# On the Technological and Methodological Concepts of Federated Embedded Systems

Avenir Kobetski
Swedish Institute of Computer Science (SICS)
Kista, Sweden
avenir.kobetski@sics.se

Jakob Axelsson
Swedish Institute of Computer Science (SICS)
Kista, Sweden
jakob.axelsson@sics.se

***Abstract* — Traditionally embedded systems are developed with a specific control task in mind, and are able to affect only a limited set of actuators, based on measurements from a limited set of sensors. With the arrival of cheap and efficient communication technology, this traditional picture is starting to change. It is our belief that future embedded systems will interact with each other, forming federations to provide new emergent services to their users. With this in mind, a pre-study was performed to discern the main concepts of such federations and the related challenges that need to be addressed. This has led to two parallel research directions, presented in this paper. One is focusing on the enabling technology that is needed for dynamic creation of new types of federations, while the other deals with the methodological concepts for creation of ecosystems in which federations of embedded systems can be dynamically formed.**

***Keywords — federated embedded systems, reconfigurable software, software ecosystems***

## I. INTRODUCTION

The invention of Internet revolutionized knowledge sharing between people. The invention of smartphones revolutionized the mobile phone industry while data sharing took another leap, both with respect to the used technology and the sheer scale of exchanged data. It is quite safe to predict that the next large leap in this direction will come when *embedded systems (ES)* start to interact by exchanging data and collaborating towards common goals.

While today most ESs are developed for a particular application and operate on limited sets of sensor and actuator signals, interacting ESs will have a much wider choice of signals to use, offering vast opportunities for new emergent services. An example of such a service is a traffic intersection management system that collects data from the approaching vehicles and transforms that data into control signals for the vehicle speed, achieving a smooth traffic flow through the intersection. Numerous other examples of emergent services, either real or imagined, can be found in many different application domains, such as automotive, transportation, construction, healthcare, manufacturing, energy, etc. [1, 2, 3, 4]. This has spawned several interesting and somewhat related research directions, such as cyber-physical systems, internet of things, systems of systems, ubiquitous systems, etc., each focusing on slightly different aspects of the concept. For a more detailed literature review, see [5].

In our work, we use the term *federated embedded systems (FES)* to emphasize the focus on embedded systems and the concepts that are needed in order for ESs to be able to interact with each other in a meaningful way. The interactions are modeled as federations of systems, both embedded and traditional, where each system in some way benefits from participation in the federation. The FESs may either be static or evolve dynamically, both with respect to their functionality and composition. In many ways, the term FES is related to the field of cyber-physical systems, but is more focused on the concepts needed for the creation and operation of federations.

To reach the FES vision, significant advances in several research directions are needed. This includes mechanisms to dynamically join, operate in, and leave federations, as well as methods for handling security, software and hardware faults, conflicting requirements, information modeling, software architecture, privacy issues, embedded systems technology, and others. While the emergent FES functionality should bring some benefits to all participating ESs, the individual ES functionality, especially the safety critical one, must be maintained. Also, the concepts need to prepare for the FESs being open, both in the sense of openness towards new FES members and in terms of open innovation, with third party developers providing software to the ESs that would enable them to participate in a specific federation. Thus, new business models will be needed that support new types of software ecosystems.

Obviously, the challenges are numerous. To get a better understanding of the FES concepts and challenges, we conducted a pre-study on the FES subject, based on a series of workshops together with several industrial partners [3]. A portfolio of applications from different application domains was collected and used as the basis for the discussions. It became evident that in order to reach the FES vision, both technological and methodological advances are needed.

The main point of this paper is to summarize the concepts of our pre-study, and to present concrete work towards the FES vision that followed. In Section II, some high level concepts of FES are presented. Section III describes a software component concept that enables dynamic software reconfiguration during runtime in vehicle applications. Section IV presents our work within software ecosystem methodology, Section V reviews some related work, while Section VI concludes the paper.

## II. HIGH LEVEL CONCEPTS

In this section, main FES-related concepts that were put forward during our prestudy [3] are summarized. Basically, the concepts were partitioned into four groups, divided by two conceptual axes, *technology* vs. *methodology* and *system-level* vs *federation-level* concepts. In the following subsections, these concepts are shortly presented.

### A. System-level Technology Concepts

On the level of individual systems, some basic technologies are needed in order for the ESs to be able to participate in federations with other systems. First of all, in order for the federations to form and for the ESs to contribute to and benefit from the FESs, the systems must be able to communicate with each other. Thus, technology for external communication is needed.

Secondly, federations and the services that they provide will often be evolvable and unforeseen at the design time of individual ESs. For this to happen, it should be possible to dynamically add and update software to the ESs at runtime. In consequence, if safety-critical functionality is allowed to be affected in such a way, there should be fault handling mechanisms that monitor how the new software complies with the system requirements, both functional and non-functional, and resort to inbuilt fall-back functionality if needed. Also, faults can be caused by the newly added software containing conflicts with other parts of installed software. Thus, logical software conflicts should be detected and handled.

In a recent work, a conceptual model for dynamically updatable embedded software was proposed [6]. It builds upon AUTOSAR [7], an architecture standard being widely used in the automotive sector. Currently, the concept is being further developed, in parallel with the development of tools and a demonstrator to show different FES application scenarios. The model is highlighted in Section III.

### B. Federation-level Technology Concepts

At the federation level, the technology needs are more intricate. Standardized protocols are needed in order for different kinds of ESs to cooperate. Such protocols should describe the communication details and the rules to which participating ESs will have to abide while functioning within a certain federation. In most federations, different types of ESs will play different roles, thus following different sets of rules.

New fault handling mechanisms are needed to handle the emergent behavior. On one hand, faults that would never exist in separated ESs may occur due to interactions. On the other hand, ESs may assist each other to overcome or to reduce the effects of faults. Again, faults can be caused by conflicting functionality. However, this time the level of abstraction is higher and the conflicts are expected to occur between ESs and their differering requirements. Related topics of importance are trust and uncertainty management in the scope of a federation.

### C. System-level Methodology Concepts

In order for the FES to become a reality, methodology related concepts must not be neglected. Today, an ES is generally produced by one original equipment manufacturer (OEM), as part of a larger product. It often contains parts, both hardware and software, from different suppliers, while the OEM is responsible for integration.

With the idea of dynamic software, the number of participating software producers will be even higher, and since third-party developers will be able to add software without the involvement of the OEM, roles and responsibilities change between the parties. This, in turn, will change information flows during development and affect tools. A successful ES will no longer be one that only provides a certain function, but one that serves as a useful platform for adding new functionality on top of it

### D. Federation-level Methodology Concepts

While interactions between different actors that contribute to ES development may be complex, they become even more entangled at the federation level. To pave the ground for evolving and persistent federations, well defined business models are crucial. On the one hand, such models should provide opportunities for different parties to benefit from the emergent functionality, encouraging them to participate in the operation and development of the federation. On the other hand, the responsibility for the federation should be clearly defined. In other words, all aspects of the emergent functionality should be owned and maintained by some stakeholder.

The distribution of responsibilities and benefits between stakeholders is a challenging question. Even more challenging is how to do this dynamically in order to keep up with the changing nature of FES. The solution should include possibilities to allow new parties to take part in the federation operation, to let existing parties to take on new roles if needed, and to adapt responsibilities to the evolving FES functionality.

It seems clear that the technological development in itself is not sufficient for the creation and evolution of lasting FESs. The methodological aspects of federation operation should be carefully investigated. In Section IV, our initial work on this subject is presented.

## III. DYNAMIC SOFTWARE RECONFIGURATION IN EMBEDDED SYSTEMS

In this Section, a component model that allows dynamically adding and removing parts of ES software is presented. This model is a concrete example of a system level enabling technology that opens up for third party developers to add new services to ESs, ultimately creating opportunities for FES formation. The model is primarily tailored for automotive applications and builds on the AUTomotive Open System ARchitecture (AUTOSAR) standard [7]. However, the standard is not limited to the automotive world. In fact, it is suitable to all ES applications where the basic software (e.g. task scheduler, device drivers, hardware abstractions, etc.) is common to several control units and can be standardized.

In the following subsections, the AUTOSAR architecture is briefly introduced, followed by our extensions to the concept together with a few implementation details. Finally, some safety and security related remarks are collected.

### A. The AUTOSAR Concept

AUTOSAR is structured around a layered software architecture that decouples the basic software (BSW) from the application software (ASW). This is accomplished by means of a component model, and a middleware called the runtime environment (RTE). Using AUTOSAR, ASW is modeled as a collection of software components (SW-C), which are in many ways similar to established component models like Koala [8], that communicate with each other and the rest of the system (e.g. standardized BSW) through so called *ports*. The internal functionality of the component only accesses its ports.

The actual communication between the ASW components, as well as their access to the lower layers, is taken care of by the RTE by interconnecting appropriate ports. This eases reuse of parts of the ASW, while RTE adds flexibility and scalability to the AUTOSAR architecture, allowing application SW-Cs to be easily redistributed between different control units simply by reconfiguring the RTE.

However, AUTOSAR has been designed to execute with limited resources and hence configuration of the system, such as allocation of SW-Cs to control units, and connection between SW-C ports, is done at design time with no structural dynamics during execution. The configuration is described in xml-files separate form the source code. These description files are used before deployment to generate C code that links ASW to BSW. Any changes in configuration require the software to be rebuilt and the control unit to be reprogrammed.

### B. Dynamic Software Components

In [6], initial work on a conceptual model that extends the AUTOSAR architecture to allow software update at runtime was proposed. The key is to extend the set of ordinary application SW-Cs with dedicated SW-Cs for running additional software, hereafter called *plug-in software*, which is installed after the vehicle has left the factory. In this work, only plug-in enabling concepts are presented, while the internal plug-in functionality, which actually defines the rules for how the ES may act on the federation level, see Section II.B, is not considered at this point, but will be addressed in the future.

Figure 1 gives an overview of how the plug-in concept relates to the underlying AUTOSAR based software. In the figure, dotted lines are used to show the plug-ins and their connections, whereas solid lines are used for the AUTOSAR SW-Cs and their links. For the concept to work, the OEM must provide plug-in enabled SW-Cs, which to start with only contain a Java virtual machine (VM) and an API that will be available to the plug-ins in the form of input and output ports, connected to the rest of the system through AUTOSAR RTE.

Also, one external communication manager (ECM) SW-C is needed, capable of communicating with a pre-defined external trusted server so that plug-ins can be installed, updated, and uninstalled at runtime. Furthermore, ECM serves as a gateway for plug-ins to communicate externally, which allows transferring information to and from off-board services, and participating in FESs. Finally, the AUTOSAR RTE must be configured so that ECM is connected to the plug-in SW-Cs.

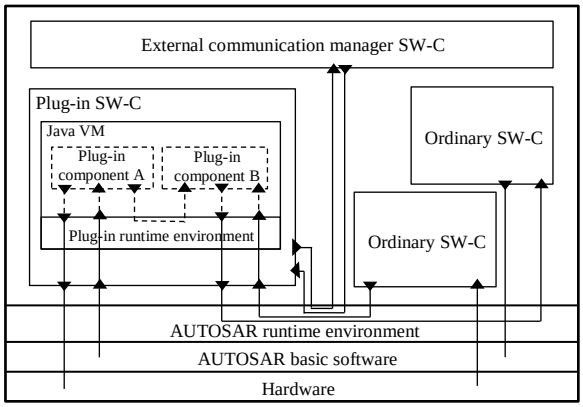


**Figure 1. The structure of a dynamic software component model.**

### C. Internal communication

Inside the plug-in SW-Cs, AUTOSAR concepts are replicated as far as possible. Plug-in components communicate with the rest of the system through ports, while the connection details are configured in the plug-in runtime environment (PIRTE). Differently from the AUTOSAR RTE, the PIRTE contains both a static and a dynamic part. The static PIRTE part interfaces with SW-C ports and maps them into Java API signals. The dynamic part, updated each time any plug-in SW-C is updated, handles plug-in ports and their connections.

Plug-in ports can either access built-in functionality through the API provided by PIRTE, or they can be connected to ports on other plug-ins, again mediated by PIRTE. This is done even if the plug-ins are part of the same application, allowing dynamic reallocation of plug-ins between control units if needed. For example, if plug-in B in Figure 1 were reallocated, PIRTE would pass the connection to the AUTOSAR RTE that in its turn would forward it through the databus to the correct control unit.

Since it is not practically possible for the OEM to provide (and connect) SW-C ports for all imaginable future plug-in ports, the communication between plug-ins on different control units is done through dedicated SW-C ports (one pair of ports per control unit), which are fully connected to each other in AUTOSAR RTE. As a result, PIRTE needs to provide the address of the receiving control unit, the receiving plug-in, and the receiving port in that plug-in with the message that is passed to the data bus. Note that all these communication details only affect PIRTE and are transparent to the plug-ins.

### D. Safety & Security

To provide a basic level of security, plugin software is sandboxed in as far as possible. First of all, plug-ins can only access the underlying system through the ports of the plug-in SW-C. It is up to the ECU developers to decide which ports to provide and how data received from these ports should be handled. If that data is used to control the underlying system, it is important that (non-reconfigurable) fallback mechanisms, with the authority to override plug-in actions, are in place. Secondly, Java VM executes in its own thread and with its own memory areas and network messages. This avoids competition for resources with the built-in functionality. Plug-ins are thus executed under a best effort scheme, whereas built-in software has predictable behavior.

A potential security threat is the installation of plug-ins. In this concept, it is only allowed to install plug-ins from a trusted server at a pre-defined address. In this way, much of the firewall issues are moved from the resource-constrained embedded system to a server. To change the trusted server address requires reprogramming of the ECU's built-in software, which has its own security mechanisms.

## IV. ECOSYSTEMS OF EMBEDDED SYSTEMS

In order to create a successful concept for FES, it is not sufficient to only look at the technical implementation, but one must also study how to organize development of the systems and to achieve sustainable business models, as described in Section II.D. The key concepts of FES actually provide several opportunities from a business perspective:

- The plug-in components can be used by an OEM to add new functionality very rapidly, thereby being more responsive to market trends or to requirements from niche users.
- Third-parties can develop plug-in components to extend the functionality beyond what the OEMs conceived, similarly to how app developers extend the functionality of mobile phones.
- Systems can be integrated into systems-of-systems, whose functions are realized by distributed software. The integration in this case is handled by a separate organization.

This means that many stakeholders have an interest in the development and use of a FES, and the interrelations between them become crucial. In traditional development of ES, there is an OEM who is responsible for the end product, and who integrates subsystems from different suppliers. The suppliers develop their parts based on detailed specifications from the OEM.

In the FES setting, the OEM instead delivers an extensible product, whose success is partly a result of how well it supports the independent development of add-on solutions, and not only how well it meets the basic requirements. Based on this platform, a thriving business ecosystem [9] can be created, where third-party actors can practice open innovation and thereby extend the value of the base product.

However, this also leads to new challenges that need to be addressed when it comes to system development. For instance, ways of sharing information between different parties must be found, so that a plug-in developer can develop and test its software without full access to the overall product. Quality assurance in general is an issue, and the base product has to be tested with respect to all possible plug-ins that can be added to it. The distribution of rights and responsibilities between the parties are also crucial. Who is liable in a situation where an incident occurs? How should the streams of income be set up and divided among the parties?

Clearly, the business side and the technical side of FES are not separate. They meet in, for instance, the product architecture, that must support in a good way the development of both base products and plug-ins, and continue to do so over the time that the system evolves.

To investigate these issues, we are currently conducting an empirical research project, where we, based on case studies and interviews, try to identify the primary interfaces between stakeholders in the ecosystem. This will be used to create a reference model that explains what flows of information, money, etc. exist between them, and can be used to provide guidance on how to organize the ecosystem efficiently [10].

## V. RELATED WORK

Several studies on the subject of cooperating systems have been published. In [11], more than 40 different definitions of the term systems of systems were reviewed and classified, while the notion of federations of systems was coined in [12]. In [2], recent research advances within the internet of things field, together with a number of application scenarios, are presented. In the field of cyber-physical systems, a lot of work has been done to present both the research challenges and possible applications, e.g. [1, 13, 14, 15]. A more extensive litterature review can be found in [5].

While all the above research directions are interrelated, they focus on somewhat different questions, even though they seem to be starting to merge. For example, the IoT field has sprung out of the desire to be able to uniquely identify any physical object, with a large focus on identification and communication. The SoS research on its hand has traditionally been focusing on the engineering management studies. The CPS field seems to be the most related to the FES concept. In fact, one could argue that CPS actually is a larger field containing FES. The difference lies in the strong emphasis that the term FES places on the use of embedded systems in federations that are generally allowed to be dynamically reconfigurable, both with respect to their composition and functionality. The perceived need for a focus on open and dynamic federations of ESs was one of the reasons for our FES pre-study [3]. Another reason was to give the opportunity for the Swedish industry to add its voice to the ongoing evolution of the CPS concept, zooming in on FES related challenges.

When it comes to dynamic reconfiguration of SW-Cs, it has been studied in e.g. [16, 17, 18, 19]. Differently from the above publications, this work provides a Java-based and simple to implement concept that builds on a standardized architecture, offering good opportunities for open software development.

The term "ecosystem" was introduced by Iansiti and Levien who described the notion of business ecosystems [20]. They explained the benefits of adopting the "ecosystem-thinking" from a business perspective and discussed various strategies organizations may utilize, depending on their role within the ecosystem. However, the article does not explicitly describe how to carry out product development, or what specific characteristics products should have in order to make the best out of such an ecosystem.

The term "software ecosystem" was introduced by Messerschmidt and Szyperski [21] but was extended by Bosch [9]. In particular, Bosch extended the classical "product line-thinking" of software products, The trend towards open platforms was started because it is too expensive for an manufacturer to develop alone all the functionality that customers would wish for, and because gathering the

requirements for customization could potentially be done more efficiently through an open platform.

Hanssen and Dybå [22] described in their work a systematic overview of software ecosystems and explained several related challenges. Jansen, Finkelstein and Brinkkemper [23] presented a research agenda for software ecosystems, discussed about the main challenges involved in a technical and business level through three dimensions: a) from a software ecosystem level, b) software supply network level, and c) from the software vendor level, and also mentioned issues of formal modeling, transparency, guidelines, standards, and actions, that are of central importance.

All in all, there is very little research that looks at ecosystems specifically for ES, but the literature either is looking on pure software, or general product development.

## VI. Conclusions and Future Work

This paper consists of three main parts. Firstly, high level concepts related to FES and its constituent ESs are presented. These concepts are further divided into those that are connected to technology development and those that relate to product and process development methodologies, e.g. business models. Secondly, our work in the technological direction is presented. This work extends the AUTOSAR architecture with the concept of dynamic component models, thus allowing installation of new plug-in software into vehicles at runtime, opening up for implementing FES governing interaction rules long after the vehicles have left factory. Although the AUTOSAR standard is from the automotive industry, the concepts are quite general and of value in other embedded system domains. Thirdly, we present our initial work in the methodology direction, aiming at defining the business models necessary for dynamic FES ecosystems.

While our work on FES methodology is currently just taking off, see Section IV for a discussion of future challenges, there is more to say about the continuation in the technological direction. More realistic tests of the concepts presented in Section III are needed to get deeper insights about the strengths and weaknesses of the proposed solution. Stressing the system in order to test robustness is also important. For example, this could help to understand what happens if power goes off during the installation of a plug-in, or how to react to the loss of messages between plug-ins, etc. To increase the practical usefulness of the proposed architecture, tools that aid in generating plug-in runtime environment will be developed. Also, in theory the concepts should be applicable to safety critical applications. However, for this to work, there is a need of fault handling mechanisms, both locally at the ES level, provided by the ES developers, and at the FES level. Trust and conflict management mechanisms are other intricate but interesting research questions. Once the basic concepts and the simulation environment are in place, such more complex issues are ready to be adressed.

## Acknowledgements

The projects presented in this paper are supported by Vinnova (grants no. 2012-02004 and 2012-03782), Volvo Cars, and the Volvo Group.

## VII. Bibliography

[1] CPS Summit Report, 2008. [Online]. Available: http://varma.ece.cmu.edu/Summit/.

[2] L. Atzori, A. Iera and G. Morabito, "The Internet of Things: a Survey," *Computer Networks*, 2010.

[3] A. Kobetski and J. Axelsson, "Federated Robust Embedded Systems: Concepts and Challenges," Swedish Inst. of Computer Science, 2012.

[4] D. Rylander and J. Axelsson, "Using Wireless Communication to Improve Road Safety and Quality of Service at Road Construction Work Sites," in *IEEE Vehicular Networking Conference*, 2012.

[5] A. Kobetski and J. Axelsson, "Federated Embedded Systems – a review of the literature in related fields," Swedish Inst. of Computer Science, 2012.

[6] J. Axelsson and A. Kobetski, "On the Conceptual Design of a Dynamic Component Model for Reconfigurable AUTOSAR Systems," in *Workshop on Adaptive and Reconfigurable Embedded Systems*, 2013.

[7] "AUTOSAR consortium," [Online]. Available: http://www.autosar.org/.

[8] R. van Ommering, F. van der Linden, J. Kramer and J. Magee, "The Koala component model for consumer electronics software," *IEEE Computer*, vol. 33, no. 3, pp. 78-85, 2002.

[9] J. Bosch, "From Software Product Lines to Software Ecosystems," in *Proceedings of the 13th Intl. Software Product Line Conference*, 2009.

[10] E. Papatheocharous, J. Axelsson and J. Andersson, *Towards an Innovative Open Ecosystem Infrastructure for Federated Embedded Systems Development*, Submitted to the 1st Intl. Workshop on Software Engineering for Systems-of-Systems, 2013.

[11] J. Boardman, S. Pallas, B. J. Sauser and D. Verma, "Report on System of Systems Engineering," Stevens Institute of Technology, 2006.

[12] A. P. Sage and C. D. Cuppan, "On the Systems Engineering and Management of Systems of Systems and Federations of Systems," *Information-Knowledge-Systems Management*, vol. 2, pp. 325-345, 2001.

[13] E. Lee, "Cyber Physical Systems: Design Challenges," in *Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, 2008.

[14] L. Sha, S. Gopalakrishnan, X. Liu and Q. Wang, "Cyber-Physical Systems: A New Frontier," *Machine Learning In Cyber Trust*, vol. 1, pp. pp. 3-13, 2009.

[15] R. Rajkumar, I. Lee, L. Sha and J. Stankovic, "Cyber-Physical Systems: the Next Computing Revolution," in *the 47th Design Automation Conference*, New York, NY, 2010.

[16] R. Anthony, A. Rettberg, D. Chen, I. Jahnich, G. de Boer and C. Ekelin, "Towards a dynamically reconfigurable automotive control system architecture," *Embedded System Design: Topics, Techniques and Trends*, pp. 71-84, 2007.

[17] M. Felser, R. Kapitza, J. Kleinöder and W. Schröder-Preikschat, "Dynamic software update of resource-constrained distributed embedded systems," *Embedded System Design: Topics, Techniques and Trends*, pp. 387-400, 2007.

[18] J. Polakovic, S. Mazare, J. B. Stefani and P. C. David, "Experience with safe dynamic reconfigurations in component-based embedded systems," *Component-Based Software Engineering*, pp. 242-257, 2007.

[19] Y. Vandewoude and Y. Berbers, "Run-time evolution for embedded component-oriented systems," in *Proc. Intl. Conf. on Software Maintenance*, 2002.

[20] M. Iansiti and R. Levien, "Strategy as ecology. , 82(3), 68-81," *Harvard business review*, vol. 82, no. 3, pp. 68-81, 2004.

[21] D. G. Messerschmitt and C. Szyperski, Software ecosystem: understanding an indispensable technology and industry, MIT Press, 2003.

[22] G. K. Hanssen and T. Dybå, "Theoretical foundations of software ecosystems," in *Proc. of the 4th Software Ecosystems Workshop*, 2012.

[23] S. Jansen, A. Finkelstein and S. Brinkkemper, "A sense of community: A research agenda for software ecosystems," in *Proc. of the 31st International Conference on Software Engineering*, 2009.