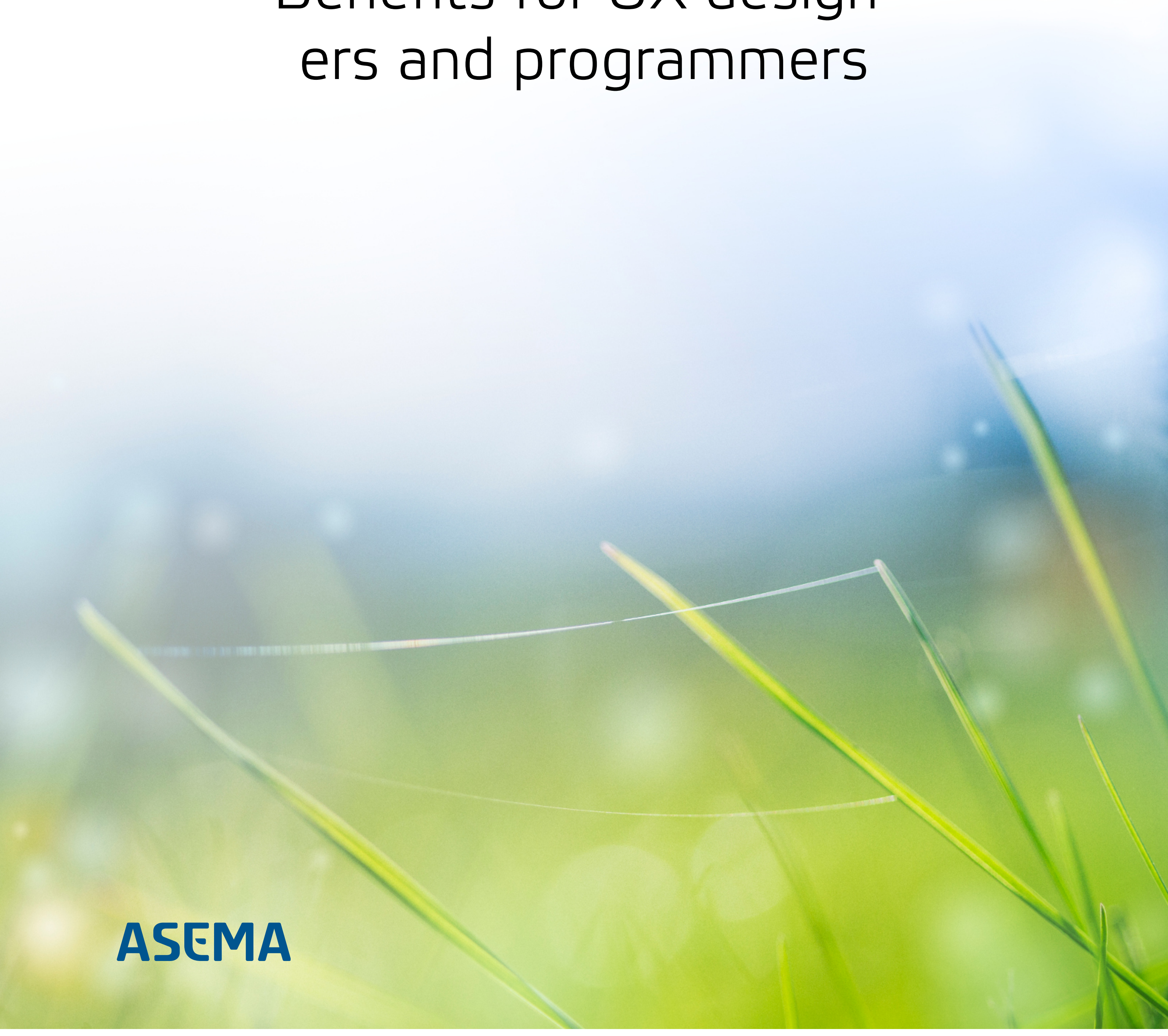
English

Asema IoT
Benefits for UX designers and programmers



ASEMA

Table of Contents

1. What are IoT projects and how does Asema IoT help in them?

Asema IoT is a software for creating Internet of Things (IoT) solutions. It offers all the building blocks and services needed to create an IoT solution ranging from database connections and storage, IoT protocols, communication drivers and device management all the way to user interface libraries, maps, Big Data analytics and mobile clients. As remote connectivity is becoming increasingly popular in various organizations, system integrators that have previously worked with ERPs, ecommerce logistics, intranets, and even document management systems are getting questions on whether they can embed some IoT functionality into the systems and projects they offer. To do that it is good to understand first what makes them different to plan and implement.

First, for many software engineers the fact that you need to actually take into account the characteristics of hardware may be the biggest hurdle when entering the world of IoT. In the best case, the implementation is just about connecting to some remote control API and assuming everything beyond that is taken care of. In the worst case the system needs direct connections to proprietary hardware boxes that have unreliable connectivity and may run out of power at any point - a headache for even skilled engineers.

A second key characteristic is the emphasis on unit costs. In a traditional software driven IT project a new user can often be considered "free" - i.e. there is little need for additional investment. But in IoT each new unit means added cost in the form of buying, installing, and maintaining the device. How that can be solved often dominates the go / no-go decision of a project.

Third, to get value from IoT, the projects often involve data integration or at least data analysis. Just connecting and collecting may be a neat technology demo, but to really get value out of the data that data needs to be put into use. Either that involves Big Data processing and statistics, or plugging the data into some other operative system.

This short document shows how Asema IoT helps in solving those issues. Asema IoT has been carefully designed to help in operating hardware reliably, with low cost, and with powerful analysis and integration possibilities. So let's dig into that a bit deeper.

2. Making hardware cheap and reliable - or skipping it altogether

Most IoT projects start with the discussion about hardware, even in cases where eventually the hardware really is not either the critical component nor the cost driver. This is where expertise is needed and Asema also helps in projects in this design phase as we have years of experience in rolling out all kinds of weird and wonderful gizmos.

Because of the background, Asema IoT has been designed since the beginning to be able to connect with hardware. It is able to process anything from dedicated IoT network protocols such as CoAP and industry standards like ModBus to operating system level messaging with DBUS, wireless protocols such as Bluetooth Smart all the way down to individual bits and bytes of I2C channels and SPI serial. Due to extensive feature support and efficient code Asema IoT is an excellent tool in cutting the cost of that hardware. It allows for maximum choice in components and consequently the ability to choose the inexpensive options.

The hardware stage is often the stage where the project can easily fail due to incorrect assumptions about cost and lacking creativity in how communication and measurement is actually done. That does take expertise and the lists of hardware methods often sound like engineering porn at best and total mumbojumbo at worst. But in practice many of the issues can be solved with common sense - and maybe a tiny bit of highschool level understanding of physics.

The majority of projects eventually need some hardware. That said, there typically still is a tendency to overdesign hardware and make it too high tech. And consequently too expensive. So while Asema IoT helps in dealing with hardware, let's actually start with a couple of examples of not doing even that. Here's a few of our favorite cases.

2.1. Simplifying the setting

A customer is setting up an automatic irrigation system for plants. Each plant has a bowl of water to which its roots go into. To make the system automatic, it needs a method to measure how much water is in the bowls. If empty, a valve opens and fills the bowl with water. Various methods for measuring how much water there is in a bowl have been considered, such as weighing the bowl and ultrasonic sensors.

The challenge is that there are 500 bowls. If one sensor costs 100, the total cost would be 50 000. That is simply way too much money. So cheaper measurement devices, say in the range 10-20, are being sought for the project to move forward. As nothing has been found so far, this seems to be a system that is impossible to implement with the given money.

But wait a second, why do you measure each bowl? It is water we are talking about. It flows. A much easier solution is to drill a hole to each bowl and connect it to the adjacent bowl with a water hose. And that to the next and so on. When water is poured into one bowl, it will flow into the next one until they both have the equal amount of water. Then measure how much water there is in one bowl. The rest will have the same amount and only one sensor is needed. The budget just got cut into 1/500th of the original. Not with high tech but with a drill and a couple of meters of garden hose.

2.2. Avoiding overdesign

A customer is building a new warehouse for parts to be used in manufacturing. They'd like to automate the purchasing of parts. The way it should work is that parts get ordered from suppliers on just-in-time fashion so that new components arrive just when the old stock is about to run out.

One item considered is a box of screws. The delivery time is known to be such that when two thirds of the box is used, a fill should be ordered. To automate this an elaborate artificial intelligence system is being planned where a camera analyzes the contents of the box and when the AI notices it is getting empty, an order is made. This seems to be an advanced task: there needs to be enough cameras on shelves to monitor various boxes. They need to reliably recognize the contents in different lighting conditions. No false alarms when the warehouse lights go off, please.

But all that is just unnecessary. Here's a simple solution. Instead of one large box of screws, have three small ones. When two are empty, order some more. No complex and expensive sensors and AI needed. Just two small boxes.

2.3. Focusing on what is relevant

A customer wants to build an energy control system to a building. To do that they are planning to control the LED lights and the heating system. The concern is that to control the lights, each of them needs a rather expensive control box.

Lighting control sounds like a great idea, we've all been taught to switch off lights when we leave the room. But a quick visit to the building control room tells otherwise. I typical heater uses at least 3kW of power, big ones even more. That is 3000 watts. One LED light is 5-10 watts. So you literally need hundreds of lights under control to get the same benefits.

So in this case it is simply better to just forget the lights, they are not relevant in this application, and control the heater. The system becomes much simpler and cheaper.

3. Multiple architectures and deployment possibilities

3.1. Go tiny, go cheap, go anywhere

Once the right type of hardware is pinpointed, this is where Asema IoT's power really comes into play. A big difference between Asema IoT and the majority of other systems available for IoT is that the core of Asema IoT has been coded with C++. Yes, not Java, not Ruby, not PHP. Hard core C++.

When it comes to IoT, there are several benefits in this approach.

First, the code is super efficient in terms of memory and speed. The Asema IoT Central server that implements all the bells and whistles of multiple IoT protocols, webservers, event handling, database connections, security, etc needs only 12 MB of memory. Compare that to a Java cloud software which typically takes 1-2 gigabytes. So you can literally run 100+ instances of Asema IoT Central simultaneously and side-by-side with the same features as some other system in the same memory. Asema IoT Edge, specifically designed for embedded computing, runs at less than 4MB. This means that it easily fits even the most memory restricted devices available. And still leaves plenty of space. Because there is no virtual machine required, the software can run on devices where one is not available: car navigators, wall panels, truck and crane computers, vending machines, and so on.

Second, because there is no Java virtual machine or similar in between to consume resources, the code runs very fast. It fully restarts in a fraction of a second and can feed in data to databases from thousands of devices simultanously. Compare that to a typical Java Tomcat which more often than not spends half a minute or more just initializing itself. And what's more, Asema IoT can directly access system resources. This becomes critical in some IoT applications where you need to connect to the hardware and need access to the device drivers. With Asema IoT there are no technical limitations to this.

The result is that there are as few boundaries to the application development. Whatever hardware is thrown at you in the project, there is always a way to hack that in. Tiny memory footprint means that the system can run with cheaper devices with limited memory. This is critical if there needs to be millions of devices out there as you do want to save every penny on their cost.

3.2. In cloud, on premise, at edge, in the pocket

Asema IoT comes in three different flavors: Central, Edge and Dashboard. As you can tell from the names, these are for different operating environments: Central for central systems such as clouds, Edge for edge computing such as embedded boxes and gateways, and Dashboard for viewing and remote control systems including smartphones, tablets and POS displays.

With that number of choices, you can build a system that always fits the particular needs of each client. If they want a central cloud, install Asema IoT Central into it. The software has been tested to function perfectly on Amazon Web Services, Microsoft Azure, Vultr, Rackspace and several others. If there is a requirement to run on dedicated servers on premises, no problem. The system can even run in a fully closed lab with no Internet connections to the outside world. If there is a need to run it without any server investment i.e. fully SaaS, well, we support that too.

All the editions sync data with each other automatically. In many IoT projects considerable amount of time needs to be spent in designing APIs between for instance a central server and mobile clients. With Asema IoT there is no need for that as each mobile Dashboard automatically syncs itself with the backend.

4. Events and shadows

When it comes to syncing data between the IoT software and hardware in the field, two directions of transfer must be considered: inbound (measuring) and outbound (controlling). For someone used to normal web programming, these can become a formidable challenge. First, in IoT usually no communication can be guaranteed as anything can be offline at any point in time. Second, lagged connections usually force programmers to twist their brains around to be able to do asynchronous programming - chained in multiple layers in the worst case.

Asema IoT solves these issues with its eventing and shadowing system and makes all that convenient for programmers. Let's show a couple of examples on what the actual difference is.

4.1. Eventing significantly simplifies user interface code

A very typical use case for an IoT user interface is to draw some gauge or bar that shows values. As values get updated, this gauge or bar needs to update. To achieve this, a programmer would typically implement something like an update timer with a network call. Like this (in JavaScript with JQuery)

```
function updateGraphs() {
 for (g in myGraphs) {
  $.ajax({
   url: mybackendapiurl + "/get?id=" + myGraphs[g].id + "&value=speed",
   type: 'GET',
   success: function(result) {
    var max = 1000;
    $("#myGraphBar_" + result.id).css("height", result.value.value / max * 400 + "px");
   },
   error: function(result) {
    if (typeof(onError) == 'undefined') {
     showErrorMessage();
    } else {
     onError(result);
    }
   }
  });
 }
}
setTimeout(updateGraphs, 5000);
```

That code will now run every five seconds, send a request to the server for each sensor and then recalculate the height of each bar and set it.

Not only is the code somewhat ugly but also has several problems. What if you'd want the values to update faster than 5 seconds? What if the server cannot reply to all the requests in 5 seconds? What if some of the requests timeout? If some of the sensors only update once per minute, why would you poll their values for nothing once every five seconds? It is evident that the approach is hard to maintain, unreliable and consumes a lot of resources unnecessarily.

Here is the same code in Asema IoT, again in JavaScript

```
myGraph.height = myObjects[id].properties.speed / 1000 * 400;
```

That really is all. One line. No other code needed. But where is the updater? How will that graph now change?

The secret is in the eventing system of Asema IoT. It is deeply integrated into the JavaScript framework itself. It automatically updates the values of objects in real time as new values come in (the backend handles this with sophisticated callback architecture that uses very low amount of resources). All the difficult network programming, value filtering, caching, database writing, signaling, etc is automatically handled and hidden in the system. When a value event occurs, that height setting, and the formula tied to it, is automatically re-evaluated. So the bar in this case really changes without any programming even though it was defined with programming. For someone who tries this the first time, it is like magic (then you get used to it - and its no longer magical but just really, really useful).

4.2. Shadowing ensures commands go through

What about control then. That usually gets even more complex as something needs to check that the command was actually received. Let's try that in a traditional way

```
function setRotationSpeed(controllerId, speed) {
 $("#lastController").val(controllerId);
 $("#lastValue").val(speed);
 $.ajax({
  url: mybackendapiurl + "/set?id=" + controllerId + "&speed=" + speed,
  type: 'GET',
  success: function(result) {
   setTimeout(pollConfirmation, 2000);
  },
  error: function(result) {
   setTimeout(retry, 5000);
  }
 });
}

function retry() {
 setRotationSpeed($("#lastController").val(), $("#lastValue").val());
}

function pollConfirmation() {
 $.ajax({
  url: mybackendapiurl + "/get?id=" + $("#lastController").val() + "&value=speed",
  type: 'GET',
  success: function(result) {
   // do something if ok
  },
  error: function(result) {
   retry();
  });
 }
}
```

Now that is decent spaghetti logic. The code somehow needs to make sure that the command went through and retry if it did not. So all kinds of timers and retries are in place. But the logic becomes horribly unreliable. What if two commands are sent at the same time to one controller? How would retries work if there are several controllers in play at the same time? It is obvious that that code just won't work properly.

And here is the same thing with Asema IoT:

```
myController.properties.speed = speed;
```

Again, that is all. What happens in the background is that the value is sent to the Asema IoT's shadow cache. This mechanism makes sure that the value reaches its destination. If the system is offline, the change is cached until the unit is online again and sent to it. All the confirms, async calls, error checking etc is done transparently in the background. Once the value has gone through to the controller, the eventing system again takes care of notifying everyone and updating graphical elements.

Both of those methods, eventing and shadowing, fully integrate into the JavaScript environment. This means that the programmers can use all their favorite tools and frameworks with them, from manual DOM modeling to AngularJS and the likes.

5. Forget dedicated mobile development

Asema IoT is a "code once, run anywhere" system. The code you create on your desktop will run as-is on a mobile. An app developed for Android will run on iOS, no modifications needed. And all this happens natively, no browser hacks required.

The way this works technically is that application packages are designed as "apps" for the Asema IoT system. These apps are programmed with JavaScript. That script is supported by the runtime in all editions of Asema IoT.

Graphics can be done eihter with HTML5 or with a powerful descriptive language called QML. The JavaScript code, QML definitions, images, videos and other material are packaged into an app packet. That packet can be downloaded to any device that runs ome edition of Asema IoT. The system ensures all of them have the same features, look and feel, gesture controls etc.

6. Server programming - watching things when no-one is watching

Asema IoT compatible apps can run either in the frontend or the backend. Most developers are used to the architecture where the heavy duty logic and number crunching is done at the backend in a server. Clients then call this logic through technologies such as servlets.

In Asema IoT Central server such backend logic is programmed again with JavaScript. The syntax and the event loop are very similar to node.js so node.js programmers should immediately feel at home with the system.

But the clever embedding of the event and shadowing system into the JavaScript engine also extends to the server side. Server code can be made stateful. It remembers what it did between calls to it. It can react to events automatically. It runs in the background even when no-one is calling.

As an example, let's say you program a system that monitors the temperature of a room. The sensor in this case is a bit wonky: to show the correct temperature, a correction factor needs to be applied. That correction factor is not constant, at zero degrees the measurement is spot on but closer to 20 degrees or above it needs a lot of correction.

A stateful server script can be left running in the background. It reacts to changes whenever a temperature measurement is received. So when a new temperature reading comes in, the correction factor can be immediately altered by the script. Corrected values can be written into database, all without calls to the server. When someone then calls the script, they get immediately correctly corrected figures. Or previous figures. Or whatever the programmer may have stored into the stateful script.

7. Maps, dashboards and all that jazz

7.1. Essential core components pre-built

In IoT there typically are a couple of very traditional views that are used in nearly all applications. One is dashboards that collect gauges, graphs, buttons, etc into one place. The second is maps that show devices in locations and tracks them.

Asema IoT comes with these components built in. Dashboards can be created and modified simply by drag and drop. Maps feature navigation integration with turn-by-turn instructions. These components are fully customizable. The framework included makes it easy to design new buttons, new icons on maps, and new graphs into the dashboards.

7.2. Samples galore

Asema IoT software comes with dozens of free sample applications for programmers to extend, modify and use. These include car sharing, traffic planning, waste handling, home control dashboards, energy displays, etc. All source code and graphics is included.

8. Smart integration

Finally there is the integration. For system integration, Asema IoT offers several methods and means. It serves REST calls with a simple and straightforward JSON API. The API can be extended either with schemas (no programming needed) or with server side scripts (in JavaScript). Those wanting a very hardcore approach can even program API plugins in C++ as DLLs (Windows) or SOs (Mac, Linux).

8.1. Choose between client or server or both

When Asema IoT Central integrated with other systems, it can do so in several modes, even simultaneously. It can act as a traditional server (source of data) or as a client that connects to a server (sink of data). Those can even be combined to that a server request responds with results from a client call forward.

8.2. The Smart API

The Smart API is a very powerful, very feature rich API mechanism embedded in Asema IoT Central. Unlike traditional APIs, the Smart API offers advanced features such as support for graph data, automatic unit conversion, semantic vocabularies, fuzzy searches, geospatial searches, automatic documentation, end-to-end encryption, micro payments and contracts, and more. The specification is open source and is highly recommended for open data, scientific computing, and other demanding applications.

Smart API is a complete system for consistent data management in APIs and by itself worth a couple of intro documents. For more info on the topic, please visit www.smart-api.io

Further information and support