

[BT](#)

- [Contribute](#)
- [About Us](#)
- [About You](#)
- [Purpose Index](#)

- Exclusive updates on:



Facilitating the spread of knowledge and innovation in professional software development

[Luke](#)

Welcome, Luke !

- [My Reading List](#)
- [Preferences](#)

Personalize Your Main Interests

- ☒ Development
- ☒ Architecture & Design
- ☒ Data Science
- ☒ Culture & Methods
- ☒ DevOps

This affects what content you see on the homepage & your RSS feed. Click preferences to access more fine-grained personalization.

[Sign out](#)



- [En](#) |
- [中文](#) |
- [日本](#) |
- [Fr](#) |
- [Br](#)

1,546,464 Jan unique visitors

- [Development](#)
 - [Java](#)
 - [Clojure](#)
 - [Scala](#)
 - [.Net](#)
 - [Mobile](#)
 - [Android](#)
 - [iOS](#)

- [IoT](#)
- [HTML5](#)
- [JavaScript](#)
- [Functional Programming](#)
- [Web API](#)

Featured in Development

[Rust: Unlocking Systems Programming](#)



[Aaron Turon explains Rust's core notion of “ownership” and shows how Rust uses it to guarantee thread safety, amongst other things. He also talks about how Rust goes beyond addressing the pitfalls of C++ to do something even more exciting: unlock a new generation of systems programmers by providing a safe, high-level experience -- while never compromising on performance.](#)

[All in Development](#)

- [Architecture & Design](#)
 - [Architecture](#)
 - [Enterprise Architecture](#)
 - [Scalability/Performance](#)
 - [Design](#)
 - [Case Studies](#)
 - [Microservices](#)
 - [Patterns](#)
 - [Security](#)

Featured in Architecture & Design

[Beyond the Hype: 4 Years of Go in Production](#)



[Travis Reeder thinks the performance, memory, concurrency, reliability, and deployment are key to exploring Go and its value in production. Travis describes how it's worked for Iron.io.](#)

[All in Architecture & Design](#)

- [Data Science](#)
 - [Big Data](#)
 - [Machine Learning](#)
 - [NoSQL](#)
 - [RDBMS](#)
 - [Data Analytics](#)
 - [Streaming](#)

Featured in Data Science

[Experiences Building InfluxDB in Go](#)



[Paul Dix discusses experiences building InfluxDB, an open source distributed time series database, in Go. He talks about what drove the decision to use Go, what's been really great about developing in the language, and a few of the pains that they've had along the way. He also digs into what performance characteristics they've seen in Go 1.4 vs. Go 1.5, which has a new garbage collector.](#)

[All in Data Science](#)

- [Culture & Methods](#)
 - [Agile](#)
 - [Leadership](#)
 - [Team Collaboration](#)
 - [Testing](#)
 - [Project Management](#)
 - [UX](#)
 - [Scrum](#)
 - [Lean/Kanban](#)
 - [Personal Growth](#)

Featured in Culture & Methods

[Q&A on the book Visualization Examples](#)



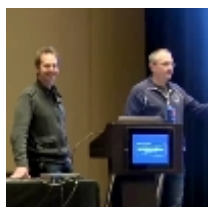
[The book Toolbox for the Agile Coach - Visualization Examples by Jimmy Janlén can be used by agile software development teams to visualize and improve their collaboration and communication. InfoQ interviewed Janlén about the strengths of visualizations and how teams can use them to track progress, deal with blockers, celebrate successes and improve.](#)

[All in Culture & Methods](#)

- [DevOps](#)
 - [Infrastructure](#)
 - [Continuous Delivery](#)
 - [Automation](#)
 - [Containers](#)
 - [Cloud](#)

Featured in DevOps

[Developing Cloud-native Applications with the Spring Tool Suite](#)



[Kris De Volder and Martin Lippert show how to work effectively with Spring projects in Eclipse and the Spring Tool Suite \(STS\). They demo all the latest enhancements in the](#)

[tools including features like much smarter property file editing, as well as new features in the Eclipse 4.5 \(Mars\) platform.](#)

[All in DevOps](#)

London

[Mar 07-11](#)

New York

[Jun 13-17](#)

San Francisco

[Nov 7-11](#)

- [Mobile](#)
- [HTML5](#)
- [JavaScript](#)
- [APM](#)
- [Java](#)
- [Microservices](#)
- [Continuous Delivery](#)
- [API Design](#)


[All topics](#)

You are here: [InfoQ Homepage](#) [Articles](#) A Reference Architecture for the Internet of Things

A Reference Architecture for the Internet of Things



Posted by [Daniel Karzel, Hannelore Marginean, Tuan-Si Tran](#) on Jan 29, 2016 / [6 Discuss](#)

- Share 
- |
-
-
-
-
-
-
-
- ["Read later"](#)
- ["My Reading List"](#)

This is the first article of a two article series in which we try to work from the abstract level of IoT reference architectures towards the concrete architecture and implementation for selected use cases. This

first article will cover the definition of a more concrete and comprehensible architecture whereas the second part will then apply this architecture to actual use cases.

We are at the edge of a new interconnected world. Under the name “Internet of Things” (IoT) or “Industry 4.0” companies are developing a new network of intercommunicating objects of our everyday life. The IoT interconnects the Things in order to exchange information to fulfill tasks for the users. Ideas of fridges communicating not only with your smart-phone, but with the producer's server farm or an energy power plant will soon become reality. Companies behind this new technology and communication boom come from all industries - not only the big-data-software players like Google, Microsoft or Apple are heading in this direction, also big insurance companies, peripheral device producers and car manufacturers are pushing the IoT.

The key to enabling communication between all of these diverse “Things” is standardization. Standardization, however, is easy to be claimed in a research environment, but difficult to be achieved in the real world. Reference architectures are of great help for standardization, as they define guidelines that can be used when planning the implementation of an IoT system.

Related Vendor Content

[Migrating from SVN to Git in 5 steps](#)

[Internet of Things: Architecture and Impact @ Oracle Virtual Tech Summit \(March 8th\)](#)

[Team Communication Made Easy - Try Atlassian HipChat](#)

[Real-Time Collaboration for Software Engineers - Try HipChat](#)

[Internet of Things 101 - Learn More about the Intel® IoT Developer Program](#)

Related Sponsor

Take Innovative Ideas Rapidly from Prototype to Production - [Build with the Intel® IoT Developer Kit!](#)



In order to achieve standardization it is necessary to create high level reference architectures like the one [defined by IoT-A](#). However, high-level reference architectures are difficult to understand because they are very abstract. If you are in a consulting job you will see that it is impossible to show such high-level reference architectures to actual customers in the industry.

We would like to go a step further and provide guidelines on how to substantiate a more concrete architecture from the [IoT-A reference architecture](#). The idea is to take the abstract IoT-A reference architecture and create a less high level architecture out of it that can even be put into a “management summary” – that’s what is happening in this article. Additionally we select some use cases and instantiate our reference architecture for them to show the complete life-cycle down to the implementation of an actual system within the IoT – that will happen in a follow-up article.

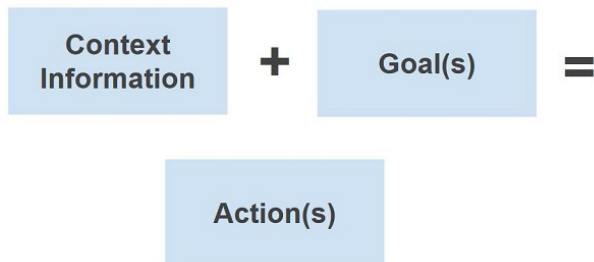
First things first, let’s define some terms:

- **Thing:** An object of our everyday life placed in our everyday environment. A thing can be a car, fridge but can also be abstracted to a complete house or city depending on the use case.
- **Device:** A sensor, actuator or tag. Usually the device is part of a thing. The thing processes the devices’ context information and communicates selected information to other things. Furthermore, the thing can pass actions to actuators.

There is a certain amount of “inevitable IoT components” that you will find (in one form or another) in every IoT reference architecture (as for example in [Google’s Brillo](#), [IoT-A](#) or [Z-Wave](#)):

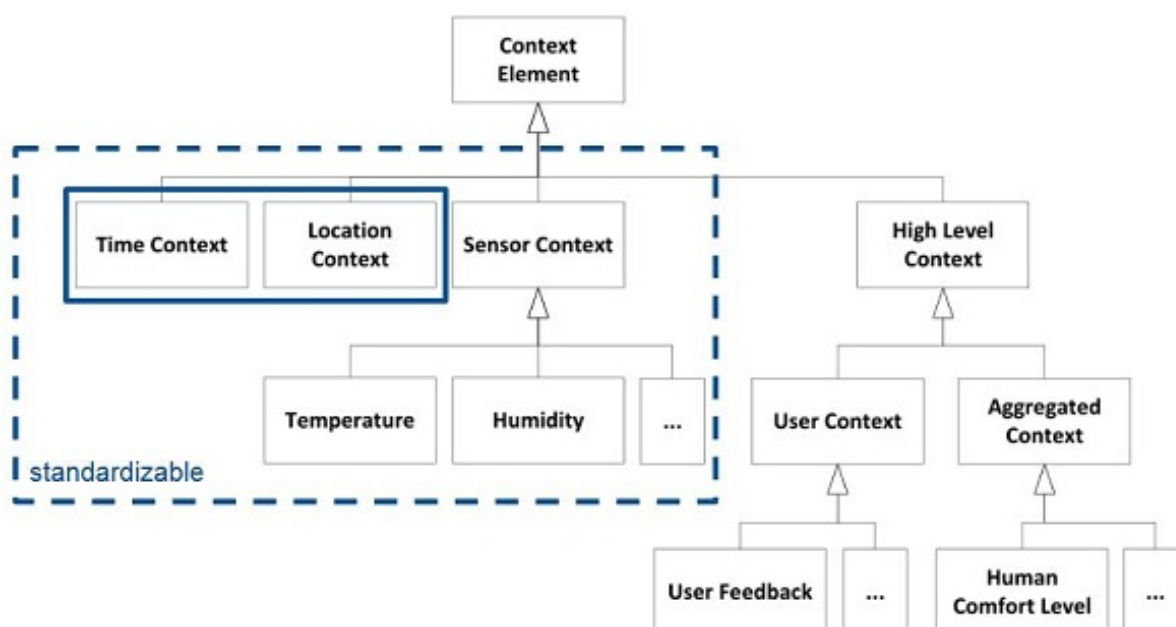
- Interoperability and integration components towards things and devices,
- Context aware computing techniques such as the definition of a context- and action-model as well as the goal definition by rule engines.
- Security guidelines that range over the complete architecture.

In a way the current architectures for the IoT can be seen as a larger-scale version of Anind K. Dey’s [context toolkit](#). The context toolkit was designed on an application level, as it was designed for Geographical Information Systems (GIS). In the IoT we have to extend the context toolkit towards the intercommunication between things. However, the basic idea of goal, context information and resulting actions remains in the IoT world.



In the IoT world we don’t only define the goal on the user level (i.e. by application), but things themselves can work towards certain goals without actively including the user. In the end the devices still serve the user but they act autonomously in the background – which is exactly the idea of [ubiquitous computing](#).

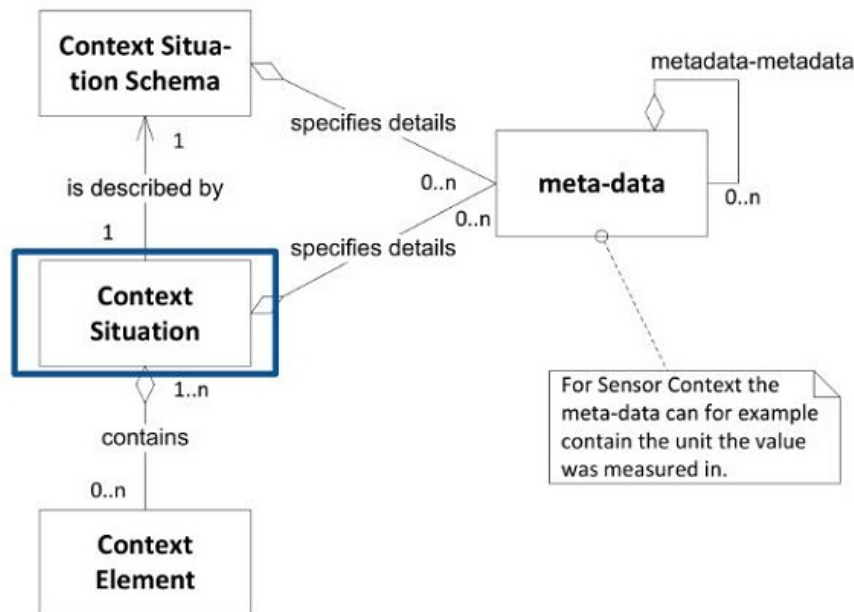
In order to get a better picture of the term “context” we will first introduce our context model and then jump into the introduction of our reference architecture. Context defines the state of an environment (usually the user’s environment) in a certain place at a certain time. The context model usually distinguishes between context elements and context situation. Context elements define specific context, usually on the device level. A context element can be for example a temperature value at a certain time and location.



Location and time are context elements themselves, but they play a special role as they are needed to locate sensor values in space and time. Without knowing where and when a temperature was measured the temperature does not help much for making conclusions.

Certain context elements can be standardized right away (e.g. a temperature value is already defined by a double value and a measurement unit such as Celsius or Fahrenheit). Other context elements are application specific (“thing-specific”) and cannot just be standardized right away. These elements are defined as “high-level” context and require a mechanism to define them for each thing.

The context situation is an aggregation of context elements. The context situation is thus a view on the environment in a certain location at a certain time.

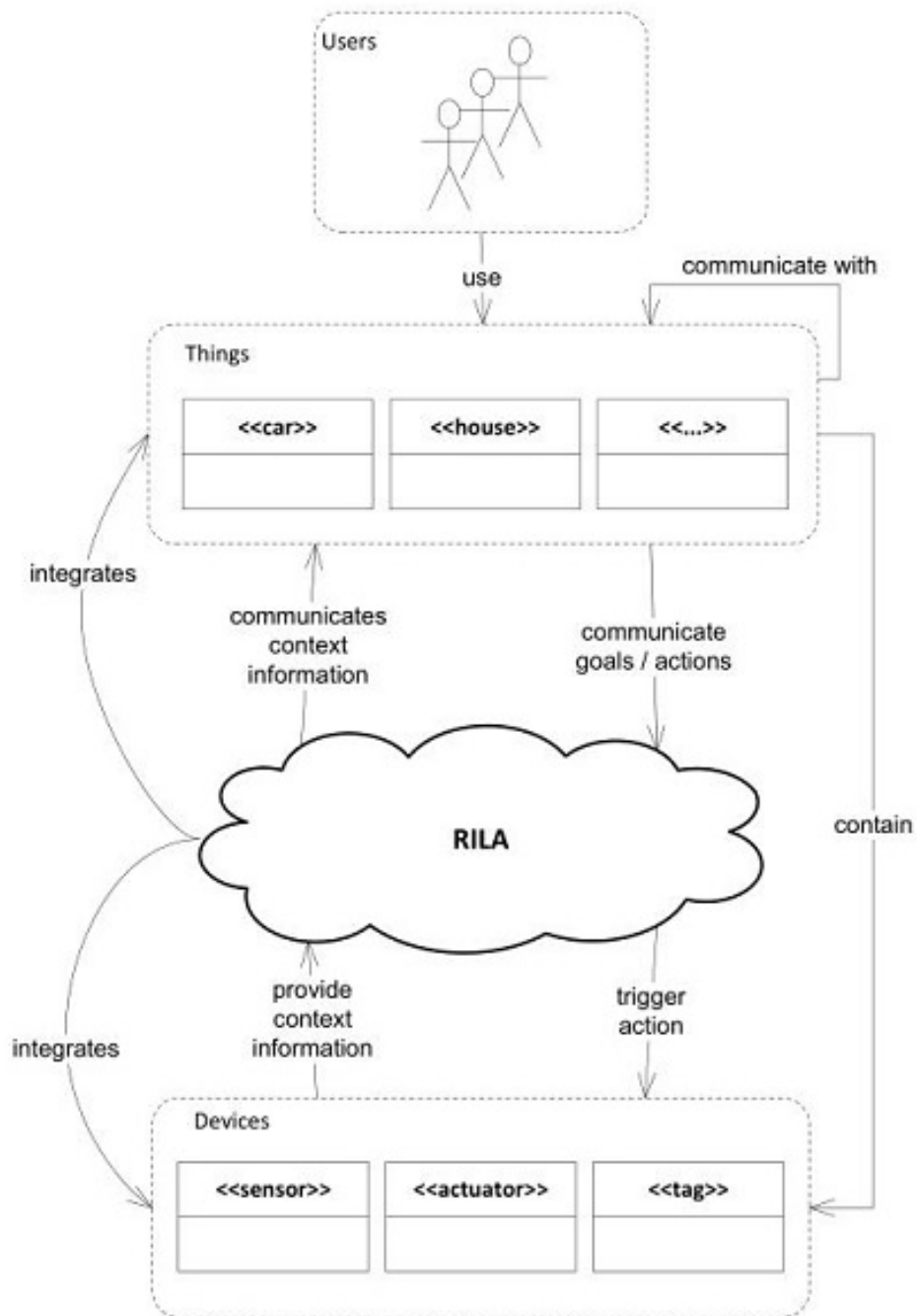


As mentioned earlier, certain context elements can be standardized right away (because they are already standardized) but others cannot (because they are use-case specific). In order to let a thing know, if it can actually communicate with another thing a certain communication standard has to be agreed on. For this purpose we introduce the context situation schema. The context situation schema defines what the thing is capable of in ways of context.

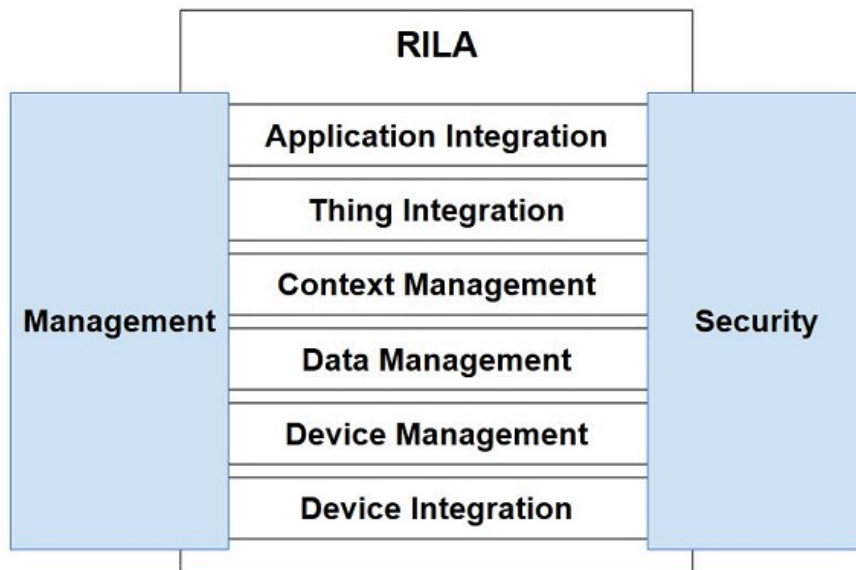
You can drive the context model further and define certain “standard functionality” that has to be introduced by everything and additional functionality that is to be defined by each thing as for example done by the [Z-Wave](#) standard.

Similarly to the context model you can also define an action model that defines what things can trigger (e.g. open a window, take a photo). Actions can only be triggered with the combination of context information (e.g. a context situation) and defined goals. Goals are usually depicted as rules of a rule engine (e.g. IF temperature > 25* THEN open window). Whenever a context situation is given to a thing the thing evaluates if an action is to be triggered according to its defined goals (i.e. rules). Depending on the use-case the context, action and goal model can be more or less complex for a certain thing. Some things might only consume actions and won’t even produce context information, while others will publish context information (or even goals) to be consumed by other things.

Now that we are on the same page about the role of context aware computing in the IoT we can jump right into the definition of our Reference IoT Layered Architecture, short “RILA”. In the IoT context RILA acts between things, devices and the user as shown in the following figure.

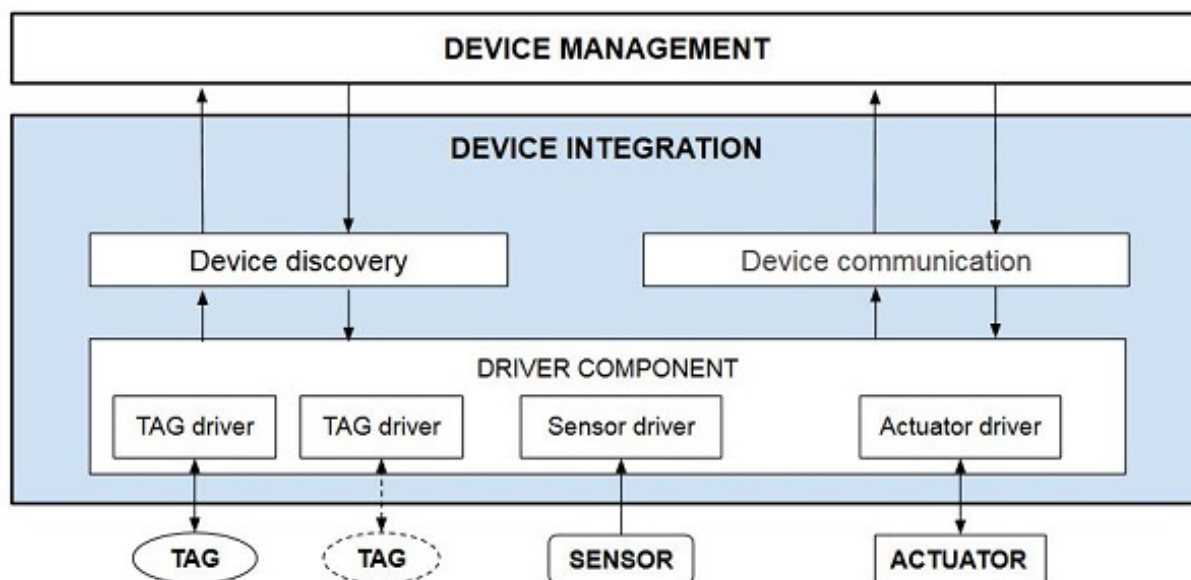


RILA consists of 6 layers. Besides these layers there are two “cross-section-layers” that affect all other layers, namely “Security” and “Management”.



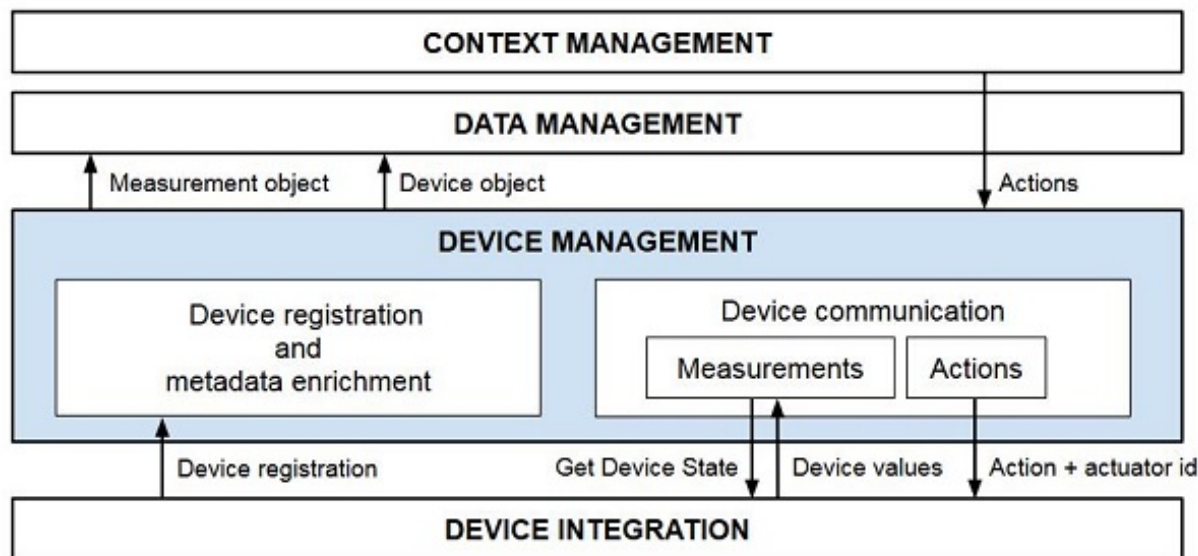
Let's take a look at each RILA layer and the component inside. We start from the bottom (Device Integration) and work ourselves towards the top.

The **device integration layer** connects all the different device types and consumes device measurements as well as it communicates actions (on device level). This layer can be seen as a translator that speaks many languages. The output of the sensors and tags depends on the protocol they implement. The input of the actuators is also defined by the protocol they implement.



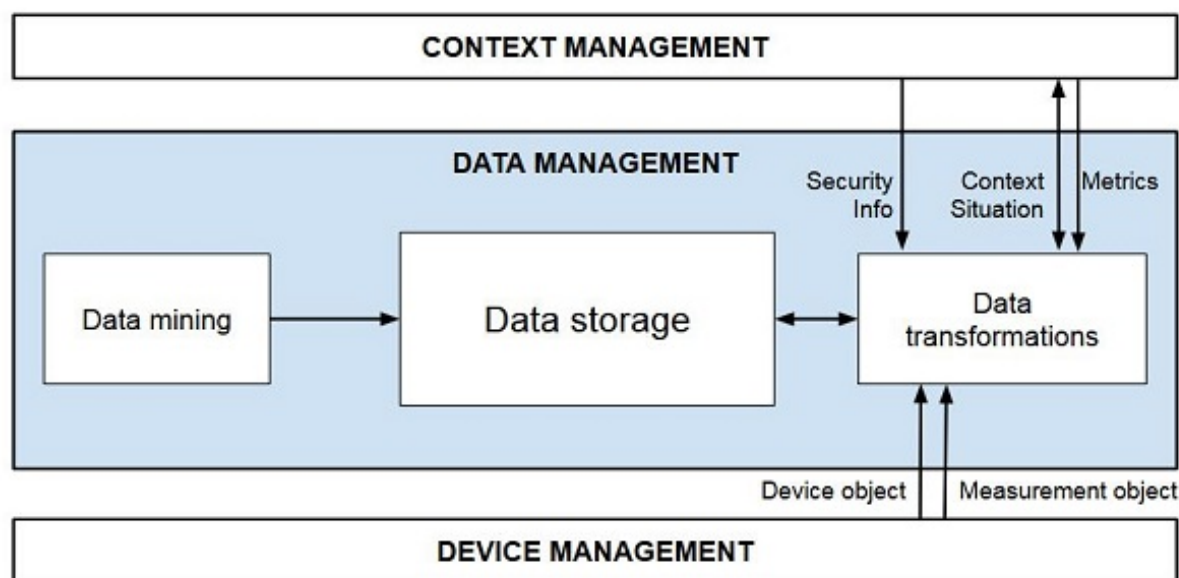
The device integration layer consists of three main components. The lowermost component is the driver component which communicates with the different sensors, tags and actuators on low level, vendor specific, communication protocols. It contains driver instances for every type of low level device known to the system. The next component is the device discovery component. It can be triggered by two events, one from the device management layer, which tells this component to add a new device and by the driver component, which notifies this component in case a new device is added. Similarly the device discovery also handles deregistration of devices. The last component is the device communication component. It is the bridge between the device management layer and the driver component. This component decides which driver is called when the device management layer addresses a device.

The **device management** is in charge of taking device registrations and sensor-measurements from the device integration layer. Furthermore it communicates status changes for actuators down to the device integration layer. The device integration layer then just validates that the status change (i.e. the action) is conform with the actuator and then translates the status change to the actuator.



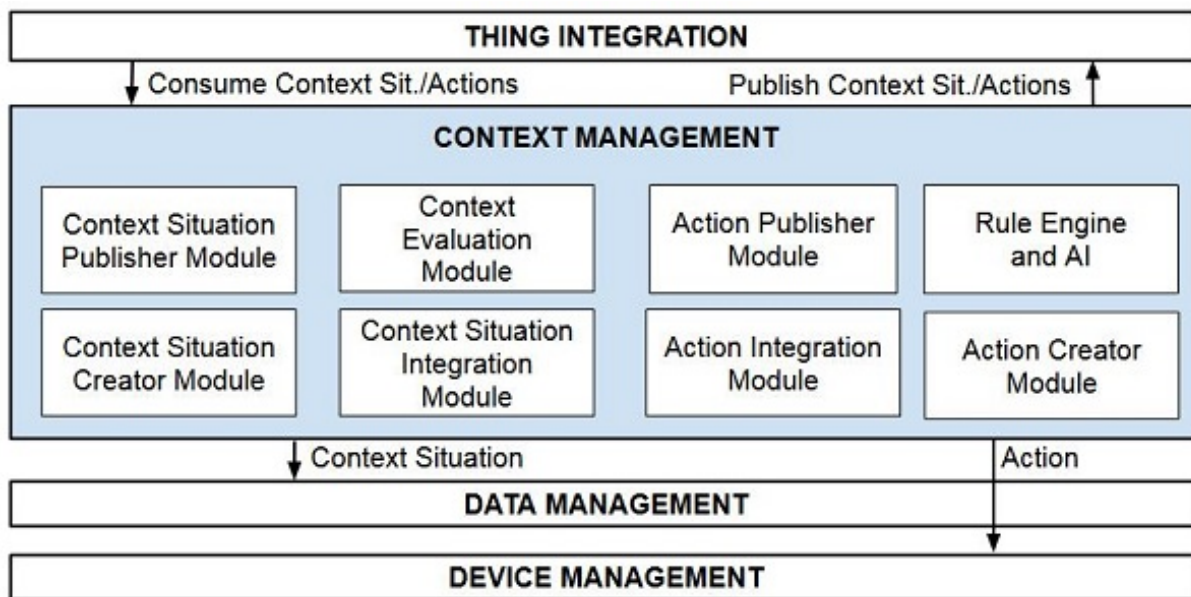
The device management layer is in control of the devices in the way that it knows which devices are connected to the system. Each change to a device's registration as well as incoming measurement data has to be communicated from the device integration layer to the device management layer, so the information can be updated and stored. That way the device integration layer manages device registration (which includes meta-data enrichment, as for example in which unit or frequency a sensor is sending data) and device communication (take the actual measurements and pass them to the data management, as well as pass actions down to the actuator-devices).

The **data management** can be seen as a central database that holds all data of a "thing", but this is only one possible implementation. For larger things within the system (e.g. a device life-cycle monitoring system collecting data from other things) data management might be a data warehouse or even a complete data farm. The implementation of the data management layer thus strongly depends on the use-case for the specific thing.

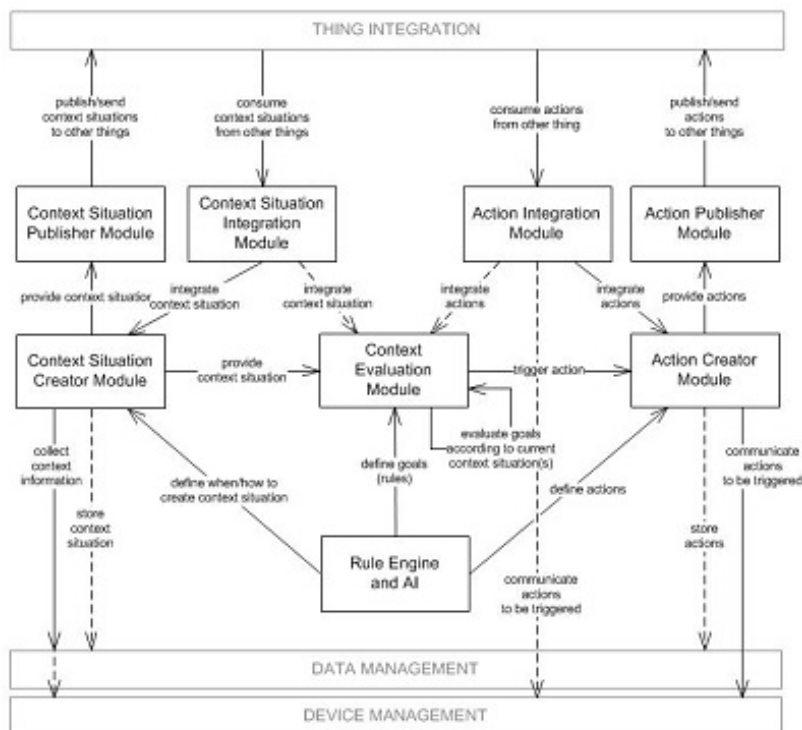


The **context management** defines the central business logic within RILA and is responsible for six tasks:

1. Define the goals of the thing.
2. Consume the context situation(s) of other things
3. Produce the (own) context situation of the thing.
4. Evaluate the (own) context situation towards the goal.
5. Trigger actions that help to fulfill the goal according to the evaluated rules.
6. Publish context situations for other things.



According to these tasks we can divide the context management into eight components as shown in the next figure.



Rule Engine & Artificial Intelligence (AI): Define and manage all of the rules necessary for context evaluation. This includes the goal (which is basically as set of rules) as well as rules for creating the context situation and actions.

Context Situation Integration Module: Listens to context situations of other things and integrates the incoming context situations.

Action Integration Module: Incoming actions of other things are evaluated and passed on to the device management layer by this component. Rules have to be considered, that define in which situations an action received from another thing can be passed on for triggering an actuator.

Context Situation Creator Module: Collects data from the system and builds the context situation(s). This can also be driven by rules.

Action Creator Module: Similar to the context situation creator module, action objects have to be created

once triggered during rule evaluation.

Context Situation Publisher Module: Provide context situations to the thing integration layer. According to the sophistication level of the implementation the context situation publisher can provide a set of context situations for different things that are subscribed or one context situation for everybody. The context situation publisher module has to take care of data permission levels towards other things. Only trusted other things should receive selected context information. Furthermore this module has to take care of defining the context situation schemas that are communicated to other things that want to subscribe. The schema is used to evaluate whether a thing is capable of communicating with another thing.

Action Publisher Module: Similar to the context situation publisher module this module is responsible to communicate actions to the thing integration layer to be communicated to other things. Additionally the action schema(s) are managed by this component.

Context Evaluation Module: Evaluates the rules using the (current) context situation and triggers actions that are communicated down to the devices or to the action creator module. The action creator module in turn passes the created actions to the action publisher that communicates the actions to other things. One way to simply evaluate rules is to build decision trees from the rules defined by the rule engine.

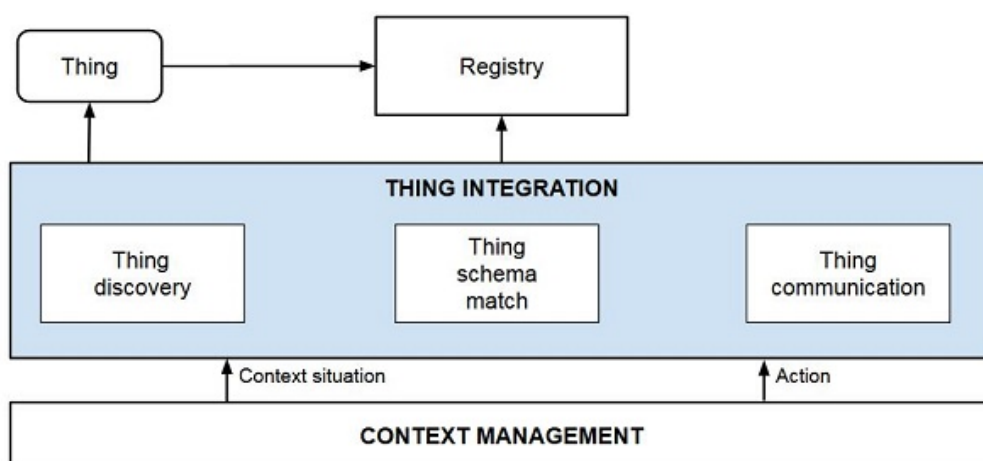
The concrete architecture and complexity of offered functionality strongly depends on the use case for the thing under development. Especially the rule engine & artificial intelligence component might not have to be very sophisticated for less intelligent things (e.g. a fridge). For things that collect context information from other systems these components will, however, be very sophisticated. Higher sophistication can be for example data science and data mining techniques.

The **thing integration layer** is responsible for finding other things and communicating with them.

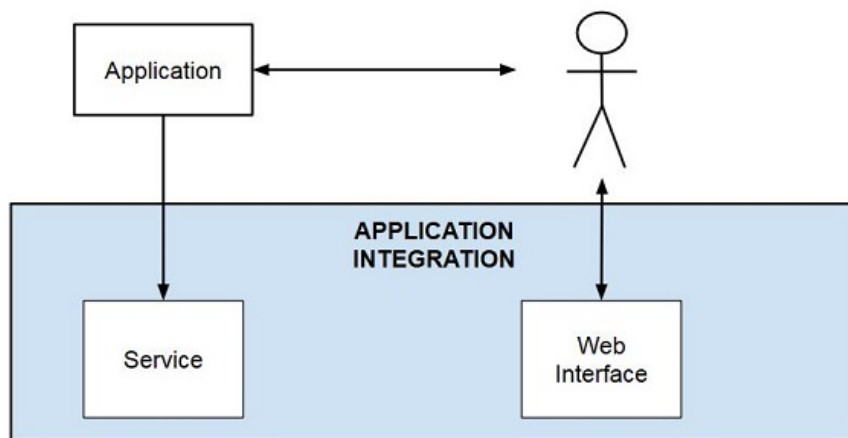
Once two things found each other they have to undergo a registration mechanism. The thing integration layer has to evaluate if the communication with the thing to be partnered with is possible. For this purpose the context situation and/or action schemata have to be compared. These are provided by the context management layer.

If the schema-match is evaluated positively, the thing can notify the other thing upon new context situation or action creation. The context situations and actions to be communicated to other things are provided by the context management layer.

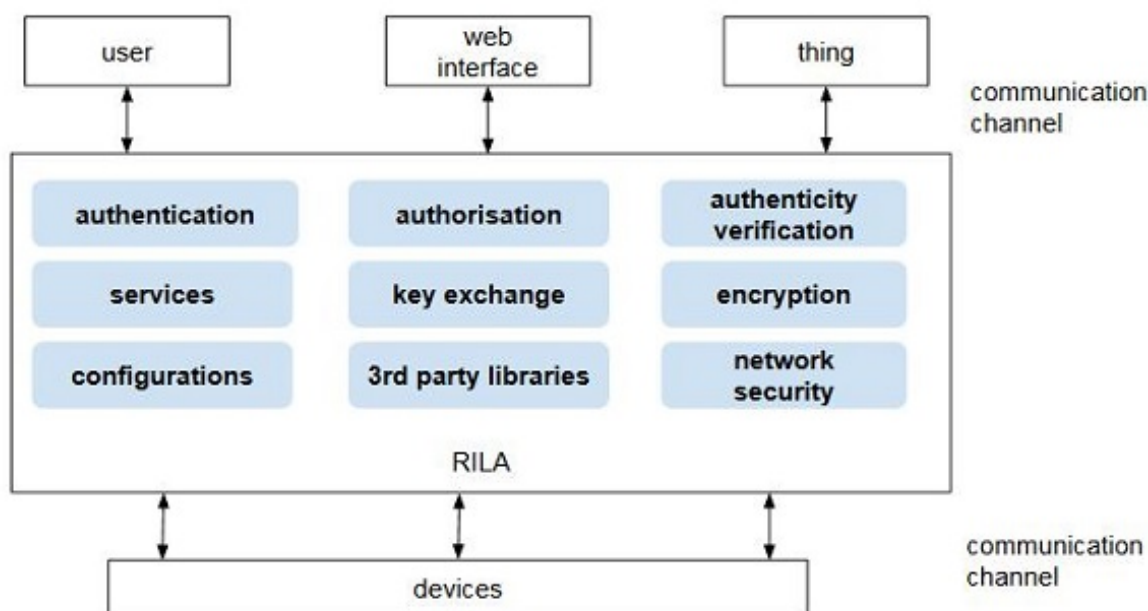
The thing registration can be done in a central component or by the thing itself (e.g. auto-discovery network scan).



The **application integration layer** connects the user to the thing. Applications that are (directly) on top of the RILA architecture are located here. The application integration can be seen as a service layer, or even as a simple UI on top of the stack. The concrete implementation of the layer depends on the use case.



At this point we are finished with the layers. Now let's remember the cross-layer challenges and take a look at security. When building an IoT system we have to consider security on all layers. Attack vectors have to be identified in order to identify appropriate security standards.



The following attack vectors can be identified:

User: The end user is a possible attack vector because it can affect the system either on purpose or without its knowledge. One common attack of this type is a phishing attack, that attempts to acquire sensitive information from the victim.

Web interface: If the application offers a web interface, then this can be subject to 'conventional' attacks like SQL injection or XSS. [OWASP](https://www.openwasp.org/) (Open Web Application Security Project) created a list of the 10 most likely attack scenarios against websites.

Thing: Smart devices often communicate with an external system via an app, which relies on a form of operating system. The two main liabilities are the app itself which might not offer sufficient security mechanisms or the operating system underneath that could be hacked or infected.

Low level hardware components: When considering hardware components and the security they provide, one must take their computing power into consideration. The main liability is represented by low power devices that simply do not have the required CPU power for secure encrypted communication. When working with many sensors, one can eliminate the outliers to get an accurate picture but security cannot be achieved. If the correctness of the data provided by the sensors is essential, then more powerful hardware is required, increasing the cost of acquisition by an order of magnitude.

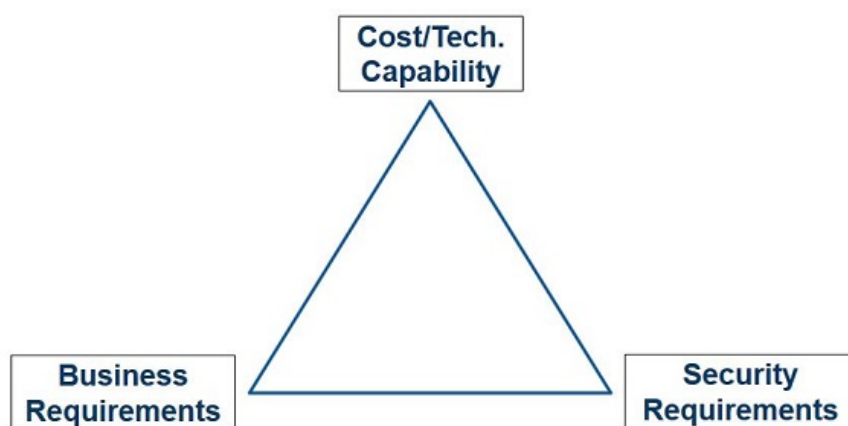
Communication channels: Securing the communication channel depends on the protocol used. We will discuss the protocols relevant for IoT and what they offer for securing their communication:

- **RFID and NFC:** Communication between tag and reader is wireless and can easily be eavesdropped. Encryption of the data is therefore essential. The symmetric encryption algorithms that are today considered sufficiently secure are 3DES and AES-128. When writing data to a new tag, the default authentication key should be changed. The key management for the tags is done by the system that controls the reader. RFID tags themselves are very diverse and security must also be taken into account when acquiring them. The Mifare Plus tag for example is an upgrade from the Mifare Classic tag, because it offers AES-128 encryption. The Mifare Classic tag uses a proprietary algorithm based on a 48-bit key which was already cracked.
- **Zigbee:** the communication channel between a [Zigbee](#) device and the application is secure because the algorithm used for encryption is AES-128. The initial key exchange on the other hand must be considered insecure. When a new device is added to the network, the key is sent to the device in plain text and can be found by sniffing, given the timing is right.
- **Thread:** the communication between two Thread devices is secured by an AES encryption. The key establishment between a new device and the application is done in a secure way using a key exchange algorithm.

Attack vectors can also be grouped into more technical attack vectors that target specific components of the system. These are:

- Authentication
- Authorization
- Authenticity validation: signature for messages
- Key exchange mechanisms
- Encryption
- Configurations – can pose a security threat from bad to default configurations
- Third party libraries – may contain security breaches and well known exploits when not updated
- Network security

The security-triangle shows us the dilemma of choosing the right amount of security according to our use-case.



The security triangle somehow represents a compromise that occurs in every use case. You can only choose *one* point inside the triangle that represents what you want or need in terms of security, cost and business requirements. Let us give you some examples:

Example 1. Acme Bank builds a bank vault: It is crucial to use secure hardware for this use case, that cannot be tampered with. In order to achieve maximum coverage on business and security requirements, the costs will increase a lot.

Example 2: Farmer Billy Bob wants to get some shiny sensors to know how his crops are doing on his

smartphone, but he does not need a lot of security. For now, farmer Billy Bob got his requirements fulfilled, had small costs and can live happily. Well, at least until farmer Jimmy Junior's son, Jimmy Junior Junior, went to study computer engineering ...

So, finding the appropriate security measures throughout the complete architecture is always a tightrope walk, as business requirements and costs often contradict with high security measures. Furthermore it can happen that certain technical requirements restrict us from using the highest security measures, as for examples low-power devices might not be able to accept a certain overhead when sending packages as this would result in consuming more energy.

At this point we want to finish the introduction of our reference architecture. In this article we wanted to show that it is possible to break down the IoT world to a more comprehensible level. Context aware computing techniques help to make certain parts of this world more understandable. In a follow-up article we will show how use cases can be derived from the RILA reference architecture introduced here to provide a more complete picture on how RILA can actually help us with the implementation of IoT systems.

About the Authors



Hannelore Marginean is a developer at Senacor Technologies in Germany. She likes to discover new innovative technologies and has spent time researching about the possibilities, the security risks and the benefits of IoT. In her spare time she likes to paint with acrylics and play the guitar.



Daniel Karzel is a technical consultant at Senacor Technologies in Germany. Already during his master's studies in Mobile Computing he was dealing with Context Aware Computing and the Internet of Things. Besides thinking about software architectures for the future he enjoys playing the accordion and traveling China in his free time.



Tuan-Si Tran is a software developer at Senacor Technologies in Germany. He is a frontend enthusiast and is interested in "bleeding edge" technology. In his spare time he likes to play tennis.

- Personas
- [Architecture & Design](#)

- [DevOps](#)
- [Development](#)
- Topics
- [Security](#)
- [Teamwork](#)
- [Distributed Teams](#)
- [Mobile](#)
- [Communication](#)
- [Internet Of Things](#)
- [IOT](#)
- [Agile](#)
- [Design](#)
- [Architecture](#)

Related Editorial

[Cellular Technologies Enabling the Internet of Things](#)

[Startup Afero Platform Addresses Internet of Things Communications Security](#)