systemum
Software Management Consulting

Overcoming the Challenges in M2M Software
The Lifecycle Perspective

Dr. Jörg-Volker Müller
ew2012@systemum.de

Systemum GmbH & Co. KG
+49 531 39045711
www.systemum.de

Rebenring 31
38106 Braunschweig
Germany

Abstract

Online applications have changed the way people think of computers. People's view on software as a locally installed product is replaced by a mixture of apps, content and online services. M2M (Machine-to-Machine) helps to connect devices, machines and even cars to the already connected world, blurring this mixture even further.

The components that make up a complete M2M solution constitute a highly complex system where software plays the central role. A solid management of software components, architectures, processes and people is essential to meet the challenges of M2M solutions throughout their lifetime.

This paper explores the role of software in an M2M world and highlights the different lifecycles of the systems involved. It will also provide guidelines for developing successful and long-lasting M2M software components and services.

Introduction

Since the invention of computer networks in general and cellular networks in particular the industry has begun to connect machines on the factory floor with control systems, equip service personnel with mobile devices to get access to company data while on the move, and interconnect cars with each other to enhance traffic flow and prevent accidents.

In this paper, the term M2M or Machine-to-Machine will be used in a broad sense: Connecting different systems, especially mobile systems and mobile and immobile computers to enhance the scope of operation. The typical usage scenarios are:

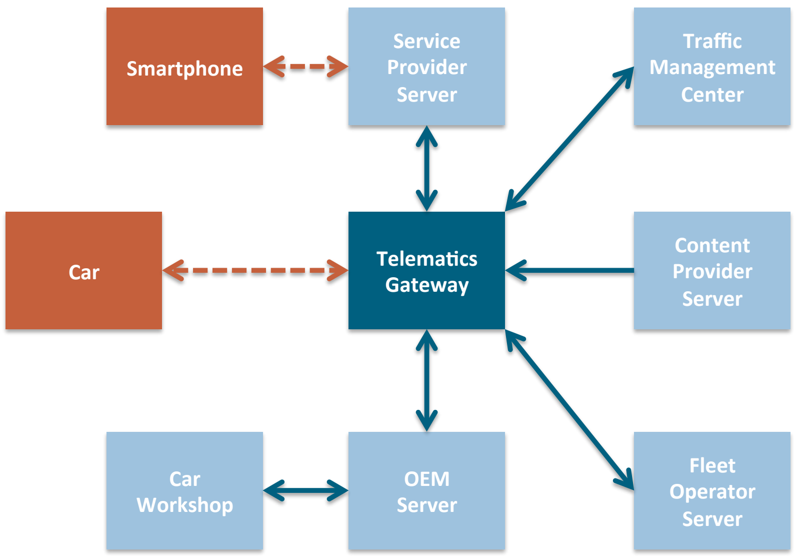
- **Automotive**: In Vehicle Telematics cars connect to each other in order to give warnings like “slippery road condition” or “traffic jam” to passing or following cars. Service centers get access to vehicle information in case of a breakdown or an emergency. Modern infotainment systems in cars connect through cellular networks with telematics centers giving the user access to traffic information and other content.
- **Tracking & Tracing**: In distribution and logistics, companies determine the current and past locations and other information of vehicles, containers or products.
- **Remote Maintenance & Control**: The owner or the manufacturer of a machine is able to monitor its operational state or to do a preventive maintenance in order to prevent problems like breakdown or wear & tear.

- **Remote Metering**: Consumption, diagnostics, and status data from water meter or energy metering devices is automatically collected and transferred to a central database for billing, troubleshooting, and analyzing.
- **POS & Payment**: Terminals at the “Point of sale” (POS) perform a payment transaction like charging a credit card. Waitresses use a wireless POS terminal to take the customer’s orders.
- **Mobile workforce management**: A mobile worker is equipped with a mobile device and has access to customer data or maintenance information of a machine he has to repair.
- **Healthcare**: M2M communication is used to provide clinical health care at a distance. Medical, imaging and health information data from the patient is transmitted to a central monitoring system. Medication can be controlled without direct access to the patient.
- **Security**: Alarm, surveillance and access control systems are connected to security personnel via M2M communication in order to protect buildings, companies, people and data.
- **Smart Grid**: In the future, we will experience a mixture of these scenarios, when electric vehicles will be connected to the power network, which uses the battery of the car as energy storage. The user can interrupt this process with her smartphone when she needs the car.

This list is far from complete but gives an idea which scenarios to consider when thinking of the challenges in the M2M world.

The Structure of an M2M System

In many scenarios, the components of the overall M2M system are not situated at one company where the IT people have everything under their control. Usually, mobile units are located at a machine at the customer’s site, or embedded in a car on the road. In other scenarios different companies deliver different services, like one providing map data for navigation purposes enhanced by the other with a navigation service.



Picture 1 Typical Players in the Car Telematics Game

Picture 1 shows the typical subsystems that may connect to a car in a vehicle telematics and infotainment scenario. Such a subsystem may be located at one of the participating companies that play a role in the scenario. A service for a certain usage scenario consists of several components that may be distributed to some or all of these subsystems.

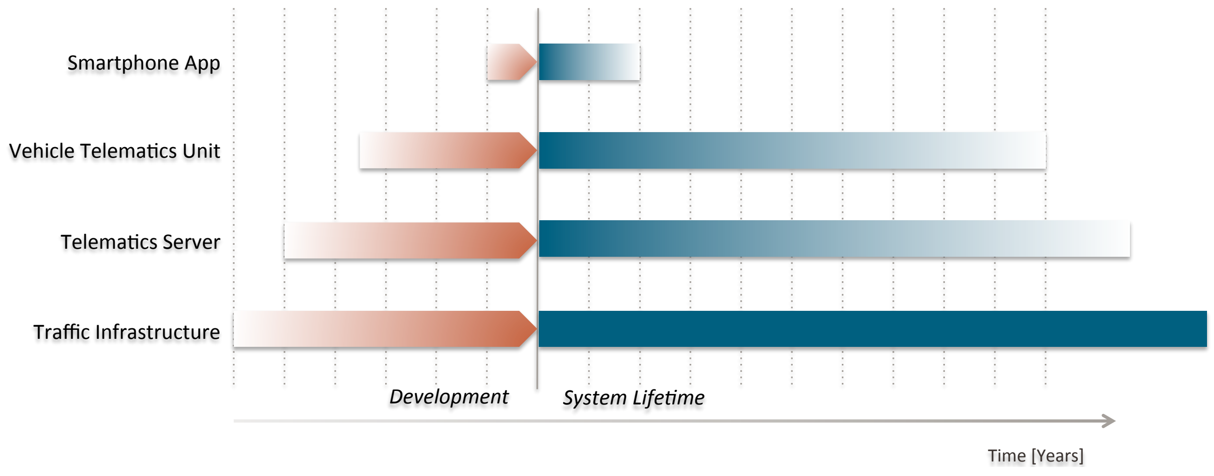
Take as an example a service to monitor the battery status of an electrical vehicle (EV): It may consist of

- a component in the *car* connecting to the battery control unit and transmitting the battery status to
- a central routing component within the *telematics gateway* which has a trusted data connection to the car and routes the information to
- the provisioning component at the *service provider* offering the service for the end user. The service provider hands over the data to
- the mobile app running on a *smart phone* which displays the status to the user. On the other hand the information may be given to
- a data mining server component at the manufacturer *(OEM)* of the car who monitors the status statistically and hands over certain soon-to-fail cases to
- the CRM component at the *car workshop* where an agent calls the user to get his car's battery repaired.

This is just a simple example that shows, that the complexity of an M2M system results from connecting different components that have to interact over the lifetime of all of the participating subsystems. These components are implemented in software. The software components have interfaces and communicate with each other using transmission protocols.

Life Cycles Differ

The systems involved in an M2M scenario differ in their expected development times, lifetimes and corresponding life cycles. A smartphone app, for example, will usually be developed in a few months. It will be in the market for a few years, undergoing several major or minor updates during its lifetime.



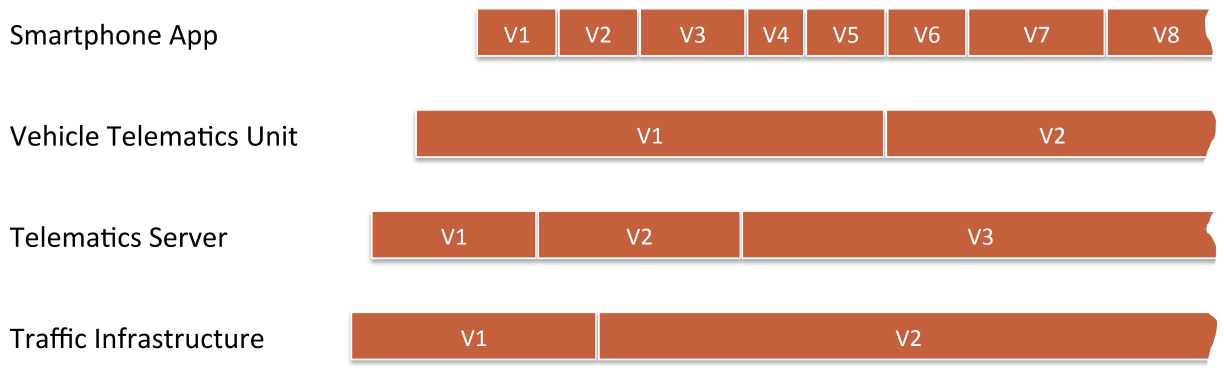
Picture 2 Typical life cycles of M2M components

Traffic lights on the other hand are installed to live for 20 years or more. Even if they were connected digitally to the traffic center management, updating is usually done via laptop on-site due to safety requirements.

Picture 2 shows some examples for differing life cycles.

The challenge is to develop components in such a way, that they are able to fulfill their purpose during the lifetime of the system. The subsystems may belong to different companies and may not be able to be updated remotely. Therefore, a deployment of a service in one step or an update on every component in the system is not possible in a typical scenario.

This leads to the conclusion, that all of the components must be built in such a way that they can work efficiently together over the lifetime of the system (interoperability). Different versions must be able to interact with each other (see Picture 3).



Picture 3 Version differences in M2M services

As a consequence,

- the resulting components must carry a version number which gives the communication partners the possibility to react to differing functionalities.
- the communication protocol must carry a version number as well in order to leave room for improvements and enhancements.
- the components must be able to handle older versions of components and protocols (backward compatibility).
- it is good to provide a mechanism to lookup features of components and protocols (meta data) in order to react to future improvements properly.

Besides improvements in the software there might be new requirements from changes to the underlying hardware, which lead to changes in the functionality of the component as a whole. The release management of the software should consider hardware versions as well.

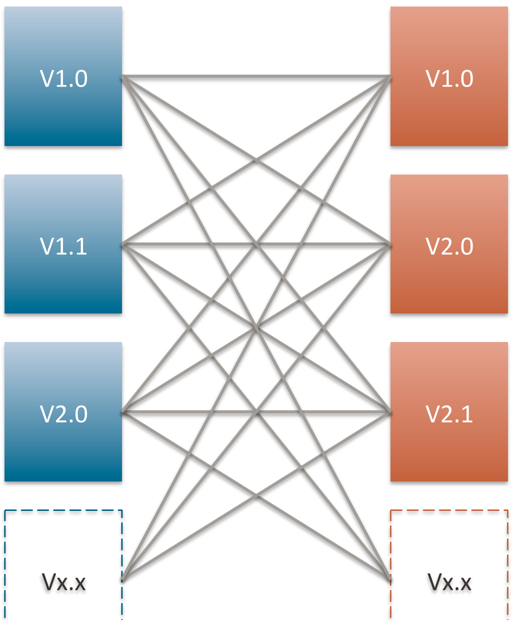
Interoperability Complexity

When providing backward compatibility components are designed to cooperate with older versions of the corresponding component on the other side of the M2M communication. This leads to very complex software code, which has to provide handle every combination of protocol version and component version. Picture 4 gives a short example on how this code might look like.

```
if protocol.version = "1.0" then
    if component.version = "1.0" then
        ...
    elsif component.version = "1.1" then
        ...
    else
        ...
    end if;
elsif protocol.version = "1.1" then
        ...
else
    error("Cannot handle protocol version");
end if;
```

Picture 4   Example code showing the handling of version differences

This will lead to a very high complexity due to interoperability. Picture 5 shows the exponential growth of combinations when both components will be enhanced and extended.



Picture 5   Interoperability requirement results in high complexity

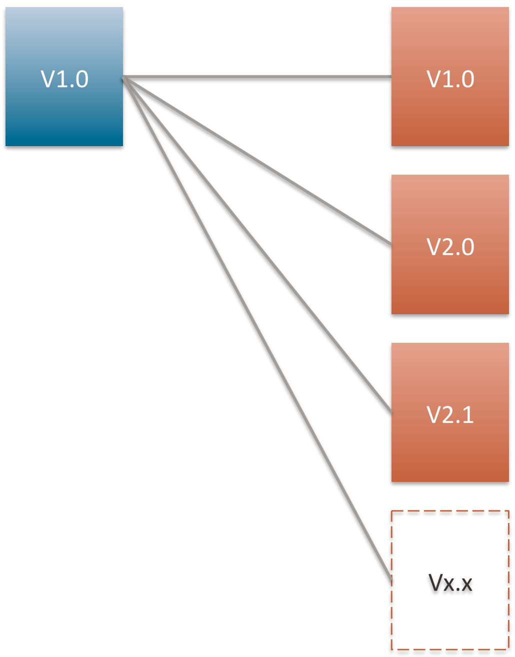
In order to reduce this interoperability complexity, the main precaution is to reduce the number of versions and design the interface of the components wisely.

Probably the most important rule in software architecture, *loose coupling*, is the key factor here: Loose coupling means interconnecting the components so that those components depend on each other to the least extent practicable. Coupling refers to the degree a change made within one component (or a protocol) demands for changes within other components.

In M2M systems, components should be designed as isolated, simple services with a small interface. This leads to a stable component, which does not have to be changed very often. The usage of complex services, especially in a remote subsystem like a machine at the customer's site, should be avoided. The example is to use `getCurrentPosition` – a generic function providing the position of the object under control – instead of a specialized `vehicleTracking` function, which does some more vehicle-specific functions.

Thinking end-to-end in M2M solutions is a prerequisite for an M2M design team. This should not lead, however, to a monolithic design of components that are heavily depending on each other. The communicating components should make use of a generic interface instead, which does not depend on their counterparts (loose coupling). For example, use `getEngineStatus` instead of a specialized `getMainBatteryTemperature`.

Picture 6 shows the reduced complexity when a simple component is designed on the left side, which does not need to be changed over the lifetime of the system.



Picture 6 Reducing the complexity with simple, generic components

When designing flexible components that are able to handle many different versions of the corresponding components it is helpful to use meta data enquiry mechanisms at runtime. The component can query its counterpart for the availability of certain functions instead of just asking for the version number and handle the differences with hard-wired code.

When more than one company is involved in the M2M system, standards should to be developed for message protocols, basic services on the device side, and basic services on the infrastructure side. This results in the standards committee managing functionality changes. Overall interoperability is improved.

Guidelines for M2M System Development

Developing M2M systems is a challenging task, because development teams from different domains must cooperate to build the system:

- IT software specialists develop the backend infrastructure. They have lots of computing power at hand, but must handle issues like load balancing, data storage, security and so on.
- Embedded specialists are responsible for the “machine” side. They have limited computing and memory resources and cannot afford, for example, to parse huge XML texts on the fly.
- Mobile device developers design apps for smartphones and tablets. They are focused on optimizing the user experience and handle the GUI with design elements, animations and proper presentation of information.

In order for these development teams to work together, it is important that everyone is able to think end-to-end and take into account what the colleagues of the other domains do.

Misunderstandings will surely happen, because of the different perspectives on performance and memory issues, different tools and different architectural views on software. Designing a solid end-to-end architecture, building project glossary and maintaining this documents throughout the lifetime is a very good prerequisite to develop a common language for everyone involved.

Because some issues like security and transmission protocols are important for everyone in the team, the best people with the most experience in the given area should design the architecture together. For example, the IT specialists know a lot about security and firewalls. This might help the embedded team to find the appropriate security solution for their subsystem.

Testing M2M Systems

The interoperability complexity of M2M Systems leads to an exponential growth of possible error sources. Therefore, systematic testing is mandatory throughout the lifetime of the system.

Testing the final system can only verify one combination of versions and can only verify the positive functions. An important factor in distributed systems, however, is the handling of transmission errors, broken communication links or sensor defects. Because not all of these error conditions can be provoked in the real world, testing is usually performed by simulating error conditions.

The test strategy should include reference implementations for some or all participating subsystems that have certain test modes where error situations can be simulated and positive and negative tests can be performed.

With every improved version of a component or a protocol the complexity grows. Therefore, automatic testing is a must. Tests for older versions can be reused to

- test the compatibility with the old version and
- can be copied and modified to match the current version.

Using such a growing automated testing framework, the correctness of the overall M2M system can be ensured throughout the lifetime of the system.

Conclusions

Finding the appropriate architecture for M2M solutions is a challenging task because the software components are distributed over devices, IT-systems and even companies. Due to the differing life cycles and continuous enhancement of the software components, backward compatibility is a must.

The resulting interoperability complexity can be reduced by finding generic components with simple interfaces, providing meta data enquiry mechanisms and standardizing services and protocols.

The development of M2M solutions has to take into account the various technical domains that are involved. People with different mindsets, technologies and tools should learn from each other to get a working end-to-end solution. Testing of M2M systems should be done automatically by using reference implementations with special test modes.

Designing M2M software requires experienced architects that are able to handle the challenging task of finding the right solutions for components, interfaces and protocols in order to obtain a successful and long-lasting M2M system.

References

Robert G. Cooper (2001) Winning at New Products.
New York, NY: Basic Books

Luke Hohmann (2003) Beyond Software Architecture, Creating and Sustaining Winning Solutions.
Boston, MA: Addison-Wesley

Kent Beck, Erich Gamma, David Saff: JUnit Test Infected: Programmers Love Writing Tests.
http://junit.sourceforge.net/doc/testinfected/testing.htm

SearchNetworking – Definition *Loose Coupling*
http://searchnetworking.techtarget.com/definition/loose-coupling