads to maintenance issues
- Plan using an Eclipse-Drool UI to create good rules
- Never attempt using if-statements inside the then part
- Use shortcuts for Boolean because they often cause errors
- Always follow the pattern of RWTE - i.e., 1. RULE 2. WHEN 3. THEN 4. END
- Avoid calling heavy java objects. It causes troubles in most cases
- Try to integrate the rules with custom classes
  rather than predefined sets to be used for your operations
- The condition that you are using when the part should be interlinked and not null
  (i.e., the condition should be linked to some values which have existence)
- Always use the Drools generalized concept of a stream as an "entry point" into the engine
  An entry point is for drools a gate from which facts come, the source where there were inserted
  The facts may be regular facts or special facts like events
- Use the import statements properly. Import statements work like import statements in Java. You need to specify the fully qualified paths and type names for any objects you want to use in the rules. Drools automatically imports classes from the Java package of the same name, and also from the package java.lang

- **Package**: Every rule starts with a package name. The package acts as a namespace for rules. Rule names within a package must be unique. Packages in rules are similar to packages in Java. They serve the same purpose.

- **Import statement**: Whatever facts you want to apply the rule on, those facts need to be imported into your application.

- **Rule definition**: Consists of the rule name, the condition, and the consequence. Drools keywords are **rule, when, then,** and **end**. The **when** part is the condition in both the rules and the **then** part is the consequence.

- **Load the rules once; batch inserts all facts and fire once:** Letting the engine work out the most optimal way to execute is more efficient and easier to maintain from the development perspective. It also prevents the rules from being loaded/parsed multiple times before getting the desired outcome.

- **Don't overload rules:** Each rule should describe one and only one scenario. The engine will optimize shared conditions: i.e., rules that share conditions of a fact (in the same order) share their Rete nodes.

4.2 Drools Performance and Memory Internals

Drools Rules "lots of objects" and "complex logic in sequence"

JBoss Drools - The keywords for performance impact are "lots of objects" and "complex logic in sequence. " JBoss Drools uses Rete's algorithm to execute rules. Rete's algorithm is an efficient pattern matching algorithm. In the later versions of Drools, the Phreak engine optimization has been added, and this has greatly improved performance, making the engine lazy evaluation instead of eggier evaluation.

JBoss Drools has its own implementation of Rete's algorithm. A rule in Drools is represented by a Rete tree. A Rete tree consists of nodes. These nodes are mostly conditional evaluations. Everything in a Drools rule is represented by a Rete tree node. Apart from conditional nodes, the Rete tree also consists of AND nodes, OR nodes, and start/end nodes (and a few other types of nodes as well).

The main part of the algorithm is the creation of the Rete tree. Whenever a fact (object) is inserted in the Drools engine, a Rete tree is created. The creation of this Rete tree is done by some intelligent algorithm. The Rete tree, once created, executes in almost no time over the facts (objects) and gives the result. Most of the time is spent creating the Rete tree rather than executing it.

This is something that you can also experience during debugging. Whenever a fact is inserted into the Drools session, it takes a bit of time. However, the firing of rules is almost instantaneous.

So, the way this Rete algorithm has been implemented inside Drools accounts for its efficiency. However, what will happen to the JBoss Drools Efficient Rete algorithm when "LOTS OF OBJECTS" will come into the picture?

The answer lies in two techniques that Drools uses to store nodes: Node Sharing and Node Indexing.

Drools caches nodes while building Rete’s tree, which is known as node sharing. Whenever a new node is created, it’s checked whether there is an equivalent node already present in the cache. If an equivalent node is found in the cache, the cached node is used instead, and the new node is discarded. This makes Drools more memory efficient.

Drools keeps a hash table of the object properties and Rete tree nodes. It is known as node indexing and is used to avoid evaluating the same conditions multiple times. Node indexing speeds up the propagation of nodes in the Rete network which accounts for high performance of the Drools engine.

The in-depth analysis of Rete tree creation and node propagation in the Rete network tells that the way rules are written might also affect the performance as it directly impacts the propagation in Rete tree. Rules written intelligently can drastically reduce the number of nodes in the Rete tree, thus, further increasing the performance.

So, these techniques help Drools to process a large number of objects efficiently. The conclusion is - increasing the number of objects in Drools hardly affects the total time taken to execute it.

4.3 Drools Testing Rules Methodology

Drools Rules tested as code or data?

Should business rules be embedded as a part of the application code that doesn't change very often or are rules more like your application data (for example, pricing lists), which you expect to change on almost a daily basis?

This comes up often. Depending on our answer, we will deploy our rules very differently.

The answer, somewhat confusing, is that rules are both.

- Rules are as powerful as the normal code and should be treated in the same way. (For example, before deployment, any changes should be thoroughly tested.)
- Rules are as easy to change as data because first, they live outside the “normal” application and second, rule engines are expressly designed to easily allow changes to the business rules.

So, we are really saying “**both,”** depending on the impact.

4.3.1 Testing of the Rules

Testing of the rules is achieved by means of unit testing each rule or a couple of rules representing a certain scenario. Each such test case is covered by several unit tests attempting to evaluate a different situation. The tests comprising these unit tests are grouped in compliance with the separation of the rules into .drl files. In other words, a test may add into knowledge builder only the .drl file holding the declarations and the .drl file representing the tested logic.

Each test is composed of unit tests and a before and after method. The before method is responsible for composing the session before each unit test and the after method for correct disposal of it. The constructing of the session is performed in a similar way as in the init phase of the application with the exception of resources, channels, and clock.

Only the resources necessary for the scenario are used for creating knowledge packages. The mock objects are substituted for actual channels as they require the application server to operate.

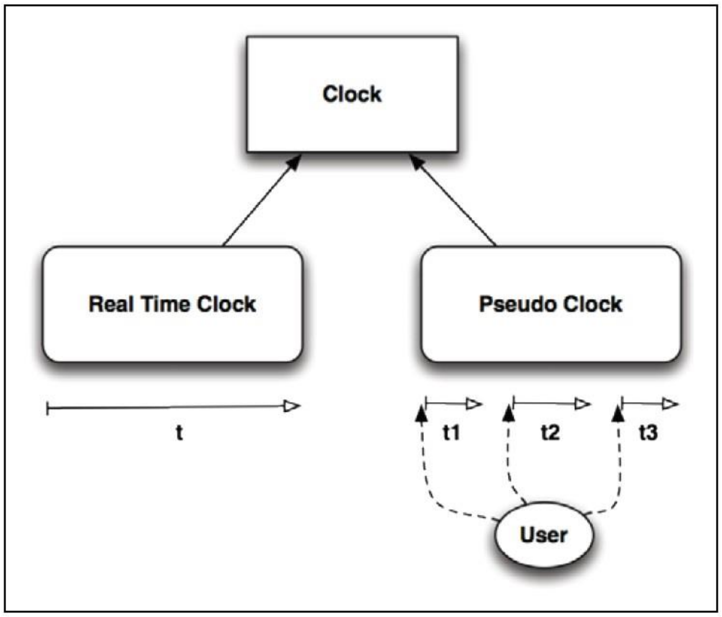


Figure 4.3.1.1: The two Drools clock implementations.

Since many of the rules employ the temporal reasoning aspects, the clock implementation in the tests must be different. Drools provides two clock implementations demonstrated in Figure 13.1, where the Real Time Clock applies the JVM clock, and Pseudo Clock enables the application to control the flow of time in all different ways. In order for the real-time clock to be the default option, the setting of the pseudo-clock has to be configured while creating the session.

4.3.2 Unit Testing Rules

An important point to note is that you should carry out unit testing in the rules that you write. It is manual unit testing, but we still checked that your blocks of rules produced the outcome that you expected. All we're talking about here is automating the process.

Unit testing also has the advantage of documenting the code because it gives a working example of how to call the rules. It also makes your rules and codes more reusable. You've just proved (in your unit test) that you can call your code on a standalone basis, which is an important first step for somebody else to be able to use it again in the future.

Run tests with a small amount of data on all of your rules, i.e., with a minimal number of facts in the rule session and test the results that the particular rule was fired as expected. For the sample data, use static data and define minimal test data for each rule test.

5 Arduino Tron AI-IoT Artificial Intelligent Internet of Things

5.1 Internet of Things Artificial Intelligent Architecture

5.1.1 Internet of Things Artificial Intelligence Conclusion

- An inference model provides a conclusion reached on the basis of evidence and reasoning.

5.1.2 When IoT meets Artificial Intelligence

In the Internet of Things (IoT), as more and more devices and pieces of software interconnect, a great necessity arises for the systems that allow complex situations to be detected in a simple collaborative way by people and devices and be able to react quickly upon detection of these situations.

Internet of Things (IoT) provides lots of telemetry and sensor data; however, the data points by themselves do not provide value unless they can be annualized and turned into actionable, contextualized information. Big data and data visualization techniques allow us to gain new insights by batch-processing and off-line analysis. Real-time sensor data analysis and decision-making are often done manually but to make it scalable, it is preferably automated. Artificial Intelligence (AI) provides us with the framework and tools to go beyond trivial real-time decision and automation use cases for IoT.

With AI-IoTBPM (Artificial Intelligence – Internet of Things), it is important to understand the difference and relationship between big data and real-time event reasoning, known as temporal reasoning. Big data analysis of sensor data retrieved from many IoTBPM devices provides statistical information on particular components and data points. Decision making will allow deciding whether there is a need for maintenance of one particular component. With temporal reasoning, IoT sensors provide information that Drools AI is acted on immediately. For example; in Drools AI-IoT, judging impact avoidance of a vehicle and making course adjustments is an example of AI temporal reasoning or a rational agent.

The AI-IoTBPM rational agent is a central concept in artificial intelligence. An agent is something that perceives its environment through sensors and acts upon that environment via actuators, servos or motors. For example, a robot may rely on cameras as sensors and act on its environment via motors.

A rational agent is an agent that acts, and that does 'the right thing.' The right thing depends on the performance criterion defined for an agent, but also on an agent's prior knowledge of the environment, the sequence of observations the agent has made in the past and the choice of actions that an agent can perform. The AI BRMS Drools itself is the heart of the agent that computes, and reasons based on the available data and its knowledge of the IoT sensors on the environment.

5.1.3 AI Patterns in STREAM or CLOUD Mode

Drools AI-IoTBPM patterns behave differently in STREAM mode when compared to CLOUD mode. In CLOUD mode, the engine assumes that all facts and events are known in advance (there is no concept of the flow of time) therefore, AI patterns are evaluated immediately.

When running in STREAM mode, patterns with temporal constraints require the engine to wait for an event to occur before activating the rule. The time period is automatically calculated by the engine and events which are considered immutable state changes, the results of which will fire a rule.

Real-time sensor data analysis and decision-making are often done manually but to make it scalable, it is preferably automated. AI provides us with the framework and tools to go beyond trivial real-time decision and automation use cases for IoT.

Executive Order Corp. has developed both a CLOUD-based and a STREAM-based architecture that observes its environment via IoT defined sensors and acts on its environment through AI BRMS Drools-jBPM software, arriving at conclusions reached on the basis of evidence and reasoning.

Executive Order Corp. has developed an IoTBPM platform that uses concepts of AI and applied those to the use case of smarter decision making in IoT.

"If a machine thinks, then a machine can do." – Steven Woodward

"It's not you interacting with the machine; it's the machine interacting with you." – Steven Woodward

5.2 AI-IoTBPM Artificial Intelligent Reasoning

5.2.1 IoTBPM AI-Artificial Intelligent Smart Things Automation

The Internet of Things (IoT) refers to a network of connected devices collecting and exchanging data and processing it over the internet. IoT promises to provide "smart" environments (homes, cities, hospitals, schools, stores, offices, etc.) and smart products (cars, trucks, airplanes, trains, buildings, devices, etc.). While this data is useful, there is still "a disconnect" in integrating these IoT devices with mission-critical business processes and corporate cloud data awareness.

The task of moving IoT devices beyond "connected" to "smart" is daunting. Moving beyond collecting IoT data and transitioning, to leveraging this new wealth of IoT data, to improving the smart decision-making process is the key to automation. Artificial Intelligence (**AI**) will help these IoT devices, environments and products to self-monitor, self-diagnose and eventually, self-direct. This is what we said in the opening of this book, "If a machine thinks, then a machine can do."

However, one of the key concepts in enabling this transition from connected to smart is the ability to perform **AI Analytics**. The traditional analytic models of pulling all data into a centralized source such as a data warehouse or analytic sandbox is going to be less useful. We are not trying just to analyze complex IoT data; we are trying to make "smart decisions" and take actions base on our AI Analytics of IoT devices.

IoT Definition *– IoT is the integration of computer-based systems into our physical-world.*

Our world is increasingly linked through the number of already connected IoT devices. IoT components are equipped with sensors and actuators that enable sensing, acting, collecting and exchanging data via various communication networks including the internet. These IoT devices such as wearable, GPS, smartphones, connected cars, vending machines, smart homes, and automated offices are used in areas such as supply chain management, intelligent transport systems, robotics, and remote healthcare. Businesses can rapidly gain a competitive edge by using the information and functionalities of IoT devices (sensors and actuators). So, a business process uses IoT information to incorporate real-world data, to make informed decisions, optimize their execution, and adapt itself to context changes.

Also, the increase in processing power of IoT devices enables them to take part in the execution of the business logic. This way IoT devices can aggregate and filter data and make decisions locally by executing parts of the business logic whenever central control is not required, reducing both the amount of exchanged data and of central processing involvement.

The power of the IoT device increases greatly when a business process (jBPM) can use them to provide information about our real-world as well as execute IoT device actions as part of our business process. The jBPM-BPMN modular allows us to define both the business processes and IoT devices behavior at the same time using one (BPM) diagram. In our examples, we will be adding Drools-jBPM to IoT devices. Making **"Things Smart"** is the application of AI to IoT platform via Drools Rules Inference Reasoning, jBPM, and Expert Systems (ES) Architecture.

With the use of AI Drools-jBPM analysis and reasoning in IoT devices, we can orchestrate dissimilar devices that normally have no awareness of each other. This creates opportunities for direct integration of computer-based into the physical-world that has never been available before. This results in improved efficiency, accuracy, and economic benefits by increased automation - reduced intervention. This IoT orchestration of IoT devices gives us the ability for action after our AI decision.

5.2.2 IoT Human Interface or Human-Task Node

Another important aspect of IoT AI Drools-jBPM is the human interface and system interaction. While some of the work performed in an IoT jBPM process can be executed automatically, some tasks may need to be executed by human actors. jBPM supports a special human task node inside processes for modeling this interaction with human users. This human task node allows process designers to define the properties related to the task that the human actor needs to execute.

Consider that our IoT device may raise an alarm, alert to a device or process fault condition or action. The jBPM supports the use of human tasks inside processes using a user task node. A user task node represents an atomic task that needs to be executed by a human actor (in this example maybe answer the alarm). As far as the jBPM engine is concerned, human tasks are similar to any other external service that needs to be invoked and are implemented as a domain-specific service.

Because a human task is an example of such a domain-specific service, the process itself contains a high-level, abstract description of the human task that needs to be executed, and a work item handler is responsible for binding this abstract tasks to a specific implementation of the human task.

The IoT AI Drools-jBPM project provides a default implementation of a human task service based on the WS-Human-Task specification. If you have a requirement to integrate an existing human task service you can use this service. It manages the life cycle of the tasks (creation, claiming, completion, etc.) and stores the state of the tasks. To have human actors participate in your IoT processes do the following:

1. Include human task nodes inside your process to model the interaction with human actors
2. Integrate a task management component (e.g., the WS-Human-Task provided by jBPM)
3. Have the end-users interact with a human task IoTBPM client interface.

How can IoT benefit from jBPM? Let us consider a complex system with multiple components interacting within a smart environment, being aware of the components' locations, movements, and interactions. Such a system can be a smart factory with autonomous robots, a retirement home with connected residents, or on a larger scale, a smart city. While the parties in the system can track the movements of each component and also relate multiple components' behaviors to each other, they do not know the components' agendas. Often their interactions are based on habits, (i.e., routine low-level processes), which represent recurring tasks. Some of these routines are more time and cost critical than others, some may be dangerous or endanger others, and some may just be inefficient or superfluous. Knowing their agendas, their goals, and their procedures can enable a better basis for planning, execution, and safety.

How can jBPM benefit from IoT? Let us consider a complex process with multiple parties interacting in the context of a business transaction. Such a process can be, for example, a procurement process, where goods are ordered, delivered, stored, and paid for. While the system can track each automatically executed activity on its own, it relies on messages from other parties and manually entered data in the case of manual activities. If this data is not entered or entered incorrectly, discrepancies between the digital (i.e., computerized representation) process and the real-world execution of the process occur. Similar concerns hold, if the process participants do not obey the digital process under certain circumstances (e.g., an emergency in healthcare) or have not entered the data, though in the real-world process the respective activity was already executed.

Such scenarios might be better manageable when closely linking the digital process with the physical-world as enabled by the integration of IoT and jBPM (e.g., the completion of manual activities can be made observable through the usage of appropriate sensors). IoT can complete jBPM with continuous data sensing and physical actuation for improved decision making. Decisions in processes require relevant information as a basis for making meaningful decisions. Data from IoT, such as events provided through in-memory databases or **CEP** (Complex Event Processing), can be useful in this context.

5.2.3 AI-IoT Drools-jBPM Artificial Intelligent Reasoning Makes IoT Smart

AI-IoT is a mix of Business Processes (BPM) with Business Rules Drools (Reasoning), to define advanced and complex scenarios. Also, Drools Rules Engine adds the ability of temporal reasoning, allowing business processes to be monitored, improved, and cover business scenarios that require temporal inferences. Event stream processing focused on the capabilities of processing streams of events in (near) real time, while the main focus of CEP (Complex Event Processing) was on the correlation and composition of atomic events into complex (compound) events. IoTBPM for CEP is primarily an event processing concept that deals with the task of processing multiple events with the goal of identifying the meaningful events within the IoT event cloud. CEP in IoTBPM employs techniques for detection of complex patterns of many events, event correlation and abstraction, and event hierarchies.

5.3 IoTBPM Server Drools-jBPM Installation Configure

5.3.1 IoTBPM Server Configure Eclipse, Drools-jBPM and BPMN2

This quick guide provides installation and configuration instructions for the IoTBPM Server program, Eclipse IDE, Eclipse Plugins, Drools-jBPM and BPMN2 on your Windows computer.

- Download and install the "Eclipse IDE for Java Developers."
- Use the Eclipse feature to add new software, which is available on the Eclipse menu "Help -> Install New Software." Select the "Add" option and these two packages:

  Drools + jBPM Update Site 7.20.0
  http://downloads.jboss.org/jbpm/release/7.20.0.Final/updatesite/

  BPMN2-Modeler 1.4.2
  http://download.eclipse.org/bpmn2-modeler/updates/oxygen/1.4.2/

The Executive Order Corp. Arduino Tron ESP8266 MQTT telemetry transport Machine-to-Machine (M2M) IoTBPM Server software and Arduino Tron MQTT AI-IoTBPM Client using Arduino Tron AI-IoTBPM Drools-jBPM latest software can be download from the Github website.

- Download Arduino Tron from https://github.com/eodas/IoTBPMServer
- GIT the IoTBPM Server from the source code repository and import the existing maven project.

Once installed, you can configure the different runtime properties in the *iotbpm.properties* file. It is easier to uncomment the runtime environment that you intend to execute. Note: If you have an Arduino Tron Server, update the arduinoAgent IP address; otherwise, just leave this committed out.

In Eclipse, run the class IoTBPM as a Java application. This is the main class for AI-IoTBPM Server Drools-jBPM expert system.

EOSPY Client – To install the EOSPY client application on your phone, download the EOSPY application from the Google App Store. To start the EOSPY client, click on the Eagle icon on your phone. The EOSPY client screen will appear. You can also download the EOSPY TI-SensorTag client version.

To configure a new EOSPY client, you will need to enter the IoTBPM Server address, domain name, or IP address into the server address. Next, add this device in the IoTBPM Server by entering the device name and the device identifier. Swipe the service status **ON,** and YOU'RE DONE. The device will appear on the IoTBPM Server map the next time the EOSPY client sends GPS position information.

GPS Tracking Devices – Many companies make various off-the-shelf GPS tracking devices. Configuring these devices will vary a little from vendors. First, add the new device with a unique identifier into the IotBPM Server – Executive Order Sensor Processor System Server. Next, configure your device to use the appropriate IoTBPM Server IP address and port number. If the device fails to report, check the IP address and device ID.

Device Unique Identifier

If you don't know your device identifier, you can configure your device first and look at the server log file. When the server receives a message from an unknown device, it writes a record containing the unique identifier of a new device. Look for records like "Unknown device – 123456789012345"; "Unknown device" 123456789012345 is your new device identifier.

Address and Port

To select the correct port, find your device in the list of supported devices. The port column of the corresponding row contains default port numbers for your device. If you want to use variations from the default ports, you can change them in the configuration file.

EOSPY system supports more than 90 GPS communication protocols and more than 800 models of GPS tracking devices from popular GPS vendors. Review the list of supported devices for information about your GPS tracking device on www.eospy.com

5.4 Arduino Tron AI-IoT Drools-jBPM Installation Configure

5.4.1 Arduino Tron IoT AI-Artificial Intelligent Streamline Automation

Arduino Tron allows you to send IoT sensor data information directly to the AI-IoT Drools-jBPM Expert System from any Arduino device. This provides a very lite streamline IoT to Drools-jBPM (Business Process Management) application process with or without GPS LAT/LON, speed, bearing, and altitude positioning information. This makes for a very efficient IoT Drools-jBPM Expert System.

If you have fixed IoT sensors monitoring conditions/environments or IoT devices that operate equipment, then Arduino Tron is all that is needed. The IoT devices send their WiFi information directly to the Arduino Tron Agent. The Arduino Tron Agent then processes the IoT data through the Drools Inference Engine and the jBPM process model.

With Arduino Tron jBPM-BPMN modular, it allows us to define both the business processes and IoT devices behavior at the same time using one diagram. With Arduino Tron adding just Drools-jBPM to IoT, we make the IoTBPM devices "smart. " This technique also helps Drools to process a large number of objects more efficiently and is designed for high volumes of data.

Arduino Tron allows you to send IoT sensor data and information directly to the AI-IoTBPM Drools-jBPM Expert System from the Arduino device. This provides a very lite streamline IoT to Drools-jBPM (Business Process Management) application process with sensor and GPS positioning information.

This quick section helps you install and configure the Arduino Tron – Executive Order Sensor Processor System components.

Executive Order Arduino Tron has several components to provide a complete AI-IoTBPM system:

- **Arduino Tron AI-IoTBPM (Java):** Internet of Things Drools-jBPM Expert System.
- **Arduino Tron Sensor (Arduino):** Application to send sensor MQTT Telemetry Transport.
- **Arduino Tron Agent (Arduino):** Software to control external Arduino connected devices.
- **Arduino Tron Server (Arduino):** Web Server Interface for Controlling all IoT devices.

You can have an unlimited number and combination of Arduino Tron IoTBPM Devices and/or EOSPY GPS Client tracking devices in use with Arduino Tron AI-IoT.

(Optionally, you can download the EOSPY server from our website www.eospy.com and Download the EOSPY GPS client from the Google Store, standard or TI-SensorTag version.)

5.4.2 Arduino Tron AI-IoT Configure Eclipse, Drools-jBPM and BPMN2

This quick guide provides installation and configuration instructions for the Arduino Tron AI-IoTBPM program, Eclipse IDE, Eclipse Plugins, Drools-jBPM and BPMN2 on your windows computer.

- Download and install the "Eclipse IDE for Java Developers."
- Use the Eclipse feature to add new software, which is available on the Eclipse menu "Help -> Install New Software." Select the "Add" option and these two packages:
  Drools + jBPM Update Site 7.20.0
  http://downloads.jboss.org/jbpm/release/7.20.0.Final/updatesite/
  BPMN2-Modeler 1.5.0
  http://download.eclipse.org/bpmn2-modeler/updates/oxygen/1.5.0/

The Executive Order Corp Arduino Tron ESP8266 MQTT telemetry transport Machine-to-Machine (M2M) IoTBPM software and Arduino Tron MQTT AI-IoTBPM Client using EOSpy AI-IoTBPM Drools-jBPM latest software can be download from the Github website.

- Download Arduino Tron from https://github.com/eodas/IoTBPM
- GIT the Arduino Tron AI-IoTBPM from the source code repository and import existing maven project into the Eclipse IDE project.

Once installed, you can configure the different runtime properties in the *arduinotron.properties* file. It is easier to uncomment the runtime environment that you intend to execute. In Eclipse, run the class ArduinoTron as a Java application. This is the main class in ArduinoTron AI-IoT Drools-jBPM system.

5.5 Arduino Tron IoTBPM Sensor Software Suite

5.5.1 Arduino Tron Sensor Send MQTT Telemetry Transport and Web Serer

The power of the IoT (Internet of Things) device increases sigificantly when business process (jBPM) can use them to provide information about our real-world as well as execute IoT devices as part of our business process. The jBPM-BPMN modular allows us to define both the business processes and IoT devices behavior at the same time using one diagram. With Arduino Tron adding Drools and jBPM to IoT, we make the IoT devices "smart." Moving beyond just collecting IoT data and transitioning, to leveraging the new wealth of IoT data, to improving the SMART decision making is the key. The Executive Order Arduino Tron AI-IoTBPM will help these IoT devices, environments, and products to self-monitor, self-diagnose and eventually, self-direct.

Arduino Tron allows you to send IoT sensor data and information directly to the AI-IoTBPM Drools-jBPM Expert System from the Arduino device. This provides a very lite streamline IoT to Drools-jBPM (Business Process Management) application process with sensor and GPS positioning information.

The Arduino Tron MQTT sensor software allows you to interface and sends MQTT Telemetry Transport Information from your external connected Arduino devices to the Arduino Tron AI-IoT Drools-jBPM server. The AI-IoTBPM Arduino Tron Agent software uses a WiFi wireless transceiver interface to stream telemetry information to the Arduino Tron Agent for any control module sensors or remote control connected Arduino device.

5.5.2 Arduino Tron IoT Sensor

Arduino Tron Sensor – To install the Arduino Tron application on your Arduino device, download the Arduino Tron Sensor application from GIT. Update the with WiFi network values for network SSID (name) and network password. Update the Arduino Tron Agent IP address and unique unit ID values.

Also, you may use a DHT11 digital temperature/humidity sensor and other sensors (see the Arduino Tron Sensor sketch for more details and information).

5.5.3 Arduino Tron IoT Sensor Emulator

Arduino Tron IoT Sensor Emulation - Prototype an Arduino-based low-power WiFi Internet of Things (IoT) device with built-in sensors emulation that could be used to deliver sensor data from any location in the world, and potentially control connected devices such as thermostats, lights, door locks, and other automation products.

Use a serial monitor to emulate sensor input values to Arduino Tron. This allows you to prototype the final IoT device before custom-designing the PCB (printed circuit board).

5.5.4 Arduino Tron Agent Control External Arduino Connected Devices

The Arduino Tron Agent AI-IoTBPM software interface allows you to send commands with the Arduino Tron AI-IoTBPM Server software to control external Arduino connected devices. The IoTBPM Arduino Agent software uses a WiFi wireless transceiver interface to control and interact with module sensors and remote control devices.

You can control any device from the Arduino Tron Agent software or stream any interface over the WiFi internet. With the Arduino Tron Agent software, you can automatically turn on lights, appliances, cameras, lock/unlock doors, and open doors from the Drools-jBPM expert system processing.

5.5.5 Arduino Tron Web Server Interface for Controlling IoT Devices

Arduino Tron Web Server provides an IoT dashboard, management, and control for remote management of your Internet-Enabled IoT devices. The Arduino Tron Web Server operates completely on an Arduino ESP-01, integrated low power 32-bit CPU RISC processor, on-chip memory and integrated WiFi 802.11 b/g/n. The ESP-01 Arduino Tron smart microdevice is about the size of a quarter.

The beauty of the Arduino Tron Web Server is that the IoT device technology becomes interactive with humans through a simple Web Server. With the Arduino Tron Web Server getting your IoT project working in the cloud is a fast-easy solution. We discuss the Arduino Tron Web Server in Chapter 6.

5.6 IoTBPM Server Drools-jBPM Application Examples

5.6.1 IoT Control AI-IoT Drools-jBPM Artificial Intelligent Smart Automation

In the IoT Control jBPM example, we demonstrate how to invoke **business rules** from within our application and how to execute our jBPM processes and how to handle the interactions between **process** and **rules** using an IoT sensor device and manual switch.

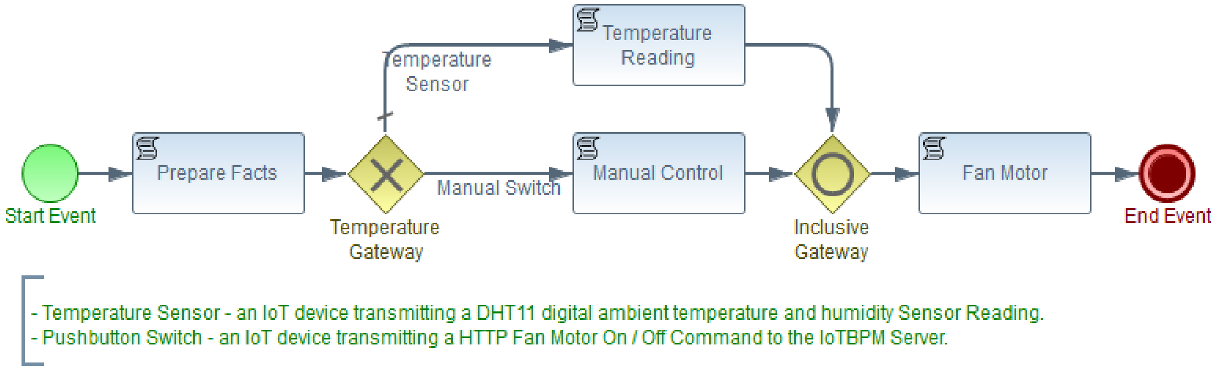
Business processes and rules are two core concepts which are defined as follows:

- **Business processes:** Represents what the business does.
- **Business rules:** Represents decisions that the business makes.

Although processes and rules are two different things, there is a clear advantage if your end users are allowed to combine processes and rules. This means for example:

- Rules can define which processes to invoke
- Rules can specify decisions in that process
- Rules can augment (or even override) the behavior specified in the process (for example to handle exception cases)
- Assignment rules can be used to assign actors to (human) tasks
- Rules can be used to dynamically alter the behavior of your process

Example 5.6.1.1. AI-IoT Control Fan Motor jBPM



- Temperature Sensor - an IoT device transmitting a DHT11 digital ambient temperature and humidity Sensor Reading.
- Pushbutton Switch - an IoT device transmitting a HTTP Fan Motor On / Off Command to the IoTBPM Server.

In this example we use IoTBPM Server Drools-jBPM to solve a knowledge and notification problem. We have an electronic cabinet with a temperature sensor IoT device installed to notify us of the cabinet temperature and indicate if the cabinet fan motor is on or not. Our rule is, if the temperature sensor is *below* the turn fan motor on reading, we can use the manual control to turn the fan motor on and off. If more temperature sensor is *above* the turn fan motor on reading, then we cannot use the manual control switch to turn the fan motor off. This would be a safety override to prevent someone from switching the fan motor off when it is needed.

With this use of IoT Drools-jBPM analysis and reasoning in IoT devices, we were able to orchestrate dissimilar devices (IoT devices, temperature sensors, switches, and motors) that normally have no awareness of each other. This creates opportunities for direct integration of computer-based into the physical-world that has never been available before.

This IoT orchestration of devices gives us the ability for action after our AI decision. Also, we can use IoT devices to update mechanical fan motor operations, giving them a new awareness of the situation and environment. Of course, we could improve on our IoTBPM Server Drools-jBPM example application by adding outside IoT temperature sensors and predicting the need to turn the fan motor on. This would become a preventive cooling system that could prolong the life of our electronic equipment.

Example 5.6.1.2. Manual Control Switch Fan Motor Off Rule

```
rule "rule for manual control switch fan motor off"
```

```
  when
       $event : DeviceEvent( $eventId : id, $eventId == "100222", $switchEvent
       : event, $switchEvent == "off" )
  then
       com.iotbpm.model.StateList.getInstance().putState("switch", "off");
end
```

This rule example demonstrates the use of a pushbutton signal as a switch. Each time you press the IoT button, the state is turned on (if it's off) or turned off (if on). This also demonstrates how we can repurpose a legacy signaling device to operate with new or different behavior in IoT.

Example 5.6.1.3. Declare Rule to Fire when Temperature is Over 73

```
rule "temperature reading too warm"
  when
       $event : DeviceEvent( $eventId : id, $eventId == "100111", $temp :
temp, $temp > "73" )
  then
       com.iotbpm.model.StateList.getInstance().putState("dht11", "on");
       com.iotbpm.model.StateList.getInstance().putState("motor", "on");
end
```

The IoT Control rules are all relatively similar, for the DHT11 temperature sensor and manual control switch updating the control status each time that Rule is fired and delegating the action to the jBPM process. By delegating important decisions to be taken into your rules system, the **business processes** become much more resilient to change. This IoTBPM example provides a "**smart**" environment for the equipment room cooling fan by adding AI-IoT Drools-jBPM artificial intelligent and smart automation.

Example 5.6.1.4. Declare Override Rule to Switch Fan Motor Off

```
rule "rule to overrides manual control switch fan motor off"
  when
       $temp : String() from
com.iotbpm.model.StateList.getInstance().getState("dht11")
       $switch : String() from
com.iotbpm.model.StateList.getInstance().getState("switch")
       eval(($temp == "off") || ($temp == ""))
       eval($switch == "off")
  then
       com.iotbpm.model.StateList.getInstance().putState("motor", "off");
end
```

In the IoT Control jBPM example, we define an IoT temperature sensor device transmitting a DHT11 digital ambient temperature and humidity sensor reading and an IoT a pushbutton switch device transmitting an HTTP fan motor on or off command to the IoTBPM Server. The IoTBPM device ID is used in the device ID gateway to differentiate between the DHT11 temperature reader and the manual switch commands. This gives us the ability to have special operations in our business process for each of our IoT devices (the IoT temperature sensor and IoT pushbutton switch).

With this use of IoT Drools-jBPM analysis and reasoning in IoT devices, we were able to orchestrate dissimilar devices (temperature sensor, manual switch, and fan motor) that normally have no knowledge or awareness of each other. This demonstrates opportunities for direct integration of computer-based systems into the physical-world that has never been available before by repurposing legacy systems or mechanical systems instead of replacing the hardware or physical machine. This also allows you to update legacy mechanical system with new safety features and device status notifications that were never built into the original systems.

5.6.2 Illuminance IoTBPM Server Drools Agent Automation

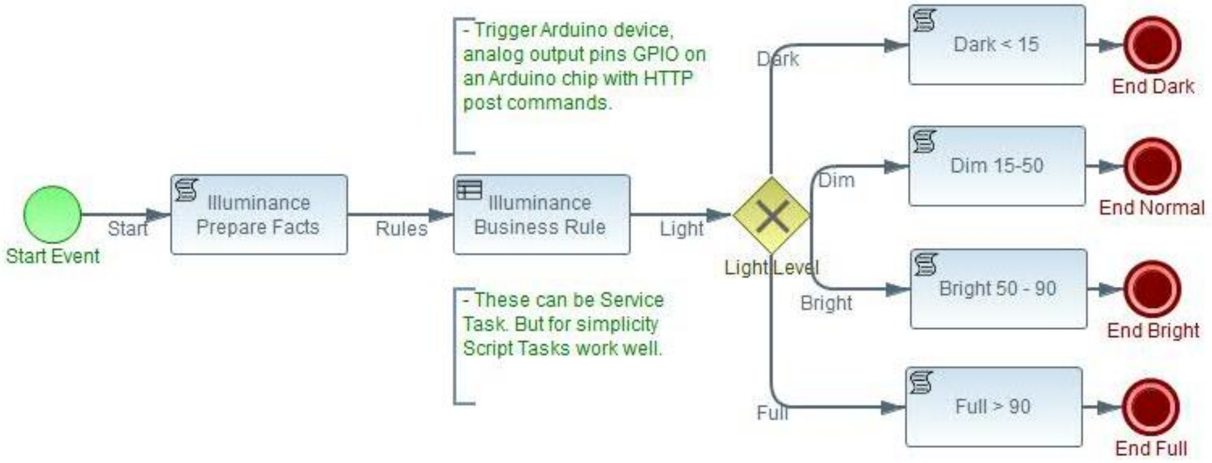
In the previous example, we looked at the use of IoTBPM Drools-jBPM Server to solve a knowledge and notification safety problems. Now, let's look at an end-to-end IoT device to device AI-Drools-jBPM example. In the illuminance example, we will demonstrate how to invoke Drools **business rules** from within our jBPM executing processes and how to handle the interactions between **process** and **rules**.

We will use the IoT device client to stream remote sensor information to the AI-IoTBPM Server-jBPM application. Then we will use the AI-IoT Drools-jBPM on this IoT information stream to decide what behavior and business process functions to execute. Finally, we will execute the AI-IoTBPM Drools-jBPM decision using the AI-IoT Device Arduino Tron Agent software interface, which allows us to send commands with the AI-IoT Device software to control external Arduino connected IoT devices.

The AI-IoTBPM Arduino Tron Agent software uses a WiFi wireless transceiver interface to control and interact with module sensors and remote control devices. You can control any device from the IoTBPM Server or stream any interface HTTP command over the internet. With the AI-IoTBPM Server software, you can automatically turn on lights, appliances, cameras, and open doors from the AI-IoTBPM Server Drools-jBPM Expert System HTTP command processing model.

This gives us a complete IoT, to Drools-jBPM, to IoT, end-to-end process example. By streaming IoT information directly into our business process management application we can modify IoT behavior and functionality without having to physically change our IoT devices.

Example 5.6.2.1 Illuminance AI-IoTBPM Server Drools-jBPM


- Trigger Arduino device, analog output pins GPIO on an Arduino chip with HTTP post commands.
Start Event
Start
Illuminance Prepare Facts
Rules
Illuminance Business Rule
Light
Light Level
Dark
Dim
Bright
Full
Dark < 15
Dim 15-50
Bright 50 - 90
Full > 90
End Dark
End Normal
End Bright
End Full
- These can be Service Task. But for simplicity Script Tasks work well.

The EOSpy AI-IoTBPM Arduino Tron Agent NodeMCU can receive action commands from the IotBPM Server via HTTP URL post for 7 Seg display, relay, buzzer, infrared emitter, LED, motor actuators, gate access, auditable speech, alarms, or any other device via an HTTP URL post command. You can use a command like the following to send actions to the EOSpy AI-IoTBPM Arduino Tron Agent:

```
com.arduinotron.server.AgentConnect.getInstance().sendPost("/LED3=ON");
```

In this illuminance example, we will use AI-IoTBPM Drools-jBPM Server to manage the light level in an office. Our IoTBPM sensor will use a photoelectric sensor to set the light field to the illuminance level. If it's too dark, we want to turn on the light. If it's too bright, we want to draw the blinds. If the light level is less than 15, we will send a signal to the lamp switch to turn on. If the light level is greater than 90, we will send a signal to the curtain motor to close the blinds.

With this use of IoT Drools-jBPM analysis and reasoning in IoTBPM devices, we are able to orchestrate dissimilar devices that normally have no awareness of each other. This creates opportunities for direct integration of computer-based into the physical-world and allows us the ability to control IoTBPM behavior outside of the device. This allows devices to be deployed and modified or updated at any time.

After our IoTBPM illuminance light level is read, we use the illuminance business rule group to call and execute our rules for this process. A business rule task provides a mechanism for the process to provide input to a business rules engine and to get the output of calculations that the business rules engine might provide. This rule flow group identifies the IoTBPM device that is reporting the light level.

Example 5.6.2.2. Illuminance Rule Flow Group

```
//declare any rules to fire in the jBPM AI_IoT.Illuminance process
rule "Rule for device ID: 100111"
      ruleflow-group "Illuminance Rule Flow"
      when
            $event : ServerEvent( $eventId : id, $eventId == "100111" )
      then
            System.out.println("ruleflow-group device ID: 100111 " +
$event.getEvent() + " - " + $event.getName());
end
```

In the rules process class we execute a fileAllRules() prior to executing startProcess() for the jBPM.

```
// go! - fire all rules
long noOfRulesFired = this.kSession.fireAllRules();

// Start the process with knowledge session
instance = kSession.startProcess(processID, params);
```

This allows us to do some AI reasoning (discovery) prior to executing our business process. In this example, this allows us to evaluate the IoTBPM event in the updated UI display with the information.

Example 5.6.2.3. IoTBPM Evaluate Event Rule

```
//declare rules to fire when fileAllRules() is executed
rule "Key Press 1 Rule"
    when
        $event : ServerEvent( $eventId : id, event == "keyPress_1" )
    then
      System.out.println("Rule Key Press 1: " + $event.getEvent() + " - " +
$event.getName() + " - " + " Light Value: " + $event.getLight());
end
```

Set this URL address in the *iotbpm.properties* file *arduinoAgent=http://10.0.0.2* to configure the connection to the EOSpy AI-IoTBPM Arduino Tron Agent.

We then use a command like the following to send actions to the EOSpy AI-IoTBPM Arduino Tron Agent for the Dark Script Task, where we will send a signal to the lamp switch to turn on, and the Bright Script Task, where we will send a signal to the curtain motor to close the blinds.

```
com.arduinotron.server.AgentConnect.getInstance().sendPost("/LED3=ON");
```

The EOSpy AI-IoTBPM Arduino Tron Agent NodeMCU can receive action commands from EOSpy via HTTP URL post for 7 seg display, relay, buzzer, infrared emitter, LED, motor actuators, gate access, auditable speech, alarms, or any other device via an HTTP URL post command.

We can use a relay in this Arduino circuit to switch an electronic device on and off. A relay is an electromagnetic switch, which is controlled by a small current, and used to switch ON and OFF relatively a larger current. A relay is a good example of controlling the devices using a smaller Arduino device.

Additionally, we could also call a RESTful Java API client with Java build-in IoTBPM Server HTTP library. This RESTful services could record telemetry information from our IoT device sensors into an enterprise database server or verify a door entry code. Any enterprise RESTful Java API client can now be used with you IoT devices with the IotBPM Server application.

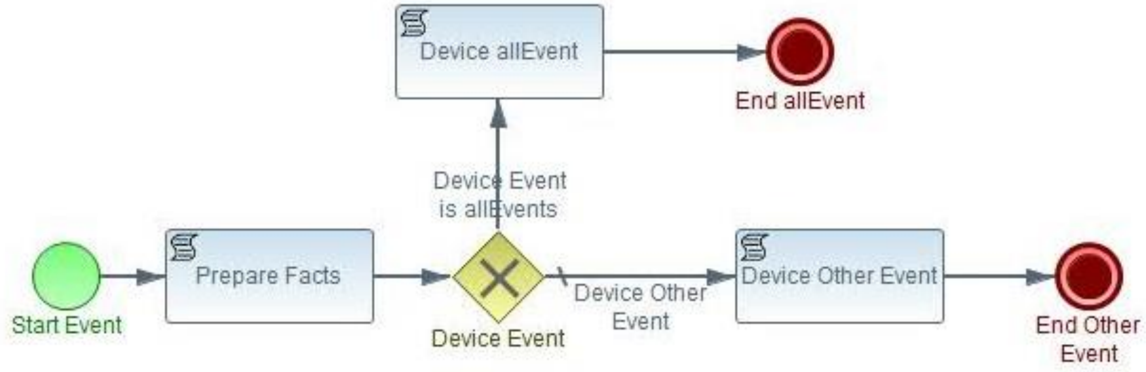
5.6.3 Gyroscope IoTBPM Server Drools-jBPM Motion Sensors

Depending on the device, several sensors can provide information that lets you monitor the motion of the device. The sensors usually include an accelerometer (accel), gyroscope (gyro) and magnetometer (magnet) sensors. Motion sensors are useful for monitoring device movement such as tilt, shake, rotation, or swing. There are also additional sensors such as gravity, linear acceleration, rotation vector, significant motion, step counter, and step detector sensors.

By processing our IoT information/data through the jBPM business process management application we can modify IoT behavior and functionality, without having to physically change our IoT devices.

In the gyroscope example, we will examine the IoT device magnetometer (magnet) sensors to determine if the IoT device is upside-down. This would be important information for a vehicle, shipping container, medical supplies, or any other item that could be damaged or dangerous if turned over.

Example 5.6.3.1 Gyroscope Process AI-IoTBPM Drools-jBPM



In the jBPM process, the following we look at the IoT device event that raised our notification. This could be a significant IoT reported event like airbags deployed, fire alarm, or hazardous materials.

The gyroscope rules will examine the IoT device magnetometer (magnet) sensors and determine if the device is ‘face up’ or ‘face down.’

Example 5.6.3.2 Gyroscope IoTBPM Evaluate Event Rule

```
//declare rules to fire in for zBias
rule "Device Gyroscope Positive zBias"
      when
   $event : ServerEvent( $eventId : id, $magnet_z : magnet_z, $magnet_z > 0 )
      $device : Devices( $deviceId : id,  $eventId == id )
      then
             $device.action = $event.getEvent();
             $device.status = "Face Up";
             modify( $device );
end
```

You can try this example and, of course, any of the others by installing the EOSPY Client Android on your phone. To install the EOSPY Client application on your phone, download the EOSPY application from the Google App Store. To start the EOSPY Client, click on the Eagle icon on your phone. The EOSPY Client screen will appear. You can also download the EOSPY TI-SensorTag Client version.

Then you can turn your phone over and watch the state change in the Gyroscope Rules example.

Also, with the IoT client device, you can send custom textMessage, getTextMessage() and alarm, getAlarm() information to the EOSpy AI-IoT Drools-jBPM application for evaluation in the Drools-jBPM.

5.6.4 GPS Position IoTBPM Server Drools-jBPM GPS Automation

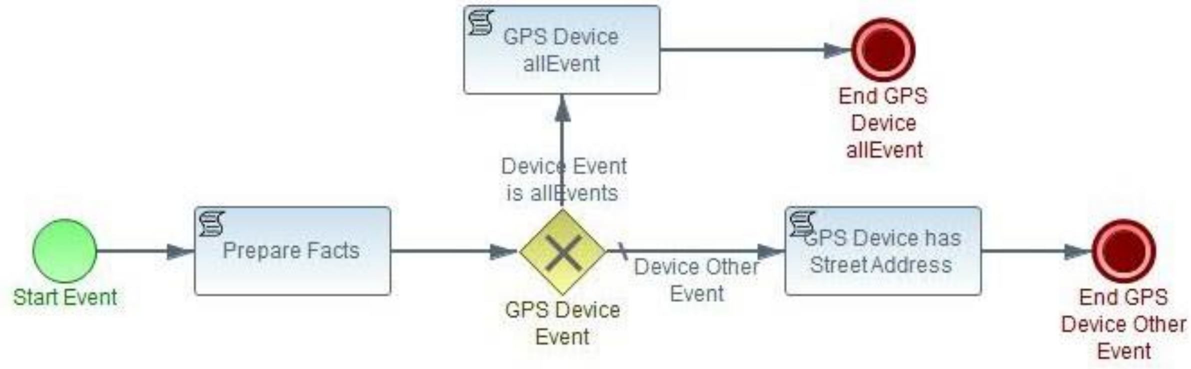
There are many IoT GPS options to choose from with EOSpy. First, you don't have to use a GPS module. If you have a fixed IoT device, consider hardcoding your GPS LAT/LON position. Then, if you have IoT device monitoring, for instance, a door open, you could pin that device location to a building map. This would give you a visual indicator of the location of which door was open on your building blueprint.

Look at the EOSPY AI-IoTBPM Arduino Tron Sensor sketch for an example of how you can find LAT/LON from an address and hardcoding a GPS LAT/LON position-location to the Arduino Tron application.

```
//Update these with LAT/LON GPS position values
//You can find LAT/LON from an address
https://www.latlong.net/convert-address-to-lat-long.html
String lat = "38.888160"; // position LAT
String lon = "-77.019868"; // position LON
```

The Arduino device has many options for GPS modules. The Arduino has methods to read information like date, time, location and satellites in view from the standard NMEA data streams. This NMEA data stream gives you the LAT/LON position to set in your LAT/LON data packet.

Example 5.6.4.1 GPS Position AI-IoTBPM Drools-jBPM



In the GPS position example, we can examine the IoT device event that transmitted the GPS position. This could be a significant IoT event reported from the device, such as motor started, airbags deployed, overheat, outside Geofence or over-speed.

Example 5.6.4.2.GPS Position Speed Over 60 Rule

```
//declare rule to fire when Device speed is over 60
rule "GPS Device Speed Alert"
        when
          $event : ServerEvent( $eventId : id, $lat : lat, $lon : lon,
          $altitude : altitude, $speed : speed, $speed > "60" )
        $device : Devices( $deviceId : id,  $eventId == id )
        then
                $device.action = "Alert";
                $device.status = $event.getEvent();
                com.arduinotron.ui.MainWindow.getInstance().log($event.getName()
+ " GPS Speed Alert lat:" + $lat + " lon:" + $lon + " speed:" + $speed);
                modify( $device )
end
```

This GPS position rule fires when the GPS device reported speed is greater than 60. The rule reports the position and speed that fired the rule. Depending on the GPS devices, the module can report additional information that is transmitted along with the LAT/LON data packet. This information may be different depending on the device type, environment, and conditions.

- **GPS Device Module Environment and Condition Rules**

There are additional rules that respond to the GPS device, if the IoT device sends text messages or if the device sent an alarm.

Example 5.6.4.3.GPS Device Send Alarm

```
//declare rule to fire when GPS Device sends Alarm message
rule "GPS Device Sent Alarm"
        when
          $event : ServerEvent( $eventId : id, $alarm : alarm, $alarm != null )
        $device : Devices( $deviceId : id,  $eventId == id )
        then
                com.arduinotron.ui.MainWindow.getInstance().log($event.getName()
+ " Sent Alarm: " + $alarm);
end
```

In addition, there are rules for processing a GPS device text message; the GPS device fixes information and GPS device address location. All of the information and messages are transmitted along in the LAT/LON data packet from the IoT device.

In the GPS position example, we can see the rules for GPS device distance, GPS device fix information, and GPS device address location being fired. If the GPS module transmitted additional information in the &alarm= or &textMessage= fields, then those rules would fire.

The jBPM Script Task node; GPS Device has street address runs a conditional Java script task.

```
log("GPS Device " +  kcontext.getVariable("name") + " event TYPE KEYPRESS 2 (2.0)");
```

A script task is executed by our business process engine. The modeler or implementer defines a script in a language that the engine can interpret. When the task is ready to start, the engine will execute the script. When the script is completed, the task will also be completed. Script tasks can run action tasks and very complex evaluations and are convenient for combining data.

- **GPS Tracking Devices**

Many companies make various off-the-shelf GPS Tracking devices. Configuring these devices will vary a little from vendors. First, add the new device with a unique identifier into the EOSPY – Executive Order Sensor Processor System Server. Next, configure your device to use the appropriate EOSPY Server IP address and port number. If the device fails to report, check the IP Address and Device ID.

Device Unique Identifier - For most devices, you should use an IMEI (International Mobile Equipment Identity) number as a unique identifier. However, some devices have vendor-specific unique identifiers; for example, TK-103 devices use 12-digit identifier.

If you don’t know your device identifier, you can configure your device first and look at the server log file. When the server receives a message from an unknown device it writes a record containing a unique identifier of a new device. Look for records like “Unknown device – 123456789012345”; “Unknown device” 123456789012345 is your new device identifier.

Address and Port - To select the correct port, find your device in the list of supported devices. The port column of the corresponding row contains default port numbers for your device. If you want to use variations from the default ports, you can change them in the configuration file.

EOSPY supports more than 90 GPS communication protocols and more than 800 models of GPS tracking devices from popular GPS vendors. Review the list of supported devices for information about your GPS Tracking Device.

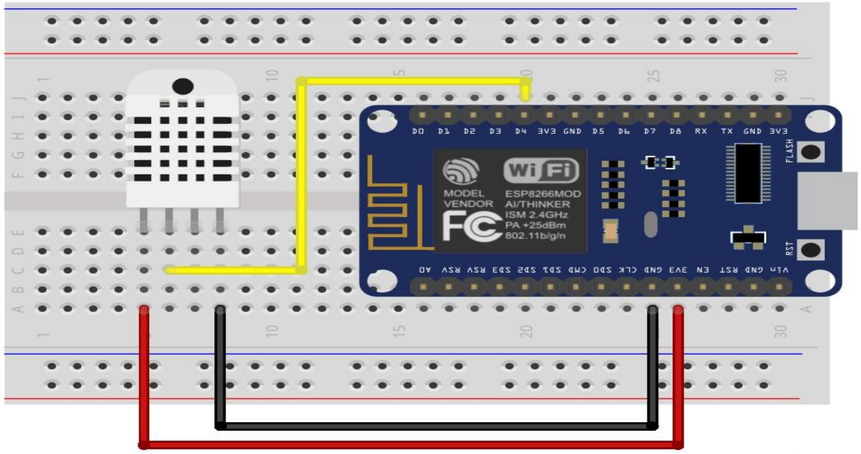
EOSpy live map server will display additional sensors data that you need to automate your processes and gain visibility into your business operations. This additional data could be ambient temperature, IR object temperature, humidity sensor, pressure sensor, ambient light, accelerometer, gyroscope, magnetometer, digital microphone, magnetic sensor, and magnetometer.

5.6.5 Environment IoTBPM Server Drools-jBPM Sensor Automation

In the previous examples, we looked at the use of AI-IoTBPM Drools-jBPM to solve a knowledge and notification problem and looked at an end-to-end AI-IoT Drools-jBPM example. We have demonstrated how to invoke Drools **business rules** from within our jBPM executing processes and how to handle the interactions between **process** and **rules**. These applications work with both EOSPY Android and the EOSPY Arduino Tron application installed on your Arduino IoT Device. Now let's look at a board build Arduino Tron IoTBPM reasoning device. This Arduino Tron device is an Industrial Internet of Things (IIoT) or Machine to Machine (M2M) device that uses sensors in the build with the Arduino device.

We will use the Arduino ESP8266 ESP-16S WiFi serial transceiver module with 4MB flash. Arduino provides a very inexpensive **IoTBPM MQTT** telemetry transport platform. Download the EOSPY Arduino Tron sensor application from GIT. Update the WiFi network values for network SSID (name) and network password. Update the EOSPY server IP address and unique unit ID values and add in EOSPY server. Also, we will use a DHT11 digital temperature and humidity sensor. See the Arduino Tron sensor sketch for more details and information.

Example 5.6.5.1 Environment Reasoning DHT11 Board Schematic


D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
FLASH
Wi Fi
MODEL
VENDOR
ESP8266MOD
AI/THINKER
ISM 2.4GHz
PA +25dBm
802.11b/g/n
RST
A0 RSV RSV SD3 SD2 SD1 CMD SD0 CLK GND 3V3 EN RST GND Vin
1
5
10
15
20
25
30
A B C D E F G H I J

This is the schematic you need to wire the DHT sensor to your ESP8266 board. We will use the EOSpy IoT client to stream remote Arduino ESP8266 DHT11/DHT22 temperature and humidity sensor information to the AI-IoTBPM Drools-jBPM application. Then we will use the AI-IoTBPM Drools-jBPM on this IoT BPM information stream to decide what behavior and business process functions to execute.

All the fields you have provided values for so far are for WiFi communications; the remaining fields are used in the Arduino Tron sensor application. To use a DHT11/DHT12 digital temperature and humidity sensor, sees the Arduino Tron sensor sketch for more details and information.

```
const bool readPushButton = true;
                    // Values for the digitalRead value from gpiox button
const bool readDHT11Temp = true;
             // Values for the DHT11 digital temperature/humidity sensor
```

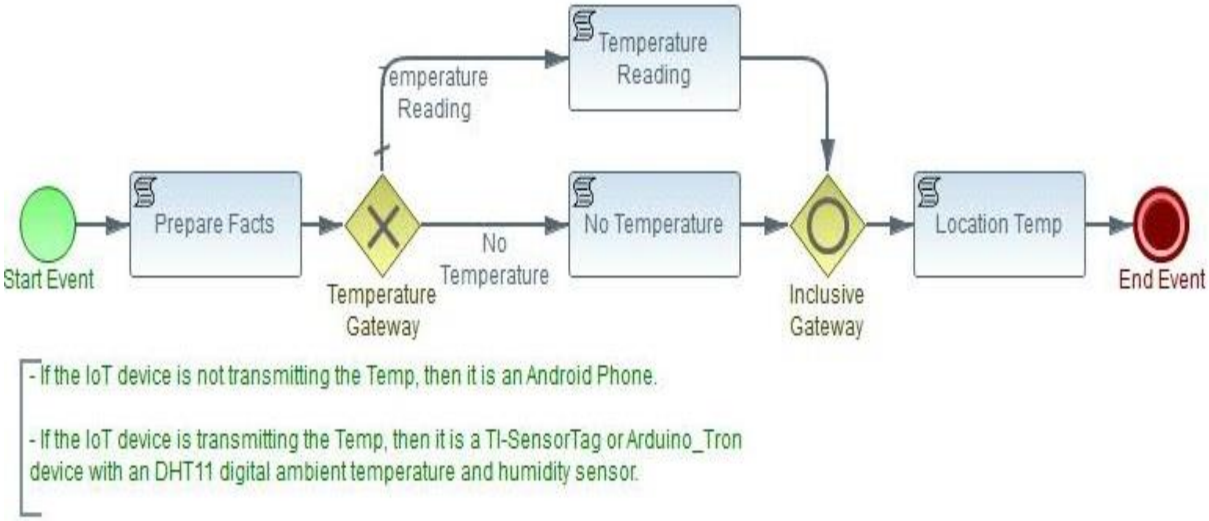
To use a DHT11 digital temperature and humidity sensor, install the SimpleDHT library and uncommit the lines for dht11 in the Arduino Tron sensor sketch for more details and information. Change the readDHT11Temp value to true. Also, you can change the reporting time by modifying the timeCounter that calls the Arduino Tron sensor sketch function for reading the DHT11.

```
void readdht11(); // read DHT11 digital temperature and humidity sensor
```

Remember to uncommit all the lines for the dht11 in the Arduino Tron sensor sketch.

```
if ((err = dht11.read(pinDHT11, &temperature, & humidity, NULL)) !=
SimpleDHTErrSuccess) { <-- uncommit for dht11
```

Example 5.6.5.2 Environment Reasoning AI-IoTBPM Server Drools-jBPM



- If the IoT device is not transmitting the Temp, then it is an Android Phone.

- If the IoT device is transmitting the Temp, then it is a TI-SensorTag or Arduino_Tron device with an DHT11 digital ambient temperature and humidity sensor.

In the environment reasoning example, the following jBPM process we examine the IoTBPM device temperature event that is transmitted by the Arduino device. Also, we evaluate any significant IoTBPM reported temperature or humidity event such as too warm or too cold condition at our location.

Example 5.6.5.3. Environment Reasoning Raise alarm - Too Warm at Location

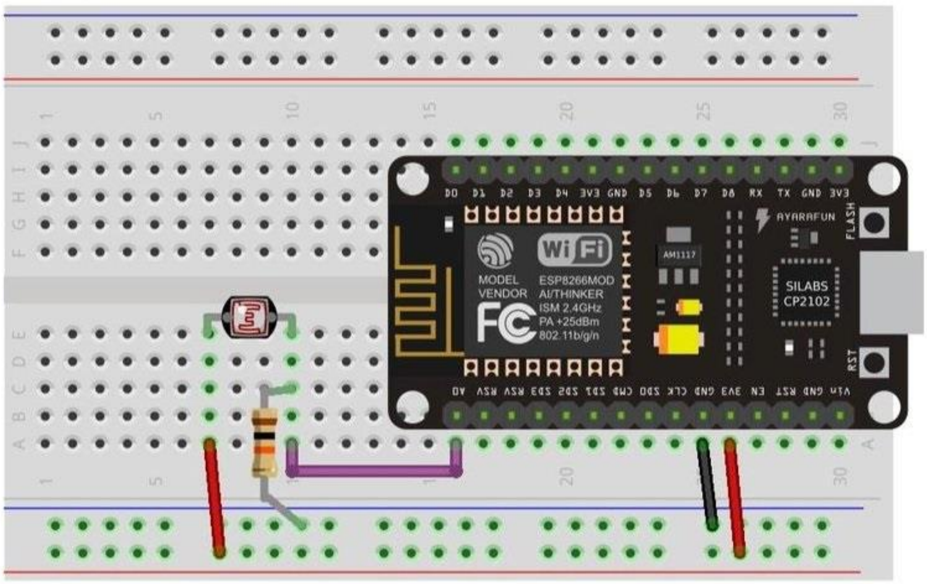
```
//declare rule to fire when Temp is over 75
rule "Raise alarm - Too Warm at Location"
       when
         $event : ServerEvent( $eventId : id, $temp : temp, $temp > "75" )
       $device : Devices( $deviceId : id,  $eventId == id )
       then
              $device.action = "Alert";
              $device.status = "Temp " + $temp;
              com.arduinotron.ui.MainWindow.getInstance().log(">>Raise Temp
Alarm " + $event.getName() + " Temperature:" + $temp);
              modify( $device );
end
```

This environment reasoning rule fires when the DHT11 device reports a temperature if greater than 75. A GPS device module can be added to our environment and condition rules; the rule then reports the position and temperature when the rule is fired. Depending on the GPS devices, the module can report additional information that is transmitted along with the LAT/LON data packet. There are additional rules that respond to the GPS device if the device sends a text message or if the device sent an alarm.

Example 5.6.5.4.Enviorment Reasoning Device Send Text Message

```
//declare rule to fire when GPS Device sends Alarm message
rule "IoTBPM Device Sent Text Message"
       when
         $event : ServerEvent( $eventId : id, $textMessage : textMessage,
$textMessage != null )
       $device : Devices( $deviceId : id,  $eventId == id )
       then
              com.arduinotron.ui.MainWindow.getInstance().log($event.getName()
+ " Text Message: " + $textMessage);
end
```

Example 5.6.5.5 Environment Reasoning LDR Board Schematic


D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
FLASH
AYARAFUN
Wi Fi
MODEL VENDOR
ESP8266MOD AI/THINKER ISM 2.4GHz PA +25dBm 802.11b/g/n
AM1117
SILABS CP2102
RST
A0 RSV RSV SD3 SD2 SD1 CMD SD0 CLK GND 3V3 EN RST GND Vin
1 5 10 15 20 25 30
J I H G F E D C B A

In addition, there are rules for processing an IoT device text message such as the IoT device fix information, and IoT device address location with the current temperature at this address location.

IoT devices can be accessed anywhere in the world via the internet and exchange all types of sensor, environment and conditional situationall data. In this example, we are exploiting only one sensor located somewhere else in the world. The NodeMCU ESP8266 microcontroller has hundreds of various sensors available for the Arduino boards. Most of these sensors are very inexpensive and work well together.

A partial list of the type of sensors available for the Arduino boards include: Soil module, Infrared sensor receiver module, Laser head sensor module, Temperature and humidity sensor module, Infrared emission sensor module, 5V relay module, Gyro Module, Finger detect heartbeat module, Microphone sensitivity sensor module, Metal touch sensor module, Flame sensor module, 3-color LED module, Hunt sensor module, Linear magnetic Hall sensors, Rotary encoder modules, Active buzzer module, Magic Light Cup modules, Small passive buzzer module, Digital temperature sensor module, Tilt switch module, Analogy Holzer magnetic sensor, Ultrasonic module, Mercury opening module, Hall magnetic sensor module, RGB LED SMD module, Mini Reed module, Bicolor LED common cathode module 3MM, Smart car avoid obstacle sensor infrared sensor photoelectric switch, Key switch module, Photo-resistor module, Breadboard power module, Tilt sensor module, Temperature sensor module, Vibration switch module, Microphone sound sensor module, Large reed module, Two-color LED module, Optical breaking module, Temperature sensor module, MP1584EN buck module, SD card reader module, PS2 Joystick game controller module, Automatically flashing LED module, DS1302 clock module, RFID Reader and Sensor, Proximity Sensor, Ultrasonic distance measurement module, Water level module and more.

The NodeMCU ESP8266 microcontroller is a very powerful and economic platform with many additional environmental and conditional sensors available. The NodeMCU ESP8266 can also interface with your existing device circuit boards. This gives you the ability to add WiFi to an existing circuit board.

The attractiveness lies within the fact that this inexpensive NodeMCU ESP8266 microcontroller board can be added to an existing device and programmed via the Arduino IDE to add new functionality like WiFi to an existing system: (i.e., alarm system, fire monitoring, asset monitoring, and building access). Also, the data from this device can be stored in the Arduino memory for later import into our application for processing. This type of store and forward connectivity is useful when the sensors are out of WiFi range.

6 IoTBPM Server Drools-jBPM IoT Automation

6.1 IoTBPM Server IoT Automation Project

In the previous sections we discussed the integration of Drools and BPM into our IoT projects. In the AI-IoTBPM Server Drools-jBPM application project examples, we demonstrated that the actions and behavior of our IoT devices could be controlled from our BPM engine and analytical reasoning can be achieved through Drools Rules. Also, with the use of AI Drools-jBPM analysis and reasoning in IoT devices, we can orchestrate dissimilar devices that normally have no awareness of each other.

We are going to expand on what we have learned in the previous sections and add some new IoT components and devices that will be a part of our AI-IoTBPM Server Drools-jBPM Automation project. These new additions will give us additional situational awareness and allow us to interact more naturally with our IoT components and devices.

6.1.1 IoT Automation Situational Awareness (SA)

IoT Situational Awareness (SA) is the perception of environmental elements by our IoT devices and events with respect to time or space, the comprehension of their meaning, and the projection of their change of status to us (humans) in a perceivable way. This might be as simple as turning on a light, displaying a message or sounding an alarm. However, we want to know more; we want to know our SA.

Additionally, as distributed IoT sensors and applications become larger and more complex, the simple processing of raw sensor and actuation data streams becomes impractical. Instead, data streams must be fused into tangible facts, information that is combined with knowledge to perform meaningful notifications. What we have accomplished with our AI-IoT Drools-BPM implementation, is our Arduino Tron IoT devices are judging and making decisions after cognitive situational reasoning.

6.2 Arduino Tron IoT Tiles Control Panel

The Arduino Tron IoT Tiles is a control panel (dashboard) for Arduino Tron IoT Things, which controls IoT smart office automation and IoT monitoring from your desk. IoT Tiles allows you to send IoT commands directly to your IoT devices and monitor all your IoT device sensors and alert messages.

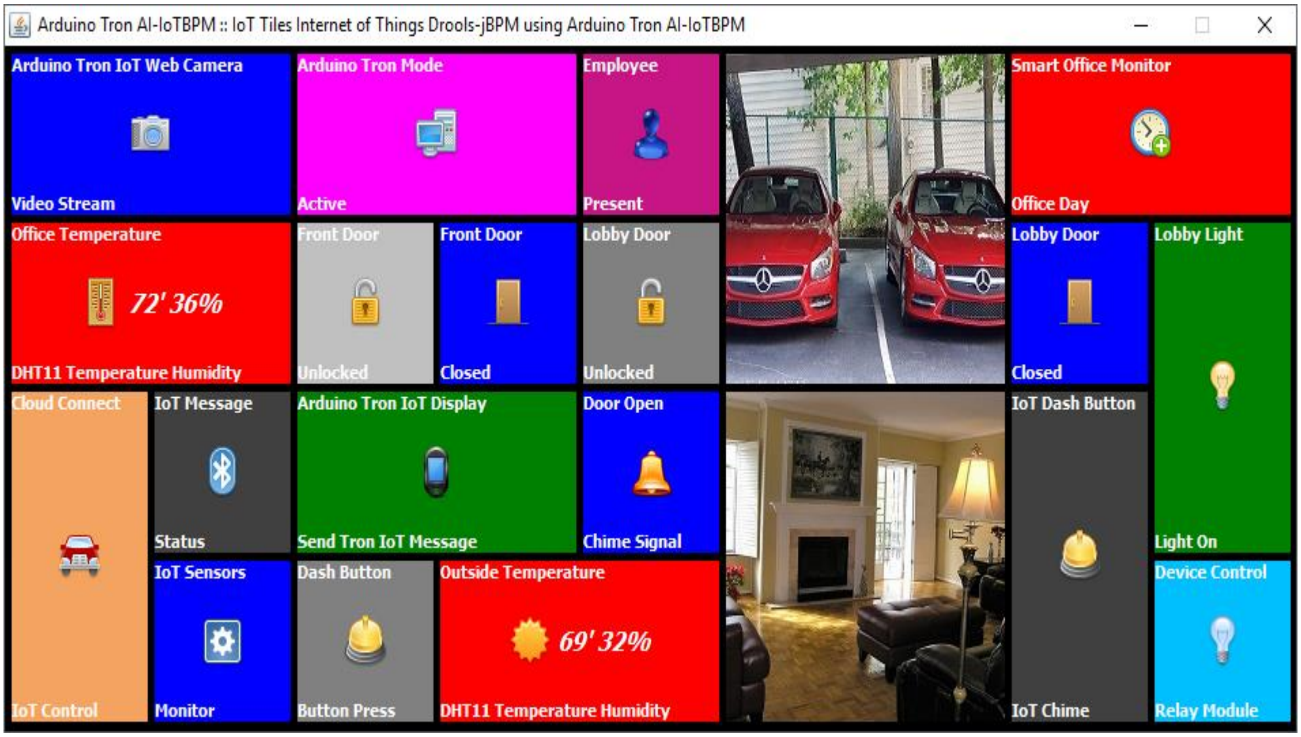


Figure 6.1.2: Arduino Tron IoT Tiles Control Panel (Dashboard)

This single dashboard gives you two-way communications with all your IoT devices and Arduino Tron IoT Things. Also, it provides situational awareness, alerts and notification messages from each of your IoT devices. Arduino Tron IoT Things events are updated “instantly” on the IoT Tiles panel. You can view the state of your Arduino Tron IoT Things: Presence Sensors, Contact Sensors, Motion Sensors, Temperature, Water Sensors, Battery Status, Vibration Sensors, Luminosity Sensors, GPS (Location), Weather (Office and Outside), or any equipment from one panel from IoT Tiles panel.

From Arduino Tron IoT Tiles you can fully control all IoT devices from one panel, use it to control: Switches, Dimmable Lights, Control lights, Momentary Switches, Theme Lights, Control Thermostats, Heating/ Cooling Thermostats, Control Things, Door Locks, Entrance Door Access, Music Players, Cameras (Image Capture), and more. Control all manufacturing floor equipment from one panel.

- **Arduino Tron IoT Tiles Automation for Controlling these IoT Devices**

1. SECURITY CAMERAS

Office security and monitoring are paramount for your property or rental space. Integrate surveillance systems with office automation audio today to take advantage of cutting edge security features like sound-activated recording and motion detection. See who's approaching and leaving your office front door--or other sensitive spaces, like your server room.

2. LIGHTING AUTOMATION

Complete control of your office automation lighting from brightness to color from your IoT office control panel. Lighting automation lets you regulate lighting options in your office remotely and conveniently.

3. OUTDOOR LIGHTING

Dim and control building lighting from your IoT device, or program the lights to turn off automatically at a designated time. If you are using a smart door sensor, you can activate the walkway lights when the office door opens or when motion is detected.

4. DOORBELL

See and talk to delivery drivers or other visitors at your loading dock or main entrance - even if you're not at the office.

5. LOCKS & SECURITY

Let smart office locks control access to your business. Locks office automation enables you to control your locks via IoT devices. Never lose another physical key or worry about whether you locked your business doors or not.

6. CLIMATE CONTROL

Use your thermostat in your office automation control system and control temperature from a tablet or computer anywhere you are. Program your office's thermostats for the nights and weekends, so you're not air-conditioning or heating empty rooms.

7. MOTION SENSORS

Wireless motion detectors for notification. Program your lights to turn on and off when people enter and leave and have the system alert you when someone's entered your office.

8. WINDOW SHADES

Reduce energy costs by programming your shades to move with the sun. Great energy savings for heating and cooling also, ambient lighting.

9. WATER LEAK ALARM

Check for water in your office and get notified of leaks and floods when you are not in the office. Water leak alarms are designed to help mitigate damage and avoid leak conditions so that they can be fixed before they become much costlier problems. These flood alerts can be crucial, and they make this sensor one of the most popular for office automation alerts.

10. PRESSURE-SENSITIVE MATS

Pressure-sensitive mats alert you to comings and goings of visitors, employees, and intruders. These wireless monitors send alerts to you when someone steps on the mat. Most are suitable for indoor or outdoor use.

11. CONFERENCE ROOM RESERVATIONS

Smart-office automates a conference room reservation. As soon as you walk into a meeting room, RFID software recognizes you and books the room on the spot.

12. HUBS & CONTROLLERS

Office automation has enabled central control devices from which you control all your automation mechanisms. Arduino Tron IoT Tiles is affordable office automation options for you, customized this hub controller for your office needs.

The beauty of the Arduino Tron IoT Tiles is that all your IoT device technology becomes interactive with humans through one simple IoT Tiles panel dashboard on your computer. With the Arduino Tron IoT Tiles getting your IoT project working with an execllent human interface is a fast-easy solution.

6.2.1 Arduino Tron IoT Display

The Arduino Tron IoT Display is a small and inexpensive display that can sit on your desk or be mounted to a wall. It provides situational awareness, alerts and notification messages from your IoT devices. The Arduino Tron IoT Things events are displayed "instantly" on the IoT Display. You can think of the IoT Display as a single IoT Tiles panel window, focused on displaying priority alerts from you Arduino Tron IoT Things. Priority alerts like "Office Door Opened" message. The Mini Arduino Tron IoT Display can be integrated with any wireless security or IoT alert system, and display alerts for driveway alarms, motion sensor, and delivery detection alerts, wireless door entry chime, doorbell or panic button alarms.

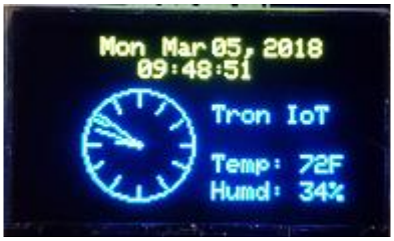


Figure 6.1.3: Arduino Tron IoT Display

This very small and inexpensive Arduino Tron IoT Display uses an SSD1306 OLED backlight display module and ESP-01. The IoT Display ESP-01 module is a receiver which connects via the WiFi network to accept IoT data and display the alerts and messages to you instantly. The Arduino Tron IoT Display is a single point to display meaningful alerts from you from IoT devices. This helps you know what is going on with your IoT network and devices.

It is much easier for you to read the IoT Display alert message than for you to fumble for your cell phone to read an alert notification that may disappear. The Arduino Tron IoT Display is a notification module that can also provide information when installed into your equipment or a professional desk interface for IoT devices.

The Arduino Tron IoT Display can be configured to continuously display the time, temperature and humidity, and then display flashing alert messages when they arrive. The temperature and humidity sensor values displayed on the Arduino Tron IoT Display module can be from your office, from outside or from any location in the world. You can read the Temperature and Humidity from any Arduino Tron DHT11 Temperature and Humidity Sensor ESP-01 WiFi Wireless Module, or DHT11 Transceiver sensor and create a fully functional weather station that can display conditions from anywhere in the world.

6.2.2 Arduino Tron Web Server Cloud Interface for IoT Device Control

IoT device management is the process of configuring, monitoring and maintaining the device software that provides its functional capabilities. Effective IoT device management is critical to establishing and maintaining the health, connectivity, and security of all of your IoT devices. The Arduino Tron Web Server provides device management to set IoT device parameters, activate and deactivate your devices, and grant access control of your IoT devices parameters from the Internet.

The beauty of the Arduino Tron Web Server is that the IoT device technology becomes interactive with humans through a simple Web Server browser interface. The integration between Arduino Tron Web Server and IoT devices presents a viable control and notification solution with a bright future – one that will connect people, things, and systems together as part of business-critical processes as never before.

The Arduino Tron lightweight Web Server provides an IoT dashboard, management, and control for remote management of your Internet-Enabled IoT devices. With the Arduino Tron Web Server getting your IoT project working in the cloud is a fast-easy solution. The Arduino Tron lightweight Web Server is a cloud-connected complete SoC (System on a Chip) architecture that integrates all components of a computer, WiFi and Web Server application software on an ESP-01 or ESP8266 WiFi chip for complete control of Internet of Things (IoT) devices from the cloud.

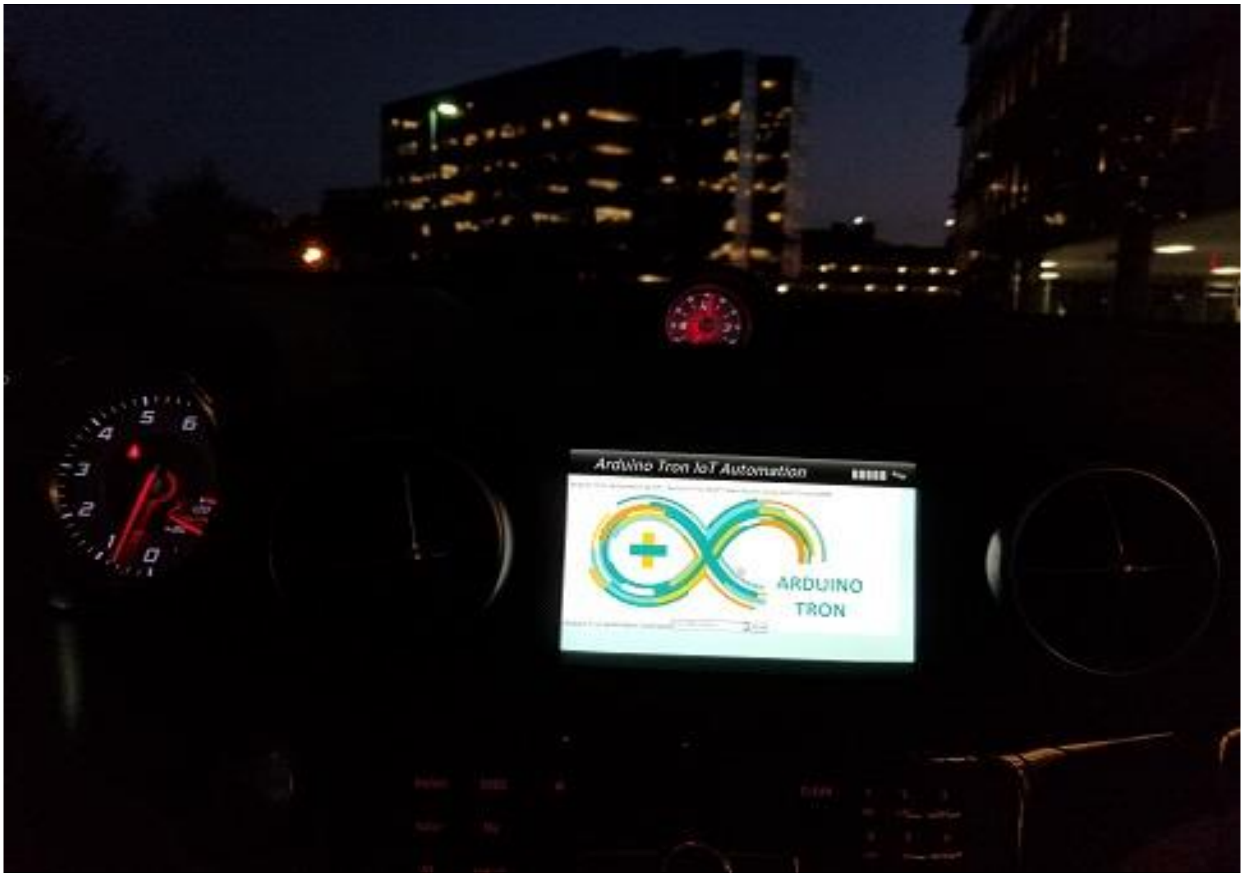


Figure 6.1.4: Arduino Tron Web Server Cloud Interface

The Arduino Tron Web Server Cloud Interface for IoT Device Control can be addressed from any web browser anywhere in the world. This connection is from a car's in-dash display using the automobile's web browser to connect to the Arduino Tron Web Server Cloud Interface.

Some applications for the Arduino Tron Web Server Cloud Interface:

- Support for MQTT IoT device control and managing
- Bidirectional IoT device communication and data storage
- Send and receive IoT data between sensor readers or triggering devices

  - Cross-device platform and server messaging
  - Monitor device sensors and stored metadata
  - Monitor and track device status or consumption levels in real-time
  - Automated code executes to trigger alerts or IoT device actions

- **Arduino Tron Web Server Interface Sketch for Controlling IoT Devices**

The Arduino Tron Web Server software uses a web browser interface and a WiFi wireless transceiver interface to stream information to the BPM-Drools Server from any remote-control Arduino Web Server.

The Arduino Tron SoC ESP-01S or ESP8266 WiFi Transceiver Module with 4MB Flash memory both provide comprehensive IoT device control and management with a very small, inexpensive and lightweight IoT device cloud Internet-connected solution.

The configuration information needed for the Arduino Web Server is similar to the Arduino Tron sketch.

Update the with WiFi network values for network SSID (name) and network password.

```
// Update these with WiFi network values
const char* ssid     = "your-ssid"; // your network SSID (name)
const char* password = "your-password"; // your network password
```

Update the IoTBPM Server IP address and unique unit ID values and add in EOSPY Server.

```
// Update these with EOSpy service IP address and unique unit id values
byte server[] = { 10, 0, 0, 2 }; // Set EOSpy server IP address as bytes
String id = "334455"; // Arduino Tron Device unique unit id
```

You will need to set a different device unique identifier for each one of your Arduino Tron Sensor ESP8266 MQTT devices. Next match the Arduino Tron sensor device unique identifier to the device unique unit ID in the Arduino Tron server application. Above are all the fields you need to provide configure values for the remaining fields that are used in the Arduino Tron Web Server application.

6.3 Arduino Tron ESP-01 Wireless Alert Sensor

The Arduino Tron ESP-01 Wireless Alert Sensor is for automated remote monitoring and notification. This IoT device reduces the resources needed to manually monitor people, places and things that could be self-monitored. The Arduino Tron ESP-01 Wireless Alert Sensor offers an easy-to-use, wireless monitoring solution that uses existing WiFi networks and internet access to gather sensor data and alert notifications instantly.

Some uses for Arduino Tron ESP-01 Wireless Alert Sensor are refrigerator temperatures, plumbing leaks, door and window access, humidity or server room temperature, front door access and much more. The Arduino Tron ESP-01 Wireless Alert Sensor online cloud-connected IoT WiFi, complete SoC (System on a Chip) provides alerts when specific conditions are met. You can address security alerts and maintenance problems before they become expensive repairs.

Arduino Tron ESP-01 Wireless Alert Sensor can be used as a simple remote alert that will immediately send an alarm signal via the WiFi network back to the Arduino Tron AI-IoTBPM Drools-jBPM Server. The Arduino Tron Wireless Alert Sensor can be integrated with any wireless security system, mailbox, garage door, driveway alarms, motion sensor, delivery detect alerts, wireless door entry chime, doorbell or panic button alarms. The Arduino Tron Wireless Alert Sensor provides for audio (door chime) or device activation (such as lamp or appliance).

Arduino Tron ESP-01 Wireless Alert Sensor Applications:
- Warehouse door monitoring
- Freezer and cooler door monitoring
- Panic or emergency button
- Button or switch integration
- Automatic trip or location sensor
- Production line tracking

- **Arduino Tron ESP-01 Wireless Alert Sensor Principles of Operation**

The Arduino Tron ESP-01 Wireless Alert Sensor can be used with any contact switch. The Arduino Tron ESP-01 Wireless Alert Sensor sends via the WiFi wireless transceiver to the BPM-Drools Server from any remote location in the world. The BPM-Drools Server can log the information and then activate any other automated process based on the event. This can be something as simple as an IoT Wireless Open-Door Chime or as complex as an employee access authorization and notification.

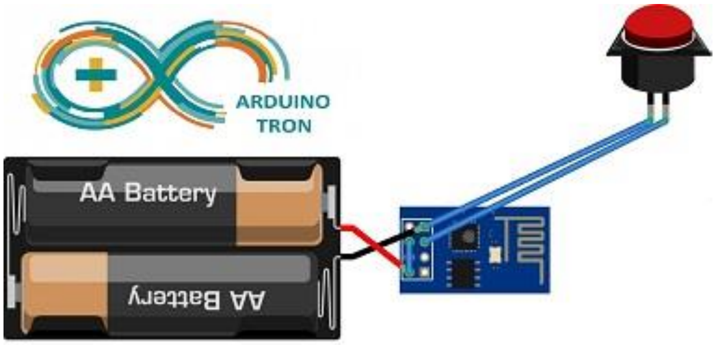
The configuration information needed for the Arduino Tron ESP-01 Wireless Alert Sensor is the same as the Arduino Tron sketch (for the ESP8266). The hardware build difference is that the ESP-01 is in deep sleep mode until the RESET pin is connected to a LOW signal (pushbutton is pressed). *ESP.deepSleep(0);* The ESP-01 uses the CH_PD (enable / power down, must be pulled to GRD directly or via a switch). The CH_PD (chip select) is all that is needed for the operation of the module. This eliminates the use of the GPIO0 and GPIO2 that determines what mode the module starts up in.

6.3.1 Arduino Tron Wireless Dash Button

The Arduino Tron ESP-01 Wireless Dash Button Alert WiFi Transmitter is for automated remote monitoring and notification. The Arduino Tron IoT Dash Button is a programmable button based on the Arduino Tron WiFi Dash Button Alert sketch. This simple WiFi dash button device is easy to configure and designed to get you started quickly using the Arduino Tron AI-IoTBPM Drools-jBPM Server IoT and Arduino Tron Cloud Services.

You can configure the button's action in the cloud to count or track items, call or alert someone, start or stop something, order services, or even provide feedback. For example, you can click the button to unlock or open your office door, open your garage door, call an employee, call a customer service representative, track the use of common office items, or products, or remotely control your office appliances. The button can be used as a remote control for any shop floor equipment. You can integrate it with other IoT devices or even your own company's applications.

Arduino Tron ESP-01 Wireless Dash Button can be used as a simple remote alert that will immediately send an alarm signal via the WiFi network back to the Arduino Tron AI-IoTBPM Drools-jBPM Server. The Arduino Tron Wireless Dash Button can be integrated with any wireless security system, mailbox, garage door, driveway alarms, motion sensor, delivery detect alerts, wireless door entry chime, doorbell or panic button alarms. The Arduino Tron Wireless Alert Sensor provides for audio (door chime) or device activation (such as lamp or appliance).



The programmable Arduino Tron WiFi IoT Dash Button is a small device that can be mounted to a wall or a button that sits on your desk or table. It provides situational awareness, alerts and notification messages from your IoT devices.

The Arduino Tron Wireless Alert Sensor provides for audio (door chime) and can be integrated with any wireless security system, driveway alarms, motion sensor, delivery detect alerts, wireless door entry chime, doorbell or panic button alarms. This adds audio functionality to the Arduino Tron Wireless Alert Sensor, used for wireless door entry chime, doorbell, or panic button alarms.

6.4 Arduino Tron ESP-01 Agent Expansion Module

The Arduino Tron IoT ESP-01 Agent WiFi Relay Expansion Module board is a simple and easy-to-use expansion board that uses the ESP-01 to drive a relay and operate devices or machines wirelessly.

The Arduino Tron IoT ESP-01 Agent WiFi Relay Expansion Module board can be used to connect devices so that multiple devices or machines act in unison through the IoT BPM-Drools Server. The Arduino Tron IoT ESP-01 Agent WiFi Relay Expansion Module board can be used to connect to office door locks, activate security alarms, turn on office lights, control thermostats, answer the doorbell, open window shades, activate motion sensors, and to control any equipment on the shop floor machines.

The Arduino Tron Agent software interface allows you to send commands with the Arduino Tron AI-IoTBPM server software to control external Arduino connected devices. The Arduino Tron AI-IoTBPM server software uses a WiFi wireless transceiver interface to control and interact with module sensors and remote control devices. Control any device from the Arduino Tron Agent software or stream any interface over the WiFi internet. With the Arduino Tron Agent software you can automatically turn on lights, appliances, cameras, and lock/unlock doors from the Drools-jBPM expert system processing.

A long pin ESP-01 shield can add additional functionality to the WiFi Relay Expansion Module board, like a speaker or other push button function. The shield eliminates the problems of trying to modify the WiFi Relay Expansion Module board and lets you simply plug in the ESP-01. Then plug the shield into the board. With a speaker added to the WiFi Relay Expansion Module board, you can use the additional sketch commands: /CHIME and /TONE. This adds audio functionality to the Arduino Tron Wireless Alert Sensor, used for wireless door entry chime, doorbell, or panic button alarms.

6.5 Arduino Tron DHT11 Temperature and Humidity Sensor

The Arduino Tron DHT11 Temperature and Humidity Sensor ESP-01 WiFi Wireless Module allows you to send information from the DHT11 digital temperature and humidity sensor directly to the Arduino Tron AI-IoTBPM Drools-jBPM Server. This module uses the ESP-01 as the main controller, and the temperature-humidity sensor is DHT11.

Also, you may use a DHT11 digital temperature and humidity sensor with the Arduino Tron Sensor sketch. The ESP8266 collects the environment temperature and humidity then uploads to the Arduino Tron AI-IoTBPM Drools-jBPM Server.

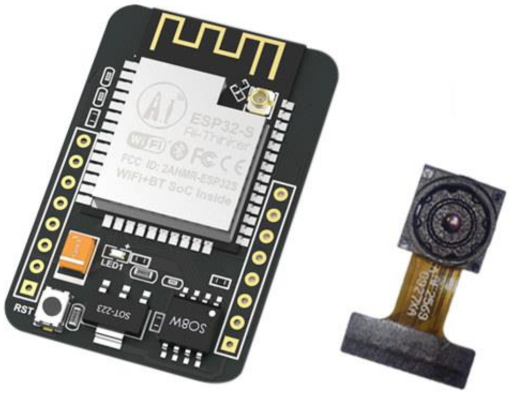
All the fields you have provided values for so far are for WiFi communications; the remaining fields are used in the Arduino Tron sensor application. To use a DHT11/DHT12 digital temperature and humidity sensor, see the Arduino Tron sensor sketch for more details and information.

```
const bool readDHT11Temp = true;
                    // Values for the DHT11 digital temperature/humidity sensor
```

To use a DHT11 digital temperature and humidity sensor, install the SimpleDHT library and uncommit the lines for dht11 in the Arduino Tron sensor (see the sketch for more details and information). Change the readDHT11Temp value to true. Also, you can change the reporting time by modifying the time-counter that calls the Arduino Tron sensor sketch function for reading the DHT11.

With the Arduino Tron DHT11 Temperature and Humidity Sensor ESP-01 WiFi Wireless Module configured, we can use DHT11 IoT information stream to decide what behavior and business process functions to execute based on temperature or humidity with the AI-IoTBPM Drools-jBPM Server. Also, the Arduino Tron DHT11 can be configured to continuously transmit temperature and humidity information to the IoT Display for a constant display of conditions.

6.6 Arduino Tron ESP32-CAM IoT Web Camera



The Arduino Tron ESP32-CAM IoT Wireless Streaming Video Web Camera is paramount for office security and monitoring for your property / rental space. Integrate surveillance systems with office automation audio today to take advantage of cutting edge security features like sound-activated recording and motion detection. See who's approaching and leaving your office front door--or other sensitive spaces, like your server room.

The Arduino Tron ESP32-CAM Wireless Web Cam is for automated remote monitoring and video notification. This IoT Web Cam device reduces the resources needed to manually monitor people, places and things that could be self-monitored.

The Arduino Tron ESP32-CAM Video Streaming and Face Recognition Wireless Web Cam offers an easy-to-use, wireless monitoring solution that uses existing WiFi networks and internet access to gather video sensor data and alert notifications instantly.

The Arduino Tron ESP32-CAM is a very small camera module with the ESP32-S chip. It features an OV2640 camera, several GPIOs to connect peripherals, and microSD card slot. The ESP32-CAM is a low-cost, low-power system on a chip (SoC) series with Wi-Fi & dual-mode Bluetooth capabilities. It allows you to set up a video streaming web server, build a surveillance camera to integrate with your home and office automation systems, and the ESP32-CAM can do face recognition and detection.

In addition to the OV2640 camera and several GPIOs to connect peripherals, the ESP32-CAM also features a microSD card slot that can be useful to store images taken with the camera or to store files to serve to web clients. It is suitable for home smart IoT devices, industrial wireless control, wireless monitoring, QR wireless identification, wireless positioning system signals and other IoT applications.

The ESP32 board is perfect for Edge computing, machine vision, face recognition. The ESP32-CAM is designed for mobile, wearable electronics, and Internet-of-Things (IoT) applications. It features all the state-of-the-art characteristics of low-power chips, including fine-grained clock gating, multiple power modes, and dynamic power scaling.

The ESP32 camera has an image array capable of operating at up to 15 frames per second (fps) in UXGA resolution with complete user control of image quality, formatting and output data transmitted or stored. All image processing functions, including exposure control, gamma, white balance, color saturation, hue control, white pixel canceling, noise canceling, and more, are all programmable through the SCCB Web Cam interface.

The ESP32 Camera WiFi+Bluetooth Module 4M PSRAM ESP32 5V Low-Power Dual-core 32-bit CPU, with OV2640 2 Million Pixels Camera ESP-32CAM SoC Module provides SPI / SDIO or I2C / UART interfaces.

<u>Arduino Tron ESP32-CAM Web Cam System Features</u>:

- The smallest 802.11b/g/n Wi-Fi BT SoC module
- Low power 32-bit CPU also serve the application processor
- Up to 160MHz clock speed, summary computing power up to 600 DMIPS
- Wi-Fi 802.11 b/g/n/
- Security WPA/WPA2/WPA2-Enterprise/WPS
- Bluetooth 4.2 BR/EDR and BLE standards
- Supports interface UART, SPI, I2C, PWM
- Supports SD card Maximum support 4G
- Built-in 520 KB SRAM, external 4M PSRAM

- Supports UART/SPI/I2C/PWM/ADC/DAC
- Supports OV2640 and OV7670 cameras, built-in flash lamp
- Supports image WiFI upload
- Supports SD card slot
- Supports multiple sleep modes
- Embedded Lwip and FreeRTOS
- Supports STA/AP/STA+AP operation mode
- Supports Smart Config / Air Kiss technology
- Supports for serial port local and remote firmware upgrades (FOTA)

6.7 Arduino Tron ESP-01 VOX-Audio Voice Alert

The Arduino Tron VOX-Audio Voice Alert system is a wireless WiFi internet connected, customizable security and notification IoT device for homes and businesses. This cloud-based audio technology gives a voice to any of the items connected on the Internet of Things. Everything from your alarm clock to your dishwasher, and industrial appliances to medical devices, essentially can now talk.

IoT devices - everyday devices that are interconnected with one another, and connect to the internet have enhanced day-to-day life for many people, audio capabilities and audio enhance these connected IoT devices themselves. Audio interactivity improves ease of use and allows users to utilize devices in a way that is most convenient for them.

Appliances like clocks, phones, smoke alarms, televisions, and automobiles, can alert users about service, or simply tell them when a cycle is done. Apps of all types can be brought to life in a way that is convenient and accessible to users. The possibilities are limited only by imagination.

IoT encompasses more and more devices that have an awareness of each other. This enhanced technology is about convenience and improving day-to-day life. The Arduino Tron VOX uses the latest technology in IoT wireless WiFi and digital voice alert systems and is ideal for audible announcements and warning in hospitals, doctor's surgery, passport control, airport arrivals, train stations, car parks, anywhere warnings or instruction messages are required.

The Arduino Tron VOX-Audio Voice Alert System can be a stand-alone unit or intergrade into a local public-address (PA) system that audibly delivers a pre-programmed message. When triggered by an Internet signal sent to the Arduino Tron VOX unit, it plays a pre-recorded voice message.

Common applications for the Arduino Tron VOX include: Wireless Driveway Alarm, Perimeter Security, Customer and Visitor Notification, Customer Greeting, Employee Safety, Perimeter Security, Intruder Detection, Visitor Notification, Outside Structure Protection, Auto Security, General Property Protection, Pool Security, Delivery and Supply Room Monitoring, and Annunciation of activity at CCTV Camera.

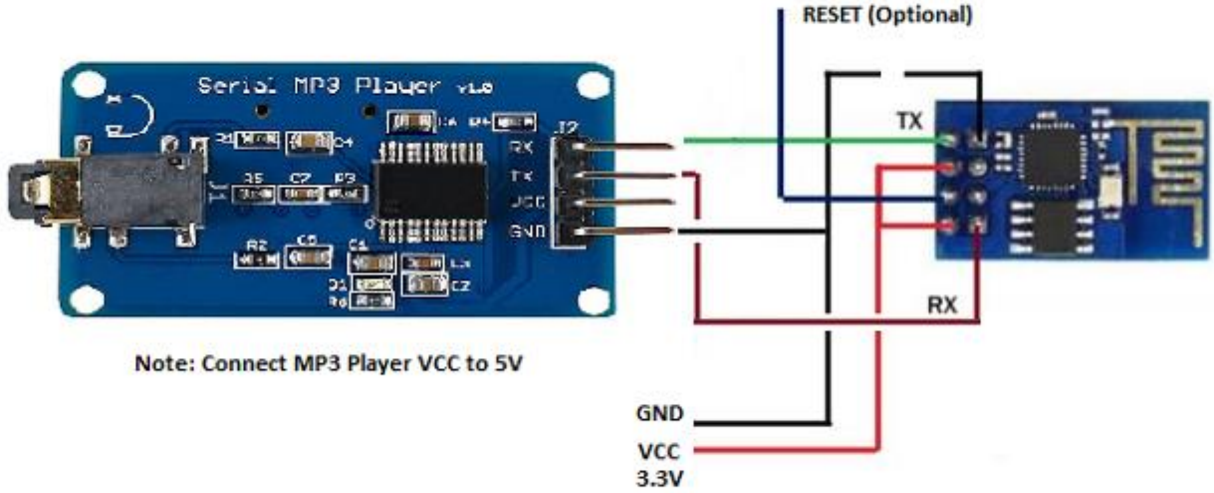
Institutions and businesses can use the Arduino Tron VOX automatic audio alert system to deliver an audible message, alert, warring or instruction. The Arduino Tron VOX System can be incorporated into hospitals, offices, engineering establishments, doctors or dentists waiting rooms or other internal environments where audible instructions are required.

Common Home Applications	**Alert Message Examples**
Notification of Car coming up the driveway	"Someone in the Driveway"
Guest at the front door	"Guest at the front door."
Some by the pool	"Someone is by the pool."
Someone in the tool shed	"Someone in the shed."

Common Business Applications	Alert Message Examples
Storeroom or Restricted Access Security	"Someone in the storeroom."
Customized Entry Greeting to Clients	"Welcome to ABC Mart"
Notification of Customer Entry	"Customer in front."
Shipment Delivery Notification	"Delivery arriving."
Annunciation of CCTV Camera Activity	"Motion by Camera"
Warehouse Inventory Protection	"Intruder in Warehouse"

The Arduino Tron VOX-Audio Voice Alert is an IoT wireless WiFi Web Server that operates completely on an Arduino ESP8266 ESP-01S WiFi Serial Transceiver Module with 4MB Flash, integrated low power 32-bit CPU RISC processor, on-chip memory and integrated WiFi 802.11 b/g/n. The ESP-01 Arduino Tron smart microdevice is about the size of a quarter and very cost effective.

The Arduino Tron VOX-Audio Voice Alert System is based on a high-quality MP3/ WAV audio YX5300 chip. You can record an unlimited number of your own custom voice alerts, messages, sounds or music. It supports 8k Hz - 48k Hz sampling frequency for MP3 and WAV file formats. Plug-in a Micro SD card into the onboard TF Card Socket that stores your audio files for automatic announcements, warnings, instructions, voice messages, music, sounds, etc.



Note: Connect MP3 Player VCC to 5V

Control the MP3 playback state by sending commands via the internet to the Arduino Tron VOX-Audio Voice Alert System module. You can switch files to play, change the volume settings and play modes.

Arduino Tron VOX-Audio Voice Alert System Features:

- Supports sampling frequency (kHz): 8 / 11.025 / 12 / 16 / 22.05 / 24 / 32 / 44.1 / 48
- Supports file format: MP3 or WAV
- Supports Micro SD card, Micro SDHC Card
- Onboard 3-watt mono amplifier
- Onboard speaker interface can connect to external speakers, 2-watt, 8-ohm
- 30 adjustable volume levels
- Onboard headphone jack can connect headphones or external amplifier

Supported Card Type: - Micro SD Card (<=2G); - Micro SDHC Card (<=32G)

Notes about the Serial MP3 Player:

1. The MP3 player chip reads .MP3 or .WAV files indexed alphabetically if you create folders with the names 01, 02, 03, etc. And then create your audio files with the name 001xxxxx.mp3, 002xxxx.mp3. You can index and be sure about the file/folder to use.
2. The Serial MP3 Player v1.0 works better with a 5v VCC supply instead on the 3.3v
3. See Arduino_Tron_VOX.ino sketch for more details on the board build.

6.8 Arduino Tron IoTBPM Server Architecture

The power of the IoT (Internet of Things) device increases significantly when business process (jBPM) can use them to provide information about our real-world as well as execute IoT Agent (devices) as part of our business process. The jBPM-BPMN modular allows us to define both the business processes and IoT devices behavior at the same time using one diagram.

With Arduino Tron IoTBPM Server adding Drools and jBPM to IoT, we make the IoT devices "smart." Moving beyond just collecting IoT device data and transitioning, to leveraging the new wealth of IoT data, to improving the SMART decision making is the key. The Executive Order Arduino Tron AI-IoTBPM Server will help these IoT devices, environments, and products to self-monitor, self-diagnose and eventually, self-direct.

The Arduino Tron allows you to send IoT sensor data and information directly to the AI-IoTBPM Drools-jBPM Expert System from the Arduino device. This provides a very lite streamline IoT to Drools-jBPM (Business Process Management) application process with sensor, measurement, telemetry and GPS positioning information. The Arduino Tron IoTBPM Server provides us the three-tiered architecture to provide a complete AI-IoT system.

- **AI (Drools-JBPM):** AI- Analysis Temporal Reasoning and CLOUD Database
- **Server (IoTBPM):** Big Data, Capture, Storage, Analysis IoTBPM Data
- **IoT (Arduino Tron):** Environment Sensors, IoT Data Collection, and MQTT

jBPM Business Process Management

jBPM is a flexible Business Process Management (BPMN 2.0) that allows you to model, execute, and monitor business processes throughout their life cycle. Business Process Management (jBPM) was established to analyze, discover, design, implement, execute, monitor and evolve collaborative business processes within and across organizations. BPM implements a strategic **Goal** of an organization.

A business process allows you to model business goals by describing the steps that need to be executed to achieve those goals, and the order of those goals are depicted using a (BPM) flow chart. Executable business processes bridge the gap between business, users, developers and IoT devices as they are higher-level and use domain-specific concepts that are understood by business users but can also be executed directly by developers and IoT devices.

Additionally, jBPM is an extremely time-sensitive and responsive technology that allows time-critical, dynamic business processes to be changed quickly and while processes are still in progress. This means that jBPM systems can take advantage of the real-time nature of data coming from and going to our IoT devices. This is an ideal fit for our IoT business needs. jBPM supports human-centric, system-centric and hybrid scenarios.

In a human-centric scenario, the jBPM system factors in the human element in the business process, putting the person in the center of decision-making and action. This makes it a great match for IoT medical and wearable technology advances.

In system-centric and hybrid scenarios, we have demonstrated how we can repurpose a legacy signaling device to operate with new or different behavior in IoT. This demonstrates opportunities for direct integration of computer-based systems into the physical-world that has never been available before by repurposing legacy systems or mechanical systems instead of replacing the hardware or physical machine. This also allows you to update legacy mechanical system with new safety features and device status notifications that were never built into the original systems.

Where is the human role in this increasingly systemized world? The beauty of IoT and jBPM is that technology becomes an important factor for its users. Systems can guide, advise and leave the most difficult decisions to the experts. Hand in hand, users and their system will be better equipped to provide easier, faster and more optimized service. The integration between IoT devices and jBPM presents a viable solution with a bright future – one that will connect people, things, and systems together as part of business-critical processes as never before.

- **The Internet of Things (IoT)**

At Executive Order Corp., we are uniquely positioned with the hardware and software experience to help your business capitalize on the **IoT** revolution. We help your company with **IoT.**

  - **Physical Engineering**: Electrical, Component, and Mechanical
  - **Design**: Prototyping, Industrial design, UI / UX
  - **Manufacturing**: Design for Manufacturing, Supply Chain Management (SCM)
  - **Security**: Device, Software, Cloud, and Protocol Specific such as BLE 4.2
  - **Software**: Networking and Infrastructure, Embedded-Systems Programming, Big Data, Machine Learning, Servers, Cloud Computing, Arduino Apps and Web

"**Internet of Things**" is a set of technology that will gradually and sometimes almost imperceptibly begin to affect us in the coming years. Any specific device or application might be small, but the combination of sensors and devices creates significant long-term changes that can both make our lives easier and more informed. The Arduino ESP8266 ESP-01S WiFi Serial Transceiver Module with 4MB Flash for Arduino provides a very inexpensive **IoT MQTT** Telemetry Transport platform.

IoT promises to provide "**smart**" environments (homes, cities, hospitals, schools, stores, offices, etc.) and smart products (cars, trucks, airplanes, trains, buildings, devices, etc.). The task of moving beyond "connected" to "smart" IoT devices is the reason for the IoTBPM Server. The Executive Order Arduino Tron AI-IoTBPM Server helps these IoT devices, environments, and products to self-monitor, self-diagnose and eventually, self-direct.

> I tell people the key with AI-IoTBPM is, "If a machine thinks, then a machine can do."
> Also, "It's not you interacting with the machine; it's the machine interacting with you."

- **Arduino WiFi MQTT Sensors and IoT Devices**

Sensors provide the data you need to automate processes and gain visibility into your business operations. At Executive Order Corp., we have extensive expertise in the Industrial Internet of Things (IIoT) and Machine to Machine (M2M) technology that can be what your company needs to increase efficiencies or distributive business models. Additionally, with IoTBPM Server you can add jBPM and Drools Rules AI-ES reasoning to your business models for "**Smart**" IoT device reasoning.

- **Custom Arduino Applications and Dedicated Devices**

For many commercial business applications, the desire is to use a device for a specific function without the distraction and security threats of an open ecosystem. Custom Arduino applications allow your business to harness the powerful capabilities of the Arduino system and tailor it to your specific needs. These dedicated device solutions can be custom mounted products or off-the-shelf handheld devices.

The Arduino Tron is an excellent, proven, pre-built, platform application that allows you to use a mobile Arduino MQTT telemetry transport device. The Arduino Tron application provides a solid and reliable platform that your company can build its own custom dedicated application on using an internet-connected or mobile network connected device.

AI-IoT Arduino Tron Sensor

The AI-IoTBPM Arduino Tron Sensor software allows you to interface and sends MQTT telemetry transport information from your external connected Arduino devices to the IoTBPM Server. The AI-IoT Arduino Tron Agent software uses a WiFi wireless transceiver interface to stream telemetry information to the IoTBPM Server for any control module sensors or remote control connected Arduino Tron device.

AI-IoTBPM Arduino Tron Agent

The AI-IoTBPM Arduino Tron Agent software interface allows you to send commands with the AI-IoTBPM Server software to control external Arduino connected devices. The AI-IoTBPM Arduino Tron Agent software uses a WiFi wireless transceiver interface to control and interact with module sensors and remote control devices.

You can control any device from the AI-IoTBPM Arduino Tron Agent software or stream any interface over the WiFi or internet. With the AI-IoTBPM Arduino Tron Agent software, you can automatically turn on lights, appliances, cameras, and lock / unlock doors from the IoTBPM Server Drools-jBPM expert system processing model.

Executive Order Corporation provides custom software and hardware development for Arduino MQTT, NodeMCU, (ESP8266 WiFi microcontroller), low cost, smart IoT, WI-FI enabled, Lua 5.1.4 devices.

The Executive Order Corp. Arduino Tron ESP8266 MQTT telemetry transport Machine-to-Machine (M2M) Internet of Things software and Arduino Tron MQTT AI-IoTBPM Client using EOSpy AI-IoTBPM Drools-jBPM latest software can be download from the Github website.

6.9 Arduino Tron IoTBPM Sensor Software Suite

6.9.1 Arduino Tron Sensor Send MQTT Telemetry Transport and Web Serer

The Arduino Tron MQTT sensor software allows you to interface and sends MQTT Telemetry Transport Information from your external connected Arduino devices to the Arduino Tron AI-IoT Drools-jBPM server. The AI-IoTBPM Arduino Tron Agent software uses a WiFi wireless transceiver interface to stream telemetry information to the Arduino Tron Agent for any control module sensors or remote control connected Arduino device.

6.9.2 Arduino Tron IoT Sensor

Arduino Tron Sensor – To install the Arduino Tron application on your Arduino device, download the Arduino Tron Sensor application from GIT. Update the WiFi network values for network SSID (name) and network password. Update the Arduino Tron Agent IP address and unique unit ID values.

Also, you may use a DHT11 digital temperature/humidity sensor and other sensors (see the Arduino Tron Sensor sketch for more details and information).

6.9.3 Arduino Tron IoT Sensor Emulator

Arduino Tron IoT Sensor Emulation - Prototype an Arduino-based low-power WiFi Internet of Things (IoT) device with built-in sensors emulation that could be used to deliver sensor data from any location in the world, and potentially control connected devices such as thermostats, lights, door locks, and other automation products.

Use a serial monitor to emulate sensor input values to Arduino Tron. This allows you to prototype the final IoT device before custom-designing the PCB (printed circuit board).

6.9.4 Arduino Tron Agent Control External Arduino Connected Devices

The Arduino Tron Agent AI-IoTBPM software interface allows you to send commands with the Arduino Tron AI-IoTBPM Server software to control external Arduino connected devices. The IoTBPM Arduino Agent software uses a WiFi wireless transceiver interface to control and interact with module sensors and remote control devices.

You can control any device from the Arduino Tron Agent software or stream any interface over the WiFi internet. With the Arduino Tron Agent software, you can automatically turn on lights, appliances, cameras, lock/unlock doors, and open doors from the Drools-jBPM expert system processing.

6.9.5 Arduino Tron Web Server Interface for Controlling IoT Devices

Arduino Tron Web Server provides an IoT dashboard, management, and control for remote management of your Internet-Enabled IoT devices. The Arduino Tron Web Server operates completely on an Arduino ESP-01, integrated low power 32-bit CPU RISC processor, on-chip memory and integrated WiFi 802.11 b/g/n. The ESP-01 Arduino Tron smart microdevice is about the size of a quarter.

The beauty of the Arduino Tron Web Server is that the IoT device technology becomes interactive with humans through a simple Web Server. With the Arduino Tron Web Server getting your IoT project working in the cloud is a fast-easy solution.

6.10Summary

Internet of Things (IoT) devices opens an opportunity to create a new generation of business processes that can benefit from IoT integration, taking advantage of networking, sensing capabilities, remote awareness and the ability for rational agents to take automated actions. A rational agent is an IoT agent that acts and that does 'the right thing.' The right thing obviously depends on the performance criterion defined for an agent, but also on an agent's prior knowledge of the environment, the sequence of observations the agent has made in the past and the choice of actions that an agent can perform.

While this data is useful, there is still "a-disconnect" in integrating these IoT devices with mission-critical business processes and corporate cloud data awareness.

The task of moving IoT devices beyond "connected" to "smart" is daunting. Moving beyond collecting IoT data and transitioning, to leveraging this new wealth of IoT data, to improving the smart decision-making process is the key to automation. Artificial Intelligence (**AI**) will help these IoT devices, environments, and products to self-monitor, self-diagnose, and eventually, self-direct. This is what we said in the opening of this book, "If a machine thinks, then a machine can do."

However, one of the key concepts in enabling this transition from connected to smart is the ability to perform **AI Analytics**. The traditional analytic models of pulling all data into a centralized source such as a data warehouse or analytic sandbox is going to be less useful. We are not trying to just analyze complex IoT data; we are trying to make "smart decisions" and take actions base on our AI Analytics of IoT devices.

***IoT Definition** – IoT is the integration of computer-based systems into our physical-world.*

With Arduino Tron (Artificial Intelligence – Internet of Things) AI-IoTBPM BPMN modular, this allows us to define both the business processes and IoT devices behavior at the same time using one diagram. The Arduino Tron AI-BRMS Drools itself is the heart of the agent that computes, and reasons based on the available data and its knowledge of the IoT sensors on the environment. With Arduino Tron adding Drools-jBPM to IoT is easy, we make the IoT devices "smart."

The power of the IoT device increases significantly when business process (jBPM) can use them to provide information about our real-world as well as execute IoT device actions as part of our business process. This increases the processing power of IoT devices, enabling them to take part in the execution of the business logic. The jBPM-BPMN modular allows us to define both the business processes and IoT devices behavior at the same time using one (BPM) diagram. In our examples, we demonstrated adding Drools-jBPM to IoT devices. Making **"Things Smart"** is the application of AI to IoT platform via DroolsRules Inference Reasoning, jBPM, and ES-Expert Systems Architecture.

With the use of AI Drools-jBPM analysis and reasoning in IoT devices, we can orchestrate dissimilar devices that normally have no awareness of each other. This creates opportunities for direct integration of computer-based systems into the physical-world that has never been available before. This results in improved efficiency, accuracy, and economic benefits by increased automation - reduced intervention. This IoT orchestration of IoT devices gives us the ability for action after our AI decision.

Artificial Intelligence (A.I.) is a very broad research that focuses on "making computers think like humans." Expert Systems (ES) uses knowledge representation: a knowledge base for reasoning, (i.e., we can process data with this knowledge base to infer conclusions).

Expert Systems and Knowledge-based Expert Systems are considered to be "applied artificial intelligence." We can reason to a conclusion (infer) beyond what we currently know.

Business Rules Management Systems builds additional value on top of a general-purpose Rules Engine by providing business user-focused systems for rule creation, management, deployment, collaboration, and analysis user tools. These further add to the value by making the Rules Engine easily implemented and ingrained into our existing Enterprise Systems.

The Arduino Tron AI-IoTBPM Drools-jBPM Expert System provides sophisticated jBPM and Drools processing. This allows us to define IoT behavior within business process and with the same level of abstraction of other atomic actions or events in our standard jBPMN.