STORY

added value of STORage in distribution sYstems

Deliverable 3.6
Report on interoperability guidelines

Revision..... 1
Preparation date .. 2016-10-31 (m18)
Due date 2016-10-31 (m18)
Lead contractor.... UCL
Dissemination level PU

Authors:
Paul Valckenaers. UCL
Nicolas Jordan..... ACT
Olivier Hersent..... ACT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

STORY

Table of contents

1 IN-DEPTH INTEROPERABILITY (EXECUTIVE SUMMARY) 4

2 INTRODUCTION 5

3 WHERE DOES IT GO WRONG? 8

3.1 EASY INTEROPERABILITY 8

3.2 PROBLEMATIC INTEROPERABILITY 9

3.3 WHY DO WE EXPECT MORE THAN WE GET (FROM INTEROPERABILITY)? 10

3.4 REMARKS 10

4 WHEN TO INTEROPERATE (AND WHEN NOT) 11

5 SYSTEMS AS COLLECTIONS OF RESOURCES 13

5.1 AWARENESS OF THE EMBEDDED RESOURCES AND THEIR ACCESSIBILITY 13

5.2 EXPLICIT AND MANDATORY RESOURCE MANAGEMENT 15

5.3 PRO-ACTIVE RESOURCE MANAGEMENT 16

6 ACTIVITIES USING EMBEDDED RESOURCES 17

6.1 THE VALLEY OF DEATH 20

6.1.1 Autocatalysis for manmade systems – critical user mass 20

6.1.2 The simple side of the valley of death 22

6.1.3 The sophisticated side of the valley of death 22

6.1.4 Inside the valley of death 22

6.2 MAINSTREAM TECHNOLOGIES AND HOW THEY EVOLVE, APPEAR AND DISAPPEAR 23

6.2.1 Language popularity 24

6.2.2 Adopting not-yet-mainstream technology 26

6.3 NON-MONOLITHIC SYSTEMS 28

7 INTEROPERABILITY OF EXISTING SYSTEMS AND TECHNOLOGIES 28

7.1 DAMAGE ASSESSMENT OF AN EXISTING SYSTEM/TECHNOLOGY 29

7.2 DAMAGE CONTAINMENT 30

7.3 SORTS OF PROGRAMMABLE EQUIPMENT 30

7.4 DISCUSSION OF SAMPLE TECHNOLOGIES (TO BE REVISED AND EXTENDED) ..... ERROR! BOOKMARK NOT DEFINED.

8 DESIGN FOR IN-DEPTH INTEROPERABILITY 32

8.1 REFERENCE ARCHITECTURE PART 1 – RESOURCES 33

8.1.1 Identification of the (embedded) resources 33

8.1.2 Reflection of resource (class/type) capabilities 33

8.1.3 Management of resource (instance) capacity/availability 36

8.2 REFERENCE ARCHITECTURE PART 2 – ACTIVITIES 38

8.2.1 Simple devices and services 38

8.2.2 Programmable devices and systems – lacking critical user mass 39

8.2.3 Programmable devices and systems – enjoying critical user mass 39

8.3 ARCHITECTURAL PATTERNS – PROACTIVE COORDINATION 41

8.4 REMARKS 41

9 CONCLUSIONS 42

10 ACRONYMS AND TERMS 43

11 REFERENCES 44

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

1 In-depth Interoperability (publishable executive summary)

This document addresses in-depth interoperability, translating and applying scientific insights going beyond the state-of-the-art [1]. It contains insights and guidelines on how to design and build in-depth interoperable components and subsystems that can be integrated with little effort and time. The guidelines allow the integration of components and subsystems – through interoperation – into an overall system while ensuring good performances. In addition, it addresses the assessment of the in-depth interoperability capabilities of existing systems as well as the containment of their eventual inherent incapability of in-depth interoperation.

Among others, arbitrary design choices are revealed to be the root cause for intrinsic interoperability issues. Examples are found in gatekeeper control systems monopolising access to valuable resources. They only offer their services whilst making narrow assumptions. This often reveals to be unfortunate when the assumptions are not complying with interoperability demands. Explicit resource allocation remedies such an issue. It minimises and contains the impact of gatekeepers preventing adequate interoperability. This type of processes constitute a first part of the answers discussed in this document.

Perhaps more important are the manners in which services are provided without making such potentially harmful assumptions. A system (of systems) architecture, design patterns and design guidelines enable and facilitate this. Reflection is a sample service provided in this manner. It refers to resources being able to provide information about their status and capabilities without being responsible for managing or controlling itself. It allows other systems to discover, assess and use the resources in a smart grid. Importantly, the resulting system elements enjoy longevity and maximise their user mass potential.

An example of containment of interoperability issues within existing systems is the device driver approach. When a device lacks the critical user mass to be a stand-alone system, its resources can be used to turn it instead into a peripheral device attached to a mainstream platform by means of a driver. This permits to develop and, more importantly, maintain it in an environment where talented professionals are readily available.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

This document covers the in-depth interoperability issues and answers – illustrated in the above examples – comprehensively. It connects this to fundamental insights, which determine what is (not) possible analogous to Carnot's Laws when designing turbomachinery.

2 Introduction

By necessity (i.e. when focusing on 'in-depth'), this document discusses in-depth interoperability on a high level (i.e. it does not address detailed and concrete IT matter). Mostly, it focuses on the relationship between ICT and relevant parts/aspects of reality (i.e. the so-called world-of-interest). The discussion mainly presents design principles and scientific insights (analogy: the second law of thermodynamics does not prescribe how to build a steam turbine but it is very relevant nonetheless).

This document provides insights and guidelines for STORY developers concerning interoperability. It does not impose rules but provides insight in trade-offs, pitfalls and opportunities. The authors will be available for discussion, clarification and brainstorming regarding the content of this document. Indeed, experience revealed that initial hurdles in understanding and mastering its contents are easily resolved in a conversation (whereas writing more extensive clarifications fails to do so).

Interoperability remains ill-understood at the present time. Interoperability and the accompanying standardisation receive lots of credit (and funding) because of the perceived potential benefits. However, achieving those benefits is characterised by large variations, ranging from smooth interoperation with negligible drawbacks – relative to what a fully integrated design might achieve – toward complete failure despite enormous efforts and investments as well as considerable (perceived) potential benefits to society. The achieved successes (and the costs of dealing with a multitude of incompatible systems) are ensuring this high amount of importance attached to interoperability and standardisation for the foreseeable future in our society.

However, many interoperability and accompanying standardisation efforts are failing and/or delivering underwhelming results for good reasons, which do not include poor performance by the people involved. When developing artefacts (especially larger complex systems), some universal

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

principles apply. These principles are like scientific laws (e.g. Newton's law on gravity, Carnot's law on irreversibility) and, therefore, unavoidably true whenever their conditions apply (e.g. Newton's laws do not apply close to the speed of light or at sub-atomic scale but, in everyday life, an engineer cannot ignore them).

This document introduces such ''laws of the artificial'' and elaborates the implications for interoperability in a smart grid, especially addressing energy storage. Overall, these universal principles add plausibility to the statement that (almost all) interoperability successes have solved easy problems, where efforts with mixed results (or failure) had no chance of success from the very beginning. In fact, the present landscape consists of:

- Easy interoperability, typical for the current interoperability success stories, which may benefit from coordinating and supporting actions but definitely do not warrant spending any public funding assigned to (research and) innovation actions. Here, failure is not caused by technical or scientific aspects; willingness to interoperate by sufficient stakeholders is the (sole) key success factor.
- Unsolvable interoperability, where conditions simply do not allow having interoperability with less effort than its potential benefits.
- Problematic interoperability, where the best possible outcomes will heavily compromise the achievement of the potential benefits (or the perceived potential). This kind of interoperability will be a considerable source of frustration for the people that have to live with it; their boss, customers, sponsors... believing interoperability to be a fact and assigning any underwhelming results to their workers/developers (who are unable to defend themselves).

In view of the above, this document *provides information and knowhow to be able to distinguish* the above categories of interoperability situations/challenges. A better understanding in this respect will *prevent wasting effort and resources on the wrong problems with the wrong approaches* (analogous to Carnot's law stopping attempts to build a perpetuum mobile).

Moreover, problematic interoperability situations can be addressed, to considerable extents, during the development of the systems that need to interoperate (but not afterwards). In other words, damage is inflicted while designing systems and a subsequent interoperability effort cannot undo

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

this damage. Hence, *design for interoperability* needs to be addressed while designing systems, and cannot be deferred until a system is operational. Note that such damage can be inflicted while designing a standard to address interoperability (in fact, this is one of the more likely places for inflicting damage resulting in underwhelming interoperability benefits).

This document elaborates ***a*** *systematic approach to* (1) identify when and where this interoperability compromising[1] occurs and (2) *design systems that maximise their inherent interoperability capabilities*. For existing systems, this involves (a) damage assessment and (b) damage containment. For system development, this involves applying a (reference) architecture and architectural patterns as well as design guidelines. Generic (software) system components will be developed (in subtask T3.4.2) to facilitate developing such interoperability-enabled systems.

The in-depth interoperability discussion in this document, mainly originating from fundamental research (UCL), has been augmented by coverage of industrial developments (ACT). This discussion addresses the ICT itself (e.g. a resource will be information) whereas in-depth interoperability focuses on the relationship between ICT and real-world elements (e.g. a resource might be a pump).

The industrial developments discussion reveals that a lot has been addressed and solved already where pure ICT is concerned (e.g. ETSI M2M); the remaining work-to-be-done is to ensure adoption and penetration of these more-than-adequate solutions/technologies. However, these solutions originate from outside the industrial automation and smart power communities, which implies that significant amounts of effort and time will be needed to achieve this. The pure ICT developments that are put forward comply – to a high extent – with the (mostly basic) in-depth insights that are relevant (e.g. concerning stateless/full-ness).

The industrial developments have initiated work on semantics/… but results remain mostly 'working documents' and 'prototypes'. Here, the developments 'are touching reality' and the in-depth interoperability insights become relevant indeed. Especially, accounting for what is possible/impossible is required as well as recognising that resources and activities are both 'first class citizens'. One cannot be an attribute/subcomponent/… of the other. Likewise, distinguishing

[1] E.g. when a controller denies external access to embedded sensors, which would be useful for another application.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

types (capabilities) from instances (capacity and state) is important. Straightforward ontology developments may not cope/suffice.

Within the STORY project, this document serves foremost to disseminate the in-depth insights. Furthermore, containment strategies – e.g. the device driver approach – will be applied and investigated within (suitable) test cases. Generic strategies will be investigated and applied in suitable developments in WP4 (e.g. self-description of devices, explicit resource allocation, explicit separation/modularity for resources and activities). Lessons will be learned concerning opportunities and limitations for the application of the scientific insights. Deliverable D3.7 will reflect those lessons learned and deliver more advanced services (e.g. prediction).

The industrial developments will be pursued in the ACT-led demonstration case. In other test cases, existing/legacy installations and the equipment requirements may/will prevent this. As said, the industrial technology has been developed but needs time and effort to penetrate.

3 Where does it go wrong?

This section pinpoints the mechanism and conditions that make interoperability problematic.

3.1 Easy interoperability

Some interoperability solutions are straightforward if addressed by knowledgeable people. For instance, converting polar coordinates (R, Θ) into Cartesian coordinates (x, y) and vice versa is easy when mastering the mathematics involved. The amount of expertise that is required has no impact concerning the easiness of interoperability. E.g. few people master the mathematics needed to convert quaternions into yaws, pitch, and roll values; indeed, most people never heard about quaternions. What is important is that this conversion is fully specified and choice-free, as was the conversion of polar coordinates into Cartesian ones. They are easy, concerning interoperability, because there is *no arbitrariness* involved.

Moreover today, systems interoperate successfully using standards such as ASCII, UNICODE, or XML, which have significant amounts of arbitrariness in their design. However, they achieved high degrees of adoption, mostly by being first (cf. autocatalytic sets below). And, their *arbitrariness*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

has negligible impact on real-world performance. E.g. XML is not efficient at all (i.e. needs to use lots of bits relative to the information contained).

Law of the artificial (paraphrased): *non-trivial systems belong to autocatalytic sets*.

Members of an autocatalytic set, when they exist, decisively improve the probability that more/stronger members of their set will exist in the future (examples in nature: weeds, insects and rabbits). Artefacts must be members of a set that contains themselves and a critical user mass. This creates **a** self-reinforcing cycle (autocatalysis) in which both economic resources and information are generated, making the artefact successful and populate/dominate the world in which it resides.

A property of autocatalytic sets is that achieving autocatalysis first/early normally is decisive. Such early autocatalysis allows the set members to consume/grasp the available resources (i.e. the market, the users) and deny potential competitor autocatalytic sets, arriving later, from becoming successful as well. This phenomenon is known as lock-in (e.g. into legacy solutions). It has a positive contribution in building critical mass allowing for sophisticated systems offering advanced services. It has a negative contribution, locking society into poor early design choices (i.e. good enough at the time of this early design).

3.2 Problematic interoperability

Consider noise levels expressed in dBA and interoperability. It does not get much simpler: scalar values. However, if systems expressing their involvement in managing noise levels exclusively stick to dBA, and the challenge is demanding regarding noise, the situation will be problematic.

Problems will range from mildly problematic – in disco's, the success of music depends on making the loudest sounds within the noise envelope allowed, expressed in dBA – to seriously problematic. For instance, the low frequencies will be undervalued in a context of structural noise isolation. If noise source limitation may only be expressed/communicated in dBA, numerous noise-generating

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

activities will be forbidden for no reason (e.g. young kids emitting mostly higher frequencies) or noise isolation may fail (i.e. low frequency noise is allowed but not effectively blocked and people may not sleep well).

The key point is that dBA contains *significant arbitrariness having considerable impact*. The weights assigned to frequencies basically make assumptions that often will not be true. This issue exacerbates rapidly when developing more complicated and sophisticated systems. Indeed, without a suitable approach and in-depth understanding, it comes increasingly more difficult to avoid arbitrariness when developing high-level services and functionality. This document targets precisely how to deal with this arbitrariness.

3.3 Why do we expect more than we get (from interoperability)?

When experiencing frustration concerning interoperability, the main source of our discontent is the observation that everything that is needed is available, in principle, but the access roads are blocked, inhospitable or non-existent. Here, there are (at least) two levels of non-cooperativeness.

First, there is the issue of having to solve and address the same problem over and over, once for every system/brand/home/… This appears to be solely a matter of cost (of developing and maintaining multiple solutions for the same problem). However, as will be explained later, there are far more serious issues in this respect.

Second, there is the issue of having no access at all unless the provider of the other system cooperates. Here, this cooperation often comes with a high price (e.g. you have to transfer your knowhow and see the market-dominating provider offer your solution in his system to your competitor).

3.4 Remarks

The above leads to a conclusion that interoperability efforts and accompanying standardisation will benefit from an enhanced comprehension of what is happening, what is possible/impossible… That's precisely where this document aims to contribute. It will transfer, translate and apply existing scientific/research results into the STORY innovation action [1]. In doing so, the objective is to reduce the number and size of doomed interoperability and standardisation efforts, which are

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

numerous today. In doing so, the objective is to expose why and how many interoperability and standardisation efforts, by claiming to be successful and enjoying political/higher management backing, cause more sorrow than joy. In doing so, *the objective is to guide the development of systems that are inherently capable of in-depth interoperability*.

However, as an insight assisting in the discussion, it is important to recognise that standards, claiming to be interoperability standards, often serve another purpose first, foremost and often solely. Many standards, especially industrial ones, serve to correct/modify the relationship between providers and customers. When customers will experience lock-in (cf. autocatalysis) into a provider's solution, the establishment of an official standard for the interface of this solution will make the vendor-customer relationship a lot healthier (i.e. typically a precondition for a customer to adopt the solution). However, this application of standardisation, too often under the flag of interoperability, has contributed to a situation in which there are as many standards as there are products. This does little toward achieving interoperability. A broad understanding of this situation may blunt attempts to push these standards as interoperability solutions when they are not. In fact, an in-depth understanding of interoperability, as conveyed in this document, is of the essence when designing and developing standards aimed at delivering high-performance interoperable systems (of systems).

4 When to interoperate (and when not)

This document is about in-depth interoperability: whether (existing) systems are able to interoperate or not, how to design (new, future) interoperability-capable systems, etc. In other words, it is about how to make interoperability possible and have it delivering as much value as possible. However, the fact that it is possible does not imply that it has to be done and, especially, made available to everyone. There are (at least) two aspects to consider before opening up a system's capability to interoperate to other parties.

First, it is a business decision to make a system functionality available, or even known, to a given party. Among others, there is the consideration whether a given interfacing functionality can/will be supported in the future, when a product is further developed into a more advanced version.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

Also, disclosing information about the internals of a product is a matter of (company) policy, affecting profitability and competitiveness.

In general, interoperability is important only when collaboration of different organizations matters, when the complete solution requires such mix and match of components designed independently. This often involves a trade-off between potential gain (e.g. from increased competition which lowers prices) and cost (redesign of existing systems, time spent in aligning interfaces). In telecom, the "smart network" (standardized in the early 90s) or the "IMS" standardized in 2003-2010 are examples of extremely bad cost/benefit ratio: the integrated systems continue to be proprietary in most implementations, the standardization effort was enormous, costing all parties an enormous amount of R&D, and brought very little tangible benefit.

Second, there are technical considerations. For instance, a low level interface may only be used by knowledgeable developers when improper use may have negative repercussions (e.g. damage or wear out some equipment). Therefore, some interoperability services may only be made available to qualified developers, which may have followed a training course and passed an exam.

This document is not concerned with the above decision-making unless it affects designing for interoperability (e.g. how to accommodate some technical considerations). What is important to note is that organisations will have good reasons to design for interoperability regardless what the outcome of the above decision-making may be. Large organisations simply need it internally. Small organisations need it to adapt to (changes in) their environment. Networked organisations need it to sustain and improve their position in their network. Internal use of the interoperability capabilities suffices to go for it.

Decisions about designing for interoperability will be affected by the extra effort and time (relative to quick-and-dirty), which will become more explicit below. Furthermore, an organisation's management may desire to be inherently unable to interoperate (as a kind of *poison pill*); note that, in this situation, its business position probably is uncomfortable as it cannot refuse to interoperate by simply telling the other (overly powerful) party that it won't interoperate.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

5 Systems as collections of resources

An upper bound for interoperability is delineated by the collection of (embedded) resources within the system that is targeted by interoperability demands. Explicit awareness of the potential represented by this collection of resources is a start. Explicit resource management and allocation will preserve full interoperability capabilities; there may still be a lot of effort required to answer certain demands but it will be possible to use the resources as needed or desired.

5.1 Awareness of the embedded resources and their accessibility

A first step to ensure (i.e. design for) or assess interoperability is to make an inventory of the embedded resources. Non-exhaustively, the following classes can be distinguished (in the kind of applications envisaged by STORY):

- Sensors (e.g. temperature)
- Actuators (e.g. on-off switches)
- Hard Real-time communication links (e.g. isochronous channels)
- Hard real-time information processing capacities (highest priority processes on an embedded computer)
- Soft Real-time communication links (e.g. Internet connections)
- Soft real-time information processing capacities (e.g. home computers)
- "Ironware" (e.g. pumps) and ''concrete-ware'' (e.g. water reservoirs), ...
- ...

Accessibility of these resources is the second concern. Non-exhaustively, the following characteristics of a system are (to be) made explicit:

- Immediate / straightforward access to basic resources (e.g. direct readout of sensor values)
- Higher level services offered, of which some may be
- Gatekeeper services managing (exclusive) access to embedded resources
    - Black box or grey box?
    - Taking their wishes for granted?
      E.g. returning *set points* when requested to return *measurements*
- Time in the services offered

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

- Watchdog services (e.g. every hour)
- Hard real-time services (e.g. within 20 seconds)
- Time stamping, synchronisation (NTP support, PTP support)
- ...

Here, the key concern is that it is exceedingly hard and overly complicated to build higher-level services without making assumptions. These assumptions will be conflicting with some interoperability desirability or requirements.

For instance, some system's control may be designed based on the assumption that thermodynamic efficiency is the objective (seen from this system's stand-alone perspective). When attempting to aggregate such systems and, subsequently, monetise the aggregate on power balancing/reserve markets, this assumption may be invalid and prevent access to a most lucrative market within the grid. Gatekeeper services, making unfortunate assumptions, are major culprits in interoperability let-downs and fiascos.

Noteworthy here, during the last 10 years, the main paradigm of computer interface design changed by 180°: *The 'location of the logic' has shifted*. For many years, object oriented design was the main paradigm. The underlying assumption was that to preserve the reliability of inner logic, a system would present as little internal resources as possible, and expose in its external interfaces only a restricted set which made sure that whatever manipulation of internal resources was fully under control. However, outside of certain areas where it succeeded (e.g. graphical interface design), in general object oriented software, object oriented databases, etc., did not transform the IT industry into an easy Lego game of object building blocks as expected. The reasons for that failure:

- An object designed by engineer A rarely presents the interfaces that engineer B would like to have... different business objectives or priorities.
- An object has "state", and state is difficult to synchronize in a system, making initial design and its scaling difficult.

Since the years 2000, the main paradigm for IT interface design has become "REST", and it is almost the contrary of object oriented design in its core philosophy. REST voluntarily keeps the interfaces very simple by using a CRUD model (Create/ Read/ Update / Delete). In the interface design

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

exposing a resource's state is to be avoided as much as possible. And last but not least, try to maximize transparency in exposure of underlying data. Of course, it is perfectly possible to expose an "object oriented design" by tweaking it into a REST model, and vice versa… but the resulting designs will be odd for developers used to the mainstream REST model: while an object oriented design considers that most of the intelligence is in the object, the REST design believes most of the intelligence is likely to be outside the resource exposed through REST.

REST has had enormous success. All internet portals expose their capabilities via REST interfaces, Simple "NO SQL" databases with REST interfaces have rapidly captured a large market share, and virtually all new IT interfaces now adopt the REST principles. The reasons for success:

- By not making assumptions on the use case and keeping most logic outside of the interfaces, a REST design is more likely to "fit all needs"
- Stateless interfaces connect easily, and scale easily
- CRUD design, with just 4 verbs, makes the telecom binding of functional interfaces to real world messages easy to specify and very flexible. The same interface can map to HTTP, CoAP, MQTT or others, and the specification time spent from functional specification to full all-layer specification is drastically reduced.

The shift from object oriented interface design has shifted the focus of interoperability specifications from "what is the list of your interfaces (switch off, move up, clear, set to x % …)", to "what is the format of your REST resources". As REST only exchanges representations of resources, i.e. documents, interoperability amounts to "do you understand my document".

5.2 Explicit and mandatory resource management

Explicit resource management provides/delivers an upper bound on the damage that can be caused concerning interoperability. When a gatekeeper service reveals to be a hindrance, deallocating its resources suffices to be able to replace it by a suitable alternative. Interoperability-friendly services minimise their resource allocation requirements, possibly supporting alternatives.

Explicit resource allocation, when properly designed, allows for enhanced modularity. Consider the following example for a control system of an installation:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

- The control system implements mandatory allocation for rights to possess/access the embedded resources within the control hardware and the installation.
- The control system implements ''reflection'' for its embedded resources such that external systems may discover them online at run-time.
- At start-up, a set of privileged safety-ensuring services have ''first picking rights''. No other service may receive ownership/access rights until these services have cleared the system (i.e. given the green light).
- These safety-ensuring services minimise their requirements regarding resource ownership
    - Isochronous channels to sensors (i.e. hard real-time access)
    - Hard real-time processing time slots
    - Watch dog services
    - Pre-emption rights to actuators

Note that by solely acquiring pre-emption rights for the actuators, the control system minimises its footprint and become interoperability-friendly with a wide range of ''application'' services. By minimising its demands, the designers of the safety-ensuring subsystem no longer have to guess what kind of cooperation will be required by the ''application'' services. They are able to implement safety services that remain invisible unless there is a compelling ground to intervene, except for the computing and communication capacities needed to monitor and to decide whether intervention is warranted. In a way they implement/approximate the minimal safety requirements inherently necessary for the system to remain ''healthy'' and ''operational''.

5.3 Pro-active resource management

STORY installations aren't simply computer infrastructures. They have different characteristics (from systems in a computer network), of which the following are relevant/decisive:

- The states of STORY installations are relevant when managing resource allocation. Unlike resource allocation in computers, allocation services cannot assume/require these resources to be in a reference state when allocated or deallocated (e.g. a CPU is designed to store and retrieve states when switching ''owner'' in a way that owners don't even notice what happened between a deallocation and subsequent CPU reallocation). Reflection

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

services (have to) allow discovering what states a resource might be in and resource allocation management will (need to) be state-aware. E.g. de-allocating a battery and allocating it to another user/service must account for its charge state.

- Story installations are high-value relative to the (modest) computer hardware that is used to manage them intelligently. Significant information processing and communication efforts can be afforded by resource allocation management services. In contrast, very simple myopic strategies must be used in computer operating systems, which are managing the resources embedded in a computer network.
- Proper handling of STORY installations involves actuation at one point in time with effects covering a future period of time. In an interoperability context, multiple activities will be interfering with shared (embedded) resources. And, this interfering will not (always) be immediate.

  Therefore, a more advanced resource allocation management will be time-aware. Its resources will have a calendar/agenda[2] in which activities/services may indicate future needs (reservations).

  An even more advanced resource management generates, from the agenda/calendar content, a forecast of the resource states, e.g. allowing to check whether a deallocation will hand over the resource in a state suited/compatible with the subsequent allocation/reservation.
- Resource allocations must not necessarily be all-or-nothing. Services and activities may have and exercise (compatible) rights simultaneously. E.g. a service level ensuring activity may constrain the state of a hot water reservoir, whilst a profit-optimising service uses and respects the available margins allowed by this activity.

These more advanced resource management services will be developed in subtask 3.4.2.

6 Activities using embedded resources

To unlock the potential of the collection of embedded resources, users – with interoperability demands – need to execute the proper activities on those resources. For STORY-relevant systems,

[2] As Google Calendar on an Android smart phone, i-Calendar on an i-phone, or Calendar in Microsoft Outlook.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

this ranges from simple user interfaces, e.g. allowing to switch a device on or off through a suitable communication link and protocol, toward full-fledged programmability.

In case of a simple user interface, handling interoperability will be straightforward. When properly designed, the interface offers access to the capabilities of the embedded resources in full but in a low-level manner, leaving the application development to be done. Note that interoperability calls for resource reflection, even for these simple systems. A server, on the Internet, with machine-readable specifications of the device/installation concerned is a start (e.g. a boiler has a URL at which its technical characteristics can be obtained, both by humans and machines/software). When interoperability-unfriendly, this simple interface limits access to the embedded resources severely, offering services based on rather specific assumptions (i.e. resulting in poor performance when prevailing conditions are differing from those assumptions).

In both cases, little can or needs to be added. Either much work/development remains to be done (this is the desirable case for interoperability) or mismatched services refuse to deallocate the embedded resources (this is problematic; avoiding/replacing such systems is indicated). Interoperability matter in this area of simple interfaces can be addressed by industry, possibly facilitated by coordinating or supporting actions; (research and) innovation actions have no contributions that are worth of their expense.

For such fieldbus/industrial/automation type of interoperability, much work already has been performed by standardization bodies which already produced usable systems. Both OASIS and ETSI adopted the REST design and focuses on document formats that would allow interfacing with clear objectives in mind:

- Allow mapping onto many, ideally all, known existing fieldbus protocols: some academic interoperability efforts failed because of lack of knowledge of the complexity of the real world. For many such academic “top down” approaches, a lamp is just a “on off thing”... taking a close look at KNX or BacNet immediately changes this perception. Both OASIS and ETSI made a huge “bottom up” effort making sure their work would be usable in the real world.
- Interface means abstraction: it should be possible to expose fieldbus protocols in a way that is as homogeneous as possible, i.e. by manipulating the same set of REST resources.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

- However, abstraction should not mean limitation. Real world applications will appreciate abstraction and “protocol independent” interfaces in some cases, and in other cases will require access to low level specificities.

The REST based design for such fieldbus and automation protocols implements the following pattern:

- The resource hierarchy matches virtually all known automation protocols
- A “driver” resource exposes fieldbus resources
- A fieldbus resource exposes node resource on the network
- A node resource exposes virtual nodes (e.g. 4 buttons in a switch)
- Each virtual node exposes interface resources (e.g.; “on/off” interfaces, “gradation interfaces”)
- Interface resources expose properties, events and actions
- Each resource document exposes an XML namespace corresponding to abstract ontology (i.e. protocol independent, like OASIS oBix), while additional namespace for each protocol exposes a much native information as possible. Both name spaces are present in documents, the user parses what he needs.
- More details : see video http://cocoon.actility.com/documentation/ongv2/ETSI_M2M_videos

The interoperability on more sophisticated service levels and programmability are the focus of this task in an innovation action. In particular, the implications of universally applicable principles (laws of the artificial) are revealed and accounted for. Too often, these laws are ignored/unknown and the result is analogous to designing and building an irrigation system while ignoring Newton’s laws of gravity: it is very expensive[3] to build, maintain and operate and it performs poorly relative to what should be possible.

This section aims to bring the knowledge and systematic approach to assess how well system designs account for relevant universally valid insights. Also, a strategic objective is to create

[3] A lot of pumps are needed, energy for those pumps is needed, where gravity would do the job; pressurised pipes are needed where open canals or non/low-pressurised pipes would suffice. In ICT, this translates into high (e.g. duplicate) software and system development efforts and maintenance costs in combination with poor services.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 646426

awareness of how expensive and unsound in the long run the prevailing ''balkanised'' situation in industrial/home automation really is.

6.1 The valley of death

This section observes that manmade systems and technology (artefacts), concerning activities on embedded resources, are either very simple or highly sophisticated. In-between designs are unsustainable or even fail to the start. Exceptions enjoy a ''protected environment'' (e.g. markets in which customers are experiencing an unhealthy vendor lock-in, resulting in monopolistic situations); these exceptions do not survive when their protecting walls are breached; typically, they offer underwhelming value-for-money.

6.1.1 Autocatalysis for manmade systems – critical user mass

In this section, *critical user mass ensuring autocatalysis* is the most relevant (implication of a) scientific law of the artificial. Here, the elements/factors to be taken into account are:

- The complexity and size of the artefact (e.g. of a home automation programming tool and/or language). This determines how much intellectual effort and information processing is needed to create and sustain the artefact.
- The dynamic nature of the world in which the artefact resides. This limits how long it may take to develop and/or adapt/maintain the artefact.
- The competitiveness of the environment in which the artefact resides. This also limits how long it may take to develop and/or adapt artefacts. It equally calls for top quality from the services.

Note that an absence of competition – e.g. when an organisation has a monopoly – is undesirable in two ways (at least). First, poor service normally is a consequence thereof. Second, this organisation will be ill-prepared when its domain is opened up to competitors that already have experience in such competitive environments. Concerning smart grid technologies, smart has to compete with conventional. When ignoring the insights discussed here, it becomes more likely that a business case analysis will advise against the smart alternative (i.e. conclude that it is too expensive to go it alone with own/proprietary designs).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

Manmade systems, especially software-intensive ones, need to belong to two autocatalytic sets. The members of these sets are the manmade artefact(s) themselves and their users. The users in the first set provide the economic means/resources that are required for the artefact to emerge and remain successfully in a dynamic and competitive environment. The users in the second set (many users will be a member of both sets) provide the information needed to improve the artefact and keep it competitive.

An example of an artefact enjoying very strong autocatalysis economically but marginal autocatalysis information-wise is DECT[4] technology. Its huge number of users represent too little variability on how they use this technology (many user instances of a very limited number of user types). Among others, this has prevented adoption and adaptation of this technology in new application domains (e.g. manufacturing automation, M2M communication in smart homes). It is likely to become/be stagnant as a result.

Remark: Today, in the smart grid domain, there is a tension between the hardware-connected artefacts and IoT/Internet artefacts. The latter – younger and more advanced technologies – support hyperconnectivity, allowing for grid-wide solutions. They enjoy critical mass on a global scale with players like Google, Apple and Samsung. Here, organisations and professionals enjoy critical mass when developing and marketing smart grid applications. It's the future. On the other hand, the developer of a smart switch, pump, etc. sees critical mass when employing older industrial standards (KNX, ModBus), which do not provide this hyper connectivity. It is too early for these hardware and firmware developers to adopt IoT solutions and they often lack the resources to support both. Today, the world experiences a transitional phase in which 'bridges' will be needed in installations, allowing the respective contributors to enjoy their – mutually exclusive – critical mass. Technologies such as LoRa, LWM2M, etc. will gradually change this into a more homogeneous and cost-effective solution but the installed base implies a relatively slow migration into such *ideal* designs.

[4] In-home wireless telephone handset.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

6.1.2 The simple side of the valley of death

When systems are simple, the required number of users is small and easily realised. Typical for these easy-to-sustain solutions is:

- Low requirements for user skills. Anyone with a brain can use these systems. An ICT degree is not necessary at all.
- No debugging challenges. When something needs correcting, it is trivial to spot what is wrong and how to correct it. No trouble-shooting experience is required.
- Self-explanatory. It is not needed to follow product-specific training.

The main drawback consists of the limited possibilities of such simple designs. They lack expressivity, which often is required to interoperate or to offer the required service levels that are needed to unlock lucrative (mass) markets.

6.1.3 The sophisticated side of the valley of death

Here, we find full-fledged programming technologies, making little compromise on expressiveness, and state-of-the-art telecommunication technologies. The absence of concessions on applicability and/or expressiveness has allowed a number of them (i.e. normally the first ones to succeed) to recruit a high number of users (i.e. they have become mainstream). They enjoy very strong autocatalytic processes, which facilitate their enduring success.

Here, user mass is more decisive than technology, on condition that the technology is adequate. As the world changes, mainstream technology becomes legacy (kept alive because of lock-in) and newer technologies takes over. Unfortunately, such changes are not easily understood nor predicted. This will be discussed further, looking at programming languages and technologies, in section 6.2. Note that a similar discussion can be held on telecommunication technologies and standards. A similar discussion can be held for embedded computer hardware (e.g. IoT nodes).

6.1.4 Inside the valley of death

Attempts to design and implement solutions that have some expressiveness without becoming full-fledged programming tools are doomed. Rapidly, debugging becomes a major concern, which

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

requires skilled professionals. These professionals, when they take sensible decisions about their career, will not be prepared to invest a lot of time in a technology that is not mainstream.

Autocatalysis exhausts the pools of resources from which it extracts the means to ensure a strong presence of the set members in the future. It leaves no room for competitors that arrive late, have weak self-reinforcement … In the STORY domain, professionals – both individuals and organisations – will be drawn to mainstream technologies. As a consequence, the path between mainstream (enabling to recruit professionals) and simple (not needing those professionals) leads through a valley in which technologies, normally, cannot survive.

6.2 Mainstream technologies and how they evolve, appear and disappear

The above draws a picture in which mainstream appears to dominate forever. We know that this is not true. The world changes and mainstream changes, reluctantly. For innovation actions, it is relevant to understand how these changes – vital for society in the long run – become reality. It is essential not to waste irreplaceable resources on keeping stagnant technologies alive while hindering upcoming ones that have substantial value to offer. This text looks into programming languages. A similar analysis can be done for communication (Wi-Fi, Bluetooth, NFC, USB, LoRa …) or embedded computing platforms (Raspberry Pi, Beagle Bone Black …) where the latter currently is characterised by a fast-changing landscape.

Concerning interoperability, the point to make is that choices are limited to mainstream and, but only for good reasons, a selection of upcoming technologies. Here, the balkanised culture of industrial/home automation goes directly against the laws of the artificial, which are universally valid as soon as developments are non-trivial. As will be illustrated below, industrial/home automation may have made itself irrelevant concerning programming languages (i.e. invisible when looking at mainstream and candidate future mainstream) when not bothering about critical mass (and preferring to create customer lock-in) while developing own isolated solutions instead of seeking connections with mainstream technology.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

6.2.1 Language popularity

The Tiobe[5] index provides elementary insight in the popularity of programming languages. For the purposes of the present discussion, the eventual bias, imprecision and imperfection have little impact on the conclusions. The index distinguishes the top 20 and the other top 50 languages:

Position	Programming Language	Ratings
1	Java	17.822%
2	C	16.788%
3	C++	7.756%
4	C#	5.056%
5	Objective-C	4.339%
6	Python	3.999%
7	Visual Basic .NET	3.168%
8	PHP	2.868%
9	JavaScript	2.295%
10	Delphi/Object Pascal	1.869%
11	Visual Basic	1.839%
12	Perl	1.759%
13	R	1.524%
14	Swift	1.440%
15	MATLAB	1.436%
16	Ruby	1.359%
17	PL/SQL	1.229%
18	COBOL	0.948%
19	ABAP	0.849%
20	Pascal	0.846%

Position	Programming Language	Ratings
21	OpenEdge ABL	0.796%
22	SAS	0.787%
23	Assembly language	0.754%
24	Dart	0.671%
25	Scratch	0.628%
26	D	0.610%
27	Fortran	0.582%

[5] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Position	Programming Language	Ratings
28	Lua	0.546%
29	Logo	0.536%
30	Scala	0.531%
31	Prolog	0.518%
32	Lisp	0.506%
33	Transact-SQL	0.486%
34	Ada	0.433%
35	(Visual) FoxPro	0.411%
36	LabVIEW	0.382%
37	Inform	0.376%
38	Groovy	0.370%
39	F#	0.342%
40	RPG (OS/400)	0.312%
41	ActionScript	0.303%
42	Scheme	0.273%
43	Erlang	0.267%
44	Awk	0.262%
45	ML	0.241%
46	VHDL	0.228%
47	Ladder Logic	0.214%
48	Haskell	0.214%
49	Z shell	0.208%
50	VBScript	0.202%

This information is relevant in the context of interoperability in a number of ways. First, the top languages cannot be ignored whenever relevant. Fortunately, all have C interoperability features. In other words, offering similar/compatible/interoperable C-interfacing services will enable cooperation with almost every relevant language in this list.

Second and more relevant, when including programming facilities within the system that is developed, inventing another programming language is not an option. Adopting a suitable top language and using its language extension[6] mechanisms (e.g. adding a software library and/or a

[6] An example is the JavaScript ecosystem in which JQuery and AngularJS ensure autocatalysis by participating in an existing ecosystem; they improve autocatalysis of an existing set.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

framework) is the preferred manner of working. Except for very simple scripting, joining the ecosystem of an existing popular language is the only option when interoperability is a concern. Indeed, being able to recruit talented developers and benefiting from sizeable developers community is essential.

Selecting a language outside the top 50 is questionable. Selecting a language outside the top 5 needs a motivation. Moreover, this list needs to be reduced first, eliminating unsuitable candidates. Many of them are (too) old (e.g. COBOL, FORTRAN, and Pascal). Others are special-purpose where this purpose is of little importance in a STORY context (e.g. VHDL). Some are low-level (e.g. Assembly language, Ladder Logic).

The actual choice among the top 5 depends on the development/deployment environment (e.g. .NET). The selection of a top 50 language, other than the top 5, must be motivated by specific requirements or decisive advantages. Examples are JavaScript (in-browser code execution), LUA (embed-able scripting language offering a lot of control by the server over script execution), and Erlang (distributed computing, robustness and stability, fast and easy transition from TRL4/5 to TRL7).

Finally, the guideline against inventing and introducing another programming language applies to all aspects of the system under development. For instance, using a top 5 programming language for programming the equipment in the installation while inventing an own non-trivial scripting language for its users[7] (e.g. to represent their preferences on energy consumption) is a violation of this guideline. Such scripting languages shall be very simple and embedded in a software library module such that removal/replacement/resource deallocation is supported. When this own language poses interoperability issues (e.g. lack of expressiveness or nobody is prepared to invest time in the scripting language), the module can be replaced straightforwardly.

6.2.2 Adopting not-yet-mainstream technology

In the above, the guideline is to select mainstream, on condition that it delivers what is needed. Today's mainstream technologies have mechanism to extend their functionality, which is again the

[7] A common design mistake concerning interoperability is to treat part of the world-of-interest as second class citizens that can be addressed in data formats or simple scripts, only to discover later that the required expressiveness drives such system designs into this valley of death.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

first choice. If this is insufficient, selecting a more advanced technology is indicated. This technology needs to answer recently emerged challenges (e.g. AngularJS and security on the Internet) or old challenges that recently became much more important (e.g. Erlang/OTP and multi-core, distributed, robust computing in combination with a swift transition from TRL4/5 to TRL7).

Nonetheless, minimal indications of autocatalysis are required. Checks that can be and should be performed:

- Is acquiring the necessary skills "a good career move[8]" and/or "fun[9]" for the level of professional involved? If a Master of Science level is required, Ladder Logic will not qualify. If a professional Bachelor level is indicated, a healthy community answering questions on the Internet is a necessity.
- How accessible is the technology? Is there a (basic) version downloadable without any fuzz? If a lector in computer science needs to negotiate to get a class room license, has to do some promotion in return whilst his/her contact, in the company owning the technology, repeatedly faces internal negotiations and/or administration to donate such license … the (programming) technology shall only be adopted as a very last resort.
- How well does it cooperate with above-listed the top 5, especially with the C programming language? This C connectivity is your safety net when the novel technology fails to maintain autocatalysis and needs to be phased out (gradually).

Note that these questions relate to the ecosystem of the language and its behaviour impacting on the growth and flourishing of this ecosystem. The technical merit, which is warranting some risk taking, is case specific.

Overall, managers are aware of this issue (critical user mass) but are overly conservative for novel technologies offering decisive benefits (in a longer run) while they are too lenient against legacy and ''monopolistic'' technologies (in a balkanised industrial/home automation world). Here, the Tiobe index offers a reference to distinguish the embryonic from the promising and upcoming. Some languages have names to counter this exaggerated distrust from management (e.g. JavaScript, C# …).

[8] I.e. it looks good on a CV.
[9] E.g. Lua is used in the WoW on-line massive multi-player computer game, which has many skilled IT-ers playing.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Finally, in all cases, both simple and sophisticated, the presence of arbitrariness affects interoperability. A simple interface, communication technology ... imposing arbitrary choices (often pretending to be smart by doing this) will be interoperability-hostile. Sophisticated systems are unlikely to avoid arbitrariness, but a suitable architecture (see further) and joining/linking to strong ecosystems (i.e. mainstream or with a specific technical capability and minimal critical user mass) will enhance their in-depth interoperability.

6.3 Non-monolithic systems

When developing non-trivial systems in a dynamic (finite time windows) and demanding (competitive) environment, flexible (aggregation) hierarchies outperform developments from scratch. This is an insight brought to us by Herbert A. Simon, Nobel Prize winner based on his work on limited rationality and its implications.

Again, a scientific insight advocates against ignoring existing solutions as soon as the requirements mandate a sophisticated (= expressive) solution. Large, complex solutions combine, adopt and adapt existing systems/solutions/technologies. These subsystems need autocatalysis to enjoy an adequate level of stability and longevity when the overall system has to be competitive, sustainable and maintainable.

This insight is largely respected (in industry) already and does not require further elaboration at this point, except the need to select an autocatalytic subsystem, which was addressed already above.

7 Interoperability of existing systems and technologies

In view of the above, the in-depth interoperability of existing systems/technologies is a property, not a task to be done. Hence, the first step is to assess the interoperability capabilities of an existing artefact. This will be a damage assessment identifying how much of the intrinsic potential, represented by the embedded resources, is preserved or respectively lost. Moreover, the services, allowing to perform (user) activities on these resources, will be assessed on their expressiveness and their ability to belong to strong ecosystems. Subsequently, damage containment elaborates how the remaining interoperability-friendly services, features and functionalities can be used. In

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

particular, damage control aims to avoid that interoperability-hostile services and features are allowed to build up (even more) inertia[10]. Note that such an assessment is (at least) equally relevant for existing (interoperability) standards.

7.1 Damage assessment of an existing system/technology

Step 1. Identify the inherent potential. The collection of embedded resources is to be documented. Section 5.1 lists the resource classes that need considering in this step.

Step 2. Identify the (lack of) accessibility of the embedded resources. This includes timing aspects, including bandwidth, delay, jitter, hard and soft guarantees as well as time stamping availability and accuracy. This includes the eventual support for resource allocation and deallocation (pre-emptively or otherwise).

Step 3. Identify the services (GUI, communication links and programming facilities) that are available to have the embedded resources execute and/or undergo activities.

Step 4. Identify the reflection functionality. How much of the inherent potential is adequately modelled and/or specified to be able to plan activities, assess them before executing, have model-driven applications, etc.

Step 5. Assess the involvement of the non-trivial services in strong ecosystems (= membership of strong autocatalytic sets comprising a large and diverse user mass). Check whether these ecosystems are strongly recruiting (e.g. freely available tools, courses ... on the Internet) or exploiting a locked-in user community.

Step 6. Wrap up. Identify the arbitrariness in the existing system, especially when present in gate keepers that have exclusive access to certain embedded resource. Identify activity-related services (e.g. programming facilities) that lack critical user mass relative to their complexity. Identify activity-related services that have decisive functionality or features and their potential to obtain and sustain a critical user mass.

[10] I.e. avoid and prevent developments that rely on harmful elements in the existing systems, making it easier to deprecate and eventually remove such elements.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

7.2 Damage containment

Based on the damage assessment, the impact of gatekeepers is to be minimised and development using non-mainstream services to define/program and execute activities is to be avoided. Only basic services from gatekeepers are to be used (unless there is a compelling reason to do otherwise). Non-mainstream services are used to bring basic functionality to communication links or a C-interface, from which mainstream (or almost mainstream offering advanced and needed functionality) technology is used to implement the user applications.

A possible approach is to use the available device/installation specific hardware and services to transform the device/installation into a peripheral device connected to a suitable mainstream platform. The on-board hardware and software is used to implement a device driver, which makes the embedded resource capabilities available to the user application on mainstream platforms[11]. If a mainstream platform is already embedded in the device/installation, it can be reused and C-interfacing may be an alternative to communication links.

7.3 Sorts of programmable equipment

As background information, in (industrial) automation, the following categories can be distinguished:

- CNC or Computerized Numerical Control is commonly found on machine tools like milling and drilling machine but also on, more recently, 3D printing equipment. From an interoperability perspective, CNC is a poor choice because it is unable to interact during the execution of an activity. All input has to be provided up front, analogous to sending a document file to a printer. Any subsequent interaction, after sending the file to the equipment, is predefined and general purpose (analogous to prompting the user to add paper when running out of it). Expressiveness of CNC programming languages is limited, especially concerning interactive-ness (e.g. unable to adapt machine speeds, feeds, trajectories in function of sensor data). Often it is not possible to download a program while executing the current one and having this next program executing seamlessly as soon as the

[11] Among others, this device driver approach avoids the need for the services of a professional knowledgeable in the non-mainstream technology when adapting or maintaining the application developed using mainstream technologies.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

running one terminates. It is worth to notice that this would be perfect to implement the above damage containment approach.

- DNC or Direct Numerical Control is a CNC control in which blocks of code are sent to the machine controller and executed immediately. However, DNC implementations were not designed to achieve interoperability and interactivity beyond CNC. Expressiveness of both CNC and DNC programs is limited. In theory, DNC can be generated by software written in a mainstream language on a mainstream computer platform, rendering the machine tool into a true peripheral device (allowing to implement damage containment as discussed above).
- Robot software is characterized by a fixed and full resource ownership of a program execution system limited to a proprietary robot programming language. Autocatalysis is non-existent. The programming facilities can be used to implement the damage containment discussed above, except for demanding real-time functionalities. Note that multithreaded programming is not commonly supported within robot software systems.
- Predefined building blocks. The technology consists of components with a fixed set of services and functions, which can be configured (possibly aided by a configuration tool). Basic KNX functionality follows this model (e.g. GUI-based connecting a button to a switch with a five second delay).
- Programmable Logic Control (PLC). Commonly used programming solutions (IEC61131-3) only offer low-level programming facilities (we are excluding proprietary solutions and programming tools), especially when looking at networked and distributed applications. More recent designs still lack critical user mass (IEC61499).
- Mechatronic systems use mainstream technology, typically a real-time software platform programmed in C/C++, in a closed organisation setting. Anything/everything is/remains possible inside the developers' organisation, but access from outside has to be provided explicitly (as a service or functionality). SCADA systems are very similar. The developers have full access and rights, but outsider facilities/rights have to be provided/implemented explicitly.

The purpose of this section is not to pose the above categorisation as final or a universal truth. It only illustrates that device control designs barely considered interoperability as an important

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

concern. Many of the interoperability limitations have historical reasons. E.g. CNC and DNC were developed in the early days of the cold war when even the capabilities of a first-generation personal computer would cost over 1 million Euro/Dollar and require hardware that filled a very large room. As a result, the expensive computer would generate a paper tape with holes punched into it, which a mechanical devices would convert into movements of the machine tool (think of a music rolls in a self-playing piano). Today, this arrangement and its limitations still persist in CNC systems. Likewise, early robot software designers faced computer memory limitations, which induced design choices and limitations that persist until today.

Mechatronic designs do not induce such limitations/legacy but, until today, have no widespread answer delivering truly open and interoperable solutions. This should not be a surprise, given our poor understanding – as a developers/designers community – of what interoperability entails, given that we are only at the beginning of building large interoperable (eco-) systems, and given the conflicting interests and power balance implications of achieving interoperability. Overall, this section illustrates that much remains to be discovered, understood, disseminated and implemented, while facing existing designs that cannot evolve into the required solutions.

8 Design for in-depth interoperability

The key to in-depth operability by design is the (software) system architecture. This system architecture will be an instantiation of a reference architecture that brings the appropriate resource awareness and manner to coordinate activities interacting through the resources. The architecture maximises the (possible) ratio between user masses and system complexity/size. Architectural patterns, complementing the reference architecture, enable swift realisations of reliable and advanced solutions. This chapter introduces these in-depth interoperability facilitating elements.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

8.1 Reference architecture part 1 – Resources

8.1.1 Identification of the (embedded) resources

A first step, which is likely to be repeated/iterated when more information becomes known and/or fixed, consists of identifying the relevant resources. The identification distinguishes resource types from resource instances.

Our_Simple_Boiler example (part 1)

Embedded resources:

- Reservoir (container)
- Reservoir (content)
- Temperature sensor
- Heating actuator

8.1.2 Reflection of resource (class/type) capabilities

For the resources, that have been identified, a human and machine readable source of information is provided reflecting the resource capabilities. This includes the technical specifications but also the (kind of) connections that can exist to the rest of the world. This information source can, but does not need to, be collocated with the resource. It can be available through the internet, in the cloud, on home computers and even mobile devices. Availability (e.g. 24/7, 99.999% of the time) is an issue to be decided/resolved by the developers. In cases, it may be available at multiple places to improve and guarantee availability.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

Our_Simple_Boiler example (part 2)

Embedded resource (type) capabilities:

- Reservoir (container)
  - Dimensions
  - Connector location and dimensions (in and out)
  - Sensor location
  - Heating element location
  - Insulation characteristics (thermal)
  - Pressure range
- Reservoir (content)
  - Range of allowed liquids (tap water ...)
- Temperature sensor
  - Sensor characteristics (readout values versus degrees Celsius, delays, bandwidth, accuracy, calibration ...)
  - Power supply characteristics
  - Communication links
- Heating element/actuator
  - Heating element characteristics
  - Actuation limitations and implications (e.g. of frequent switching)
  - Power connection characteristics
  - Communication links

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

In a commercial implementation, such exposure interfaces should consider the existing work of OASIS oBix and ETSI (e.g. TR 102966), particularly when it comes to the identification of common resource hierarchies to expose objects that are controllable by means of a fieldbus/automation interface.

Advanced reflection services allow to answer ‘’what if’’ queries. Our_Simple_Boiler would be able to estimate its internal state (water temperature of its content) when given an initial state and an ‘’agenda’’ containing the actions that are to be performed from this initial state onward. Importantly, this “what if” service only requires self-knowledge from/about the resource type. Any state-related information is kept and provided by the resource instance; any usage/scenario-related information processing is provided by the activity types and/or instances.

An even more advanced version is able to exchange information with its neighbours to estimate – even better – how its environment will impact the correct answer to such a query (e.g. temperature surrounding the boiler, amount of warm water that will be consumed).

These answers can be scalar, e.g. trajectories corresponding to best estimates, or parameter values for probability distributions indicating uncertainty on those estimates additionally. An example query would be asking for the time at which a given state (temperature of the water at the sensor) is reached when switching the heating element on in the present state (retrieved by using the resource reflection services as well). Conversely, the query may indicate the duration of the heating and request a state estimate (temperature).

At no point, these reflection services may impose decisions or choices. Exotic states can be covered by crude models and services (e.g. by a catch-all state simply stating that not much is known or can be guaranteed). However, the reflection must be complete (e.g. cover time from minus infinity to plus infinity which means, among others, that decommissioning is modelled/accounted for). For instance, when legionella is detected, the reflection services must not abandon their users when they ask for extreme temperatures, possibly shortening the expected life time of the equipment.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

This reflection is only concerned with the resource type/class; it is a technological "expert." It is about capabilities, not capacity and its availability. Moreover, it is a goal-agnostic expert; it does not make (or minimises the inertia of) assumptions on the usage of the resource.

8.1.3 Management of resource (instance) capacity/availability

Actual resources (i.e. resource instances) have a (software) companion to manage their allocation and usage. Type-related matter is delegated to the resource type reflection discussed above. This resource manager is aware of the resource state but leaves the creation and transformation, manipulation and interpretation of technical information to the resource type reflection above.

The resource manager is aware of the resource's environment, and is the information source concerning:

- The containing resource instance (e.g. the room in which Our_Simple_Boiler resides)
- The neighbouring resource instances (e.g. the water piping in and out, the electrical connections)
- The contained/embedded resources (e.g. temperature sensor, heating element, body)
- The visiting resources (e.g. the water)
- ...

Basically, a computer program can detect and discover the world around the resource instance by communicating with these resource management services.

The manager knows the resource capacity/availability/allocation in a time-aware manner, and maintains the resource agenda. Advanced resource reflection allows estimating/predicting the resource state trajectory corresponding to the agenda. The manager is able to answer queries about capacity availability (i.e. information requests without commitment) and handle reservation requests (i.e. update the resource agenda). The manager is responsible for actual execution of activities (see below). The ability to answer queries is related to the (expected/predicted/measured/...) state of the actual resource instance.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

Our_Simple_Boiler example (part 3)

Embedded resource instances state (incl. access/control), trace and agenda:

- Reservoir (container)
  - Link to its type
  - Link to the connected resource instances
  - Link to enclosing resource (incl. coordinates/location within it)
  - Actual pressure (in function of time when relevant)
- Reservoir (content)
  - Link to its type
  - Actual nature of the liquid, link to its source
  - Temperature (distribution), possibly in function of time
- Temperature sensor
  - Link to its type
  - Sensor readout, possibly including a timestamped trace
  - Link to power supply resource instance
  - Availability of communication links and bandwidth
- Heating element/actuator
  - Link to its type
  - Heating element status and possibly trace
  - Actuation agenda (incl. control rights) in advanced versions
  - Link to connected power supply resource
  - Availability of communication links and bandwidth

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

The resource managers are implementing explicit and mandatory resource allocation. The activities need to obtain suitable rights to execute and, when indicated, can have their rights revoked.

8.2 Reference architecture part 2 – Activities

Interoperability includes the ability to execute activities on the concerned resource instances. The services offered range from simple and primitive toward full-fledged programmability. As revealed above, the in-between zone (halfway programmability imposing restrictions but already suffering from debugging challenges) is inhospitable; its user mass potential is low relative to autocatalysis requirements. Connecting to fully programmable/expressive solutions and employing of their extension mechanisms (e.g. adding a software library) is indicated when aiming to deliver non-trivial services.

8.2.1 Simple devices and services

In this case, the following guidelines apply:

- Preserve the intrinsic capabilities of the (embedded) resources. The offered services must not prevent ''unforeseen'' usage of the resources.
- When applicable, implement authorisation mechanisms ensuring only knowledgeable persons and accountable organisations gain access to ''privileged services'' (e.g. admin or root access).
- Invariants can be hardcoded and enforced; these must be always true no matter what happens (e.g. create dangerous situations, damage the resource severely). Increased/abnormal wear (e.g. caused by high temperatures) or poor efficiency (e.g. caused by frequent switching) are both insufficient justification for denial of access (e.g. to an actuator); this does however warrant the implementation of an authorisation mechanism.
- Undesirable usage may trigger warnings and may require privileged access rights. Resource reflection shall communicate consequences but the service interface, possibly after receiving an acknowledgement that implies a liability reassignment, must not prevent or deny such usage.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

- Access to the services supports/uses communication through suitable mainstream channels. Mainstream ensures critical user mass (autocatalysis). Suitable access includes bandwidth and response time (e.g. isochronous channel when real-time service is needed to obey the first guideline). Suitable includes practical and convenient (e.g. LORA to ease installation and configuration management). Industrial/home automation communication standards suffer from balkanisation (cf. damage control above). If necessary, a communication hub/bridge may be provided to cope with communication links lacking autocatalysis, which provides a mainstream channel to the rest of the world.
- Activities must be designed on the basis that they are able to influence resources rather than command them (i.e. they must not have illusions on what is possible and/or will happen). The activity type is the element that bridges the gap between what is to be achieved and how to do it with the resource types available. The activity instance handles the selection of the actual course-of-action based on the options offered by the activity type (which may be non-deterministic) and the resource instances (i.e. their presence, allocation and state).

8.2.2 Programmable devices and systems – lacking critical user mass

When a development employs programmable devices that do not enjoy autocatalysis, their programming facilities are used to implement a device driver (i.e. the device-side part of it). The device driver services and functionality convert the device into a simple device (as discussed above). This minimises the inertia of the programming services lacking critical user mass, making it easy to replace the device. It also limits the need for services of programmers mastering such marginal language to the development of this device driver (device-side). Application development and maintenance is done using mainstream only.

8.2.3 Programmable devices and systems – enjoying critical user mass

The above concerning activities on resources – i.e. simple devices/services or devices/systems lacking critical mass – allows to implement the required non-trivial activities on mainstream platforms. It allows to use programming languages, tools and technology that enjoy critical user mass (relative to their complexity). Alternatively and additionally, it permits to employ advanced

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

technologies offering decisive benefits/services for the situation at hand, possibly not (yet) mainstream but enjoying minimal/adequate critical user mass nonetheless.

Relatively simple applications can be programmed directly. However, when the system aims to implement comprehensive interoperable activities, a suitable structure (i.e. instantiating a proper reference architecture) is indicated. This structure distinguishes user services and service levels from device management. This is detailed below.

Allowed user service range – the collection of possible state trajectories

The activity implementation is divided into two subsystems. The first subsystem reflects the user service requirements. This reflection of user requirements is, ideally, as low-demanding as possible, minimising decisions on resource utilisation. For instance, *Our_Simple_Boiler* will leave open how electricity is consumed to keep the water temperature in the required range over time. The first subsystem represents the range of possible manners to answer user requirements. It does not generate a specific trajectory answering these requirements. This has two advantages. It maximises cooperativeness (i.e. only asks for what is really needed) and it allows to combine with a resource manager answering needs originating from elsewhere (i.e. interoperate well).

Our_Simple_Boiler example (part 4)

- Function/service specifying the required temperature-volume in function of
  - Time (of the day/week)
  - Recent consumption

  that has to be available (i.e. a fixed parameterised schedule).

- Advanced function/service interpreting the electronic agendas of the users to
  - Generate a required temperature-volume in function of time

  that has to be available (i.e. a user-driven scheme).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

Note that this subsystem only depends on the resource classes/types and may remain agnostic about resource instances (i.e. about capabilities not capacities or availability; about the possible states, not the actual states).

Device/resource management – discover resource availability/capacity and generate trajectory

The second subsystem retrieves the possible trajectories from the first subsystem, discovers the resource availability and generates a valid trajectory. Here, the activity manager interacts with the power supply resource manager to discover desirable (low-cost) trajectories. An activity manager may use the reflection services of *Our_Simple_Boiler* to discover and compute trajectories fitting the allowed user service range. Overall, the activity manager interacts with resource managers to get a suitable trajectory executed and, in advanced versions, reflected in the affected resource agendas. Interoperability is facilitated through such interaction with the resource managers.

8.3 Architectural patterns – Proactive coordination

For advanced coordination, the delegate MAS architectural pattern [1] allows generating short term predictions of activity trajectories/routings and resource utilization/state trajectories (cf. resource agenda above). It suffices for the reflection to support ''single step/action what-if services'' for an implementation of this architectural pattern to generate multi-step/action predictions. A similar architectural pattern supports discovery of and exploration for possible courses-of-action for the activities.

These patterns enable a single-source-of-truth (SSOT) decentralised design to account for (future) interactions, which delivers in-depth interoperability. Other patterns can be designed and implemented to deliver ''global or non-local'' information and action while preserving SSOT.

Concrete examples and software support for these architectural patterns will be developed in subtask 3.4.2.

8.4 Remarks

When elaborating a concrete implementation of the reference architecture discussed above, a clean separation between decision making elements and reality reflection has to be maintained. In particular, any reduction of the solution space, needed to restrain the computational complexity in

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

the decision making, must never infiltrate the reality reflecting parts. This reduction must be implemented within the decision making subsystem.

Moreover, as little functionality as possible must be realised in the decision making subsystems and as much as possible must be addressed in the reality reflecting parts. These reality reflecting parts aim to model whatever is relevant, abstaining from reducing expressiveness for small gains. For instance, time will be represented in 64 bit and in milliseconds or even microseconds, because little expenses are saved from switching to a more coarse representation for reflection services. In decision making, this can be a completely other matter but conversion to a more coarse representation is easy (if not the reflection is missing some relevant property; e.g. it may use a scalar to represent an interval or a distribution function).

The reference architecture and architectural patterns discussed and/or mentioned in this section will be supported by the implementation elaborated in subtask 3.4.2.

9 Conclusions

This document provides insights and guidelines concerning in-depth interoperability for the development activities within the STORY project. This involves the design and elaboration of new systems as well as the assessment and handling of existing systems.

It identifies the *arbitrariness of design choices* as the root cause for interoperability show stoppers. From there, the document identifies key issues such as *gate keeper services* that make unfortunate assumptions, which reveal to be falsified by interoperability requirements. Likewise, the document presents answers to these issues such as mandatory and explicit resource allocation, allowing to replace gatekeepers when needed.

Reflection is a vital service to be provided by interoperability-friendly designs, allowing other systems to discover and properly use resources in a smart grid. Indeed, reflection is intrinsically choice-free except for the decisions concerning what (not) to represent. When implemented properly, adding missing parts will not invalidate users of the original version. Note that *shallow* interoperability efforts mainly suffice (i.e. syntactical translation of data) to achieve in-depth interoperability.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

The importance of critical user mass is discussed. It has implications for non-trivial developments and industrial automation is suffering in this respect. Again, solutions are put forward such as implementing a device driver to bring the application development and maintenance into mainstream environments.

Moreover, the document discusses the architecture and design patterns of systems designed for in-depth interoperability. They provide – as much as possible – relevant information without making these problematic arbitrary choices (e.g. reflection); they limit the inertia of such choice-making elements (e.g. explicit resource allocation and de-allocation). And, design guidelines call for delegating tasks/services in order to avoid relying on assumptions about the others. The recommended system has a single-source-of-truth design in which components only rely on self-knowledge about their corresponding part of reality. The architecture specifies which corresponding parts this shall be to ensure longevity and a maximised user mass potential. E.g. aggregating activities and resources, types and instances will cause failure. They need to become separate, autonomous, so-called first class citizens in a smart grid capable of in-depth interoperability. And, time-varying aggregation of such basic system elements needs supporting.

The lessons learned and information obtained while applying the insights and guidelines within STORY will be absorbed and reported by subtask T3.4.2. It also will elaborate software providing support when following the guideline herein.

10 Acronyms and terms

NTP	Network Time Protocol	
PTP	Precision Time Protocol	
Erlang	Ericsson programming language	www.erlang.org
NFC	Near field communication	
LoRa	Long Range Wide Area Network	
SSOT	Single Source of Truth	
VHDL	Very High Definition Language	Hardware programming language
ASCII	American Standard Code for Information Interchange	
UNICODE	Computing industry standard for the consistent encoding,	en.wikipedia.org/wiki/Unicode

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3

	representation, and handling of text	
XML	Extensible Mark-up Language	
GPS	Global Positioning System	
TRL	Technology Readiness Levels	Cf. annex below
IoT	Internet of Things	
WoW	World of Warcraft	Massively multiplayer online game
GUI	Graphical user interface	
CNC	Computer (or computerized) numerical control	
DNC	Direct numerical control	
SCADA	supervisory control and data acquisition	

11 References

[1] Valckenaers, Paul, Van Brussel, Hendrik. *Design for the Unexpected: From Holonic Manufacturing Systems towards a Humane Mechatronic Society*. Butterworth-Heinemann[12], November 2015.

[12] store.elsevier.com/product.jsp?isbn=9780128036624

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Annex: h2020-wp1617-annex-g-trl_en.pdf

TRL 1 – basic principles observed

TRL 2 – technology concept formulated

TRL 3 – experimental proof of concept

TRL 4 – technology validated in lab

TRL 5 – technology validated in relevant environment (industrially relevant environment in the case of key enabling technologies)

TRL 6 – technology demonstrated in relevant environment (industrially relevant environment in the case of key enabling technologies)

TRL 7 – system prototype demonstration in operational environment

TRL 8 – system complete and qualified

TRL 9 – actual system proven in operational environment (competitive manufacturing in the case of key enabling technologies; or in space)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 646426

Project STORY - H2020-LCE-2014-3