#08AE-67

The Role of Software Engineering in Future Automotive Systems Development

Siobhán Clarke[1], Brian Fitzgerald[2], Paddy Nixon[3], Klaus Pohl[2], Kevin Ryan[2], David Sinclair[4], Steffen Thiel[2]

Lero – The Irish Software Engineering Research Centre
[1]Trinity College Dublin, Dublin, Ireland
[2]University of Limerick, Limerick, Ireland
[3]University College Dublin, Ireland
[4]Dublin City University, Dublin, Ireland

ABSTRACT

The amount and complexity of software in automotive systems is constantly increasing. Today's luxury cars include numerous electronic control units. A large part of the functionality of these units is driven by software.

In the future even more software-intensive automotive systems are expected as automotive manufacturers and suppliers tend to integrate and combine applications on more powerful platforms. The increasing amount and complexity of software in these platforms has led to the situation where software engineering has become an essential discipline within automotive systems development.

This paper identifies essential areas of software engineering that will have a significant impact on future automotive systems and systems development. Particularly, it discusses how the software engineering discipline can be developed in the context of overall automotive systems development while considering essential requirements and constraints that are or will become prevalent in this domain.

1 INTRODUCTION

Over the past decade, the amount and complexity of software in automotive systems has increased substantially. Today's luxury cars include more than 70 electronic control units (ECUs) that operate as partly networked systems to improve passenger comfort, safety, economy, and security. Many of these systems are driven by software [1].

In the future more software-intensive automotive systems are expected as automotive manufacturers and suppliers tend to integrate and combine applications such as driver assistance, car dynamics, and airbag control systems on more powerful platforms (e.g., [2]). The integration and combination of automotive applications is particularly crucial for the exploitation of strategic reuse that can drive the reduction of time-to-market and of the costs of automotive applications.

The increasing amount and complexity of software in automotive platforms has led to the situation where software engineering has become an essential discipline within automotive systems development [3]. Manufacturers and suppliers already make great efforts in adopting, tailoring, and improving software engineering processes and practices while working in globally distributed teams to build high-quality systems (e.g., [4]).

This paper identifies essential areas of software engineering that will have a significant impact on future automotive systems and systems development. Section 2 motivates domain-oriented software engineering as a general approach to provide effective research results in a given application domain, such as automotive. Sections 3-6 discuss important areas of software engineering that we believe will have an effect on automotive systems development. The areas include software product line engineering (Section 3), global software development (Section 4), service-oriented architectures (Section 5), and mathematics applied to software engineering (Section 6). We describe particular challenges with respect to the automotive domain and outline approaches within the area to address these challenges. Finally, Section 7 concludes the paper with a summary.

2 DOMAIN-ORIENTED SOFTWARE ENGINEERING

Software Engineering is a broad, dynamic and multi-faceted field, where research can vary from highly theoretical, abstract models, through the development of numerous branches of technology, into empirical studies of industrial practice.

The development of quality software requires, in addition to software engineering capabilities, knowledge about the application domain in which software is going to operate. Research in Software Engineering has progressed from seeking universal solutions, whether through concepts, notations, methods or tools, to the development of domain-specific solutions that take advantage of domain specificities. Domain-specialised workshops, conferences and publications have increased and many general conferences and journals devote sessions or issues to specific domains (e.g., [5]).

Domain-oriented research has the potential to

- narrow the problem focus,
- reflect domain standards, constraints, models and priorities,
- facilitate rapid validation, and
- accelerate industrial uptake.

We believe that domain-oriented Software Engineering can provide more effective research results as it includes the peculiarities of the given domain. Once domain-specific solutions have been developed they are available for adaptation and exploitation in other domains. Research outcomes that are proven useful in a number of domains may be generalised to provide domain independent solutions.

Within Lero, the Irish Software Engineering Research Centre, we have chosen the automotive domain as our initial focus. Within that we have identified different research areas that we believe will have a significant impact on the development of future automotive systems. The research areas include:

- Software Product Lines
- Global Software Development
- Service-Oriented Architectures
- Mathematics Applied to Software Engineering

In the following sections we will introduce these areas, describe particular challenges regarding automotive system development, and propose approaches and research directions to address these challenges.

3 SOFTWARE PRODUCT LINES

3.1 INTRODUCTION

Product line approaches are well-known in many manufacturing industries, such as consumer electronics, medical systems and automotive [6]. In recent years, approaches with similar roots have rapidly emerged as important paradigms within Software Engineering, so called Software Product Line (SPL) development approaches [7, 8].

As automotive manufacturers and suppliers design and implement complex applications, such as driver assistance systems [9], they strive for mechanisms that allow them to implement major functionalities on integrated platforms. The move towards those larger platforms has provided an opportunity for strategic reuse of software components. One key aspect of this strategic reuse is the idea to build a variety of similar systems with a minimum of technical diversity. This has resulted in a growing interest in SPL approaches both in the software engineering and the automotive systems domains.

3.2 CHALLENGES

There are several challenges related to the adoption of product line approaches in automotive software-based systems (e.g., [3, 10]). The challenges include the following topics:

- Management of large numbers of variants
- Development of product line architectures
- Integration of model-driven practices

3.2.1 Management of Large Numbers of Variants

Many automotive suppliers and manufacturers such as Cummins [7], Bosch [2, 4], and DaimlerChrysler [11] use a product line approach so as to be able to build different variants of their products for use within a variety of automotive systems. The size of the product lines is usually large since only in this case significant economies of scale can be achieved. Therefore, automotive software platforms are typically developed in a way such that they can be customized and used in hundreds of products simultaneously (e.g., [4]). These platforms could easily incorporate thousands of variation points and configuration parameters.

Managing this amount of variability is extremely complex and requires sophisticated modelling and representation techniques that can cope with large data sets. In particular, there is a strong need for appropriate approaches that support the different stakeholders in carrying out their development tasks in software product line efforts with a large number of product variants [12].

3.2.2 Development of Product Line Architectures

Software architecture provides the key framework for the earliest design decisions taken to achieve functional and quality requirements. An architecture for a family of systems needs to identify the commonality among different systems and must explicitly include a variability documentation. Architecting automotive systems is a complex and challenging design activity, and architecting a product family is even more challenging. It involves making

decisions about a number of inter-dependent design choices that relate to a range of design concerns. Each decision requires selecting among a number of design options; each of which impacts differently on various quality attributes. Additionally, there are usually a number of stakeholders participating in the decision-making process with different, often conflicting, quality goals, technical and project constraints, such as existing platforms, cost and schedule.

3.2.3 Integration of Model-driven Practices

The intelligent use of models promises to be one of the major foundations for efficient processes in automotive systems engineering. Models can be used to describe the different forms of knowledge which are captured and transformed during the engineering process, such as requirements, the high-level or detailed design, the implementation or test cases. This is the foundation for tooling and automation, which in turn promotes efficient engineering processes [3].

However, the current use of models in automotive systems engineering does not realise its full potential. For instance, models are used in isolated areas, without an integrated flow of information. Often models are used only in a semiformal way as a form of communication on the whiteboard or as illustrations in a textual specification.

This lack of clearly and precisely defined semantics undermines the use of real model-*driven* approaches, where models are expressive enough to be used in powerful interactive tools and the automatic derivation of further artefacts including the implementation.

3.3 APPROACHES AND RESEARCH DIRECTIONS

Our research focuses on the following approaches to address the challenges mentioned:

- Visually informed variability management
- Architecture-Based Development
- Seamless model-driven development

3.3.1 Visually Informed Variability Management

As mentioned previously, industrial size automotive product lines can easily incorporate thousands of variation points and configuration parameters for product customization. A promising approach to address the complexity problem in automotive systems engineering is to improve the efficiency of variability management and product derivation.

Variability management is the process by which the variability of the product line development artefacts (e.g., architectural models, software components, and hardware components) is planned, documented, and managed throughout the development lifecycle. Variability management supports critical product line engineering tasks such as product derivation. Variation points identify locations in product line artefacts at which variation will occur [13].

A large part of the application engineering activities in an established and optimized product line approach consist of reusing the platform artefacts from domain engineering and binding the variability as required for the different applications/products. In this process, the variability is resolved and fixed to the particular customer product. Potentially pre-developed customer-specific application components are integrated into the product infrastructure. If domain and application engineering are coordinated in such a way then product production becomes more a configuration and composition than a development activity. In this way, a mass-customization of products can be achieved and the upfront investment in domain engineering can be more than justified.

Systematic variability management and product derivation can and should be supported by visualisation techniques and tools that support the understanding, management, and effective use of product line development artefacts, their built-in variability, and the dependencies among them. With suitable techniques such visualisations can also amplify the cognition about large and complex data sets created and used in industrial software product line engineering. Our research is focused on exploring the potential of visual representations such as trees and graphs combined with the effective use of human interaction techniques such as dynamic queries and direct manipulation when applied in a software product line context.

3.3.2 Architecture-Based Development

Software architecture embodies some of earliest design decisions, which are hard and expensive to change if found flawed during downstream development activities. The role of software architecture in a family of system becomes much more vital as architectures of individual products are derived from the core architecture. Hence, any flaw in the core architecture usually has ramifications for the achievement of required quality attributes for individual products in a family. A systematic and integrated approach is required to address architectural issues throughout the software development lifecycle.

Hofmeister et al. [14] have proposed a general model of software architecture design. This model has three activities: architectural analysis, architectural synthesis, and architectural evaluation. However, it does not consider the post-architecting activities, which are equally vital to ensure that the intent behind the architecture design remains correct during implementation and maintenance of the software architecture. One of the main characteristics of architecture-based development is the role of quality attributes and architecture styles and pat-

terns, which provide the basis for the design and evaluation of architectural decisions in this development approach. Figure 1 shows a high level process model of architecture-based development that consists of six steps, each having several activities and tasks. Our research is focused on developing an integrated framework which consists of methods, models and tools to systematically elicit and model requirements, effectively and efficiently transform requirements into a product line architecture, rigorously evaluate that architecture, and establish and test traceability between requirements, architecture and implementation.

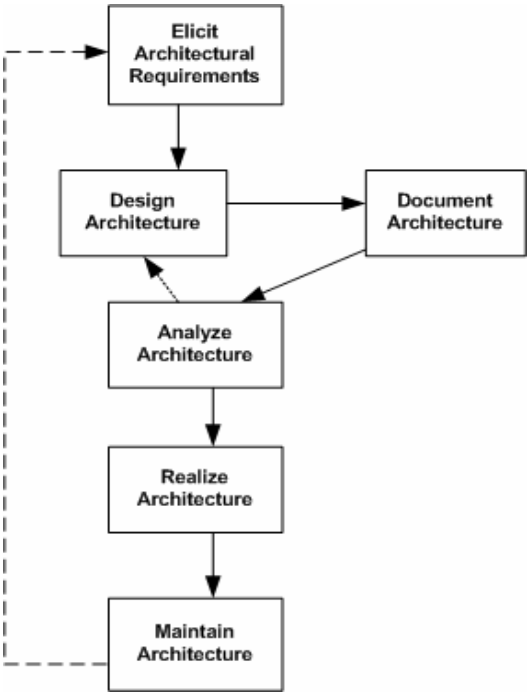

Figure 1 – Architecture-Based Development Process

3.3.3 Seamless Model-driven Development

For seamless model-driven development with a flexible combination of interactive tools and automation we need semantically rich models. As a foundation for this, we require well-defined languages that allow describing the knowledge used in the particular domain.

Consequently, if we want to apply model-driven approaches in an SPL context we require languages to describe typical SPL aspects (variation points, variants, features, configurations, realization of variability). These modelling languages also have to support the different conceptual levels of domain and application engineering.

Similarly, if we want to apply model-driven SPL approaches in an automotive engineering context we also require languages to describe domain-specific knowledge, for instance the dynamic behaviour of electronic components, the interplay between software and hardware or the communication primitives exchanged on a CAN-Bus network [9].

As part of our research we are developing such modelling languages that can be used to describe the models processed in model-driven approaches. Within that we are especially interested in modelling languages for the SPL automotive context. Models described in the modelling languages can be used as a foundation for interactive tools or automation.

For instance, we have defined a metamodel for feature models [15]. Such models can be used in tools which allow us to (a) visualise the complex dependencies between the numerous features and (b) interact with the related feature configuration. At the same time, such feature models can be used as input for model-driven product derivation. For instance, we developed a model-driven approach with model-transformations described in ATL [16] that derives the architecture for a particular application from the domain architecture for the overall product line. The decision which components are included in the application-specific architecture are based on feature configurations based on the aforementioned metamodel.

4 GLOBAL SOFTWARE DEVELOPMENT

4.1 INTRODUCTION

The significance and growth of software in the automotive components industry has led to a move towards standardisation (e.g., [9]) across the sector. New software component suppliers have the potential to claim a significant share of this global market.

Global Software Development (GSD) is the practice of distributing software development across multiple sites, either within the same organisation or across different organisations. Various forms of GSD are possible: the entire software development process can be outsourced to another company, components of a software product can be developed by another organisation or another division within the same company, or a phase of the development cycle (such as system testing) may be carried out at a remote site, either in a subsidiary or a third-party organization. Globalisation has resulted in globally distributed software development with many organisations setting up software development centres in Eastern Europe, China and India that collaborate with their counterparts.

4.2 CHALLENGES

GSD offers many potential benefits to the automotive sector including reduced cost of development, quicker time to market through ‘follow-the-sun’ development, access to new markets and customers, increased innovation and shared best practice, and access to multi-skilled labour regardless of location.

However, GSD also introduces additional problems relating to communication, coordination and control of the development process. These arise due to the

distances involved in three dimensions – geographical, temporal, and socio-cultural, as shown in Table 1.

Table 1: Different Dimensions in Global Software Development

	Temporal Distance	Geographical Distance	Socio-cultural Distance
Communication	Improved record of communications Reduced opportunities for synchronous communication	Closer proximity to market Access to remote skilled workforces Face to face meetings difficult	Innovation and sharing best practice Cultural misunderstandings
Coordination	Coordination needs can be minimised Typically increased coordination costs	More flexible coordination planning Reduced informal contact can lead to lack of critical task awareness	Greater learning and richer skill set Inconsistent work practices can impinge on effective coordination Reduced cooperation arising from misunderstanding
Control	Time zone effectiveness can be utilised for gaining efficient 24x7 working Management of project artefacts may be subject to delays	Communication channels can leave an audit trail Difficult to convey vision and strategy Perceived threat from training low-cost "rivals"	Proactiveness inherent in certain cultures Different perceptions of authority can undermine morale Managers must adapt to local regulations

4.3 APPROACHES AND RESEARCH DIRECTIONS

Within the GSD research area there are a number of interesting research themes that we expect to have a significant impact on future automotive systems development. These include Agile and Open Source approaches, the socio-organisational influences on global software development, and the future positioning of small-to-medium sized enterprises within global value chains.

- **GSD for SMEs.** Enabling small and medium sized enterprises (SMEs) to benefit from GSD by adapting the GSD processes that work for large corporations to the smaller operation.

- **Agile Approaches for GSD.** While agile methods would not be an obvious choice for a distributed environment, some of their principles may be effective across a virtual team.

- **Social and Cultural Aspects of GSD.** The practice of GSD has produced virtual teams comprising members from Western, high-wage economies and Eastern, low-wage economies. How can these teams work together when their cultures are so different?

- **Open Source Software.** Open source software is a hugely successful example of GSD. In many cases, excellent software is produced by teams that have never met. How has this been achieved?

- **Requirements.** If a third-party is going to develop the software, then it is essential that the requirements are clearly understood. How can this be achieved across the various flavours of GSD?

'Two-stage offshoring', for example, offers a development model for cross-continental software development. As part of these companies, the Irish sites act as a 'bridge' in their offshoring arrangements: While the US sites offshore work to Ireland, the Irish sites offshore work further to India and hence, have experience of being both customer and vendor in two-stage offshore sourcing relationships.

Our research has found how tailored agile development practices can reduce the challenges inherent in GSD. Also, the reported potential benefits of GSD as listed above are being studied for a deeper understanding, leading to their greater realisation for companies globalising their software development activities.

Open-sourcing, is a term we have coined for the relatively recent phenomenon whereby a company may 'liberate' an open source version of hitherto proprietary software and seek to grow a global development community around it. We have investigated the obligations which need to be fulfilled by both the company and the community if open-sourcing is to be successful. We believe this may be a viable and valuable software development model for the automotive sector.

Our research has also revealed an ongoing shift from open source software (OSS) as community of individual developers to OSS as community of commercial organizations, primarily SMEs. Outsourcing to the OSS community provides ample opportunity for companies to headhunt top developers – hence moving from outsourcing to a largely unknown OSS workforce towards recruitment of talented developers from the open source community

5 SERVICE-ORIENTED ARCHITECTURES

5.1 INTRODUCTION

In-vehicle software has controlled much of the functionality of a car, from braking systems to fuel economy systems, for some time [9]. In recent years, more sophisticated software *services* have become available, such as collision avoidance or parking assistance services.

In parallel with the development of in-vehicle software services, there has been a corresponding improvement in networking technologies designed for automotive use. For example, WAVE (Wireless Access in the Vehicular Environment) [17] and CALM (Continuous Air Interface Long and Medium range) [18] support communications between vehicles and with the environment, and are likely to drive the development of standardised automotive communications protocols. In such an environment, the potential for consideration of automotive software as "service" software can be fully realised. The software-as-services paradigm underpins a flexible software deployment model that supports the dynamic provision and integration of software services on a large scale [19].

Where existing in-vehicle software services are statically included in a vehicle's capabilities, vehicle-to-vehicle and vehicle-to-infrastructure communication opens up a more dynamic market for software services, targeted as relevant to the driver [20]. The range of possibilities is large and includes services that benefit from cooperation between vehicles such as forward collision warning and adaptive cruise control, and services made available to a vehicle from environment infrastructure such as electronic parking payments and highway-rail intersection warning.

5.2 CHALLENGES

While the scope for dynamic service provision in a large scale is encouraging for automotive software providers, the complexities of engineering such services are less so. Many challenges exist, among them: composition of the relevant services in a time-bounded fashion; adaptation of the combination and behaviour of software services in a manner appropriate to the vehicle's situation; coordination of multiple vehicles' behaviour; and integration of software services into heterogeneous target environments.

There is also a need for significant flexibility in the software process, as a reduced time to market in this competitive market is a high priority. The service-oriented engineering model must therefore itself be highly adaptive and designed to consider target platform heterogeneity in flexible manner.

5.3 APPROACHES AND RESEARCH DIRECTIONS

In our research we investigate the application and combination of general advances in software engineering techniques to provide a model for building flexible service-oriented architectures that can take advantage of the dynamic service deployment possibilities associated with vehicle-to-vehicle and vehicle-to-infrastructure communication. In particular, we are using model-driven engineering (MDE) principles [21] with the aspect-oriented development paradigm [22].

MDE handles complexity by providing domain-specific languages (DSLs) that capture the semantics of the domain in a manner familiar to the domain expert, while separating the domain concerns from models of other elements of the software. In addition, MDE manages flexibility by providing automated transformations from the domain models to multiple target platforms. The aspect-oriented paradigm handles complexity by supporting the separation of the kinds of concerns that cut across other parts of the system, and would be otherwise scattered across multiple affected parts. In particular, we are employing a number of DSLs to address the challenges facing engineers of dynamic services for the automotive domain (see Figure 2).

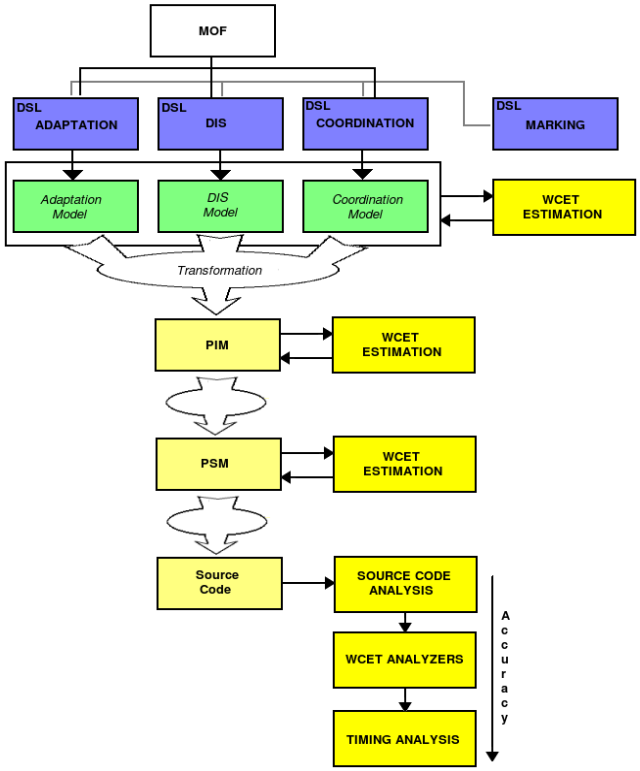

Figure 2: Model-Driven Engineering of Advanced Driver Assistance Systems.

The adaptation DSL provides a means to specify the required adaptation actions (such as merge, upgrade or install) with corresponding properties (such as priority or version number) [23]. The driver information system DSL provides a means to specify the functionality of the service including central concepts such as context, message, driver and vehicle. The coordination DSL models how vehicles coordinate with each other while satisfying constraints such as real-time.

Standard marking techniques will be used to refine the domain specific models with appropriate specifications for each target platform. The transformation process uses aspect-oriented programming techniques to compose the modelled behaviour appropriately for each targeted environment. Throughout, timing analysis is applied to monitor the time-bounded requirements of each individual service as well as the combination of services

6 MATHEMATICS APPLIED TO SOFTWARE ENGINEERING

6.1 INTRODUCTION

Today's cars consist of highly sophisticated, interacting software driven subsystems that manage various aspects of car from comfort to safety. Current "top of the line" cars can have up to 100MB of binary code distributed over 70 subsystems [24]. The complexity of current and future automotive software is so high that it is humanly impossible to test all possible cases that may occur. When the software forms part of a safety critical component this is a major cause for concern. Suitably chosen and tailored mathematical approaches can greatly increase our confidence in the reliability and correctness of highly critical software-intensive systems.

6.2 CHALLENGES

The research challenge in this area is to increase the understanding of the abstract mathematical principles underlying software, and how they can be exploited together with other techniques to improve the confidence that software works as it is intended. Our research approach is to improve and apply mathematical approaches and integrate them into the full software development life cycle.

6.3 APPROACHES AND RESEARCH DIRECTIONS

Our research focuses on three specific approaches to address the challenges mentioned:

- Hybrid Systems Design Methodology
- Feature Interaction Detection and Resolution
- SOA Integration and Migration

6.3.1 Hybrid Systems Design Methodology

This research addresses the specification, design and verification of embedded software systems in environments with continuous dynamics. These hybrid systems are characterised by both discrete and continuous state changes. These hybrid systems, and hence their software components, may also have to meet hard and soft timing constraints.

We are developing a design methodology that will assist the user in developing a specification that not only captures the behaviour of the embedded software but also captures a description of the continuous dynamics of the environment which is not only being controlled by the software but is also interacting with the software. The design methodology will guide the user from the initial analysis phase through to architectural and detailed design phases. In order to encourage adoption of the methodology it is vital that:

- the methodology is supported by a toolset; and
- initial stages of the methodology uses notations and design practices that are commonly used by the designers.

The internal formalism that is used to capture the dynamic behaviour of the subsystems is hybrid automata [25]. This formalism extends the classical finite state machines (FSM) by adding continuous state variables to the states. States with continuous state variables also have evolutionary equations that define how the continuous state variables change over time while the system is in that state. However, the design methodology will use notations that are commonly used by designers and will seek to isolate the designer from the underlying formalism.

6.3.2 Feature Interaction Detection and Resolution

Feature interaction is where at least two features within a system, which operate successfully independently, interfere with each other when used together. While these features in isolation may be correctly designed and implemented, unexpected interactions between the features may occur when these features are integrated into a larger system. Given that newer systems are usually built on top of legacy systems, the potential for interactions is very large. It is essential that these potential interactions are detected as early as possible in the software development process.

We are using software verification techniques and tools to analyse the specifications of various features, to detect when possible interactions may occur, and to suggest possible resolutions for such interactions. The approach is based on the *distillation* program transformation algorithm [26]. This will enable wide range of properties to be proved fully automatically, and will also produce counterexamples that can help to identify interactions.

6.3.3 SOA Integration and Migration

The Service-Oriented Architecture (SOA) paradigm has received a lot of attention as a methodological software engineering framework for service-based platforms. In particular, the interoperability benefits of service platforms have given new impetus to the software integration and software migration problem.

In one of our research projects we use graph theory as a rigorous mathematical approach to address the current software engineering problem of SOA-based architecture integration and migration. Graph theory provides a rich description notation with transformation and grammar aspects as well as algebraic and category-theoretic foundations [27]. Graph transformations can be used to support architecture transformation and integration. They have been successfully used to capture the structures and dependencies of components and services in software architectures [28]. However, the different types

of dynamic dependencies in service-based system architectures and their orchestration and interaction processes go beyond the current solutions for static and structural connectivity dependencies. The central question that we are addressing is how to use graph theory to provide a formally sound and effective integration solution.

7 CONCLUSIONS

This paper has discussed essential areas of software engineering and their relation to automotive systems development. Particularly, we have introduced software product line engineering, global software development, service-oriented architectures, and mathematics applied to software engineering as those areas where we expect a major impact on how systems will be developed in future in the automotive domain. We have highlighted specific challenges in these areas and have outlined approaches and research directions to address these challenges.

Our future work will include the further development and extension of these approaches and their evaluation and improvement in industrial settings.

ACKNOWLEDGMENTS

This work is partially supported by Science Foundation Ireland under grant number 03/CE2/I303-1.

REFERENCES

[1] R. Rajagopalan: “Automotive Software Market – Mix of Opportunities and Implications”, Market Insight Report, Frost & Sullivan, September 2004.

[2] A. Hein, T. Fischer, S. Thiel: “Product Line Development for Driver Assistance Systems (in German),” In: G. Böckle, P. Knauber, K. Pohl, K. Schmid (Eds.): *Software-Produktlinien: Methoden, Einführung und Praxis*, dpunkt-Verlag, 2004, pp. 193-205.

[3] M. Broy: “Challenges in Automotive Software Engineering,” *Proceedings of 28th International Conference on Software Engineering (ICSE-2006),* Shanghai, China, May 20-28, 2006, pp. 33-42.

[4] M. Steger, C. Tischer, B. Boss, A. Müller, O. Pertler, W. Stolz, and S. Ferber, "Introducing PLA at Bosch Gasoline Systems: Experiences and Practices," *Software Product Line Conference (SPLC-2004),* Boston, MA, USA 2004.

[5] 30th ACM/IEEE International Conference on Software Engineering (ICSE 2008), Experience Track on Automotive Systems. http://icse08.upb.de/calls/ automotive.html

[6] F. van der Linden: “Software Product Families in Europe: The ESAPS & CAFE Projects”, IEEE Software, 19 (4), 2002. 41-49.

[7] P. C. Clements, L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[8] K. Pohl, G. Böckle, F. van der Linden. *Software Product Line Engineering : Foundations, Principles, and Techniques*. Springer, New York, NY, 2005.

[9] J. Schäuffele, T. Zurawka. *Automotive Software Engineering: Principles, Processes, Methods, and Tools*. SAE International, Warrendale, Pa, 2005.

[10] R. Baillargeon, “Vehicle System Development: A Challenge for Ultra-Large-Scale (ULS) Systems,” First Workshop on Software Technologies for Ultra-Large-Scale Systems, Minneapolis, MN, USA, 2007.

[11] M.-O. Reiser, M. Weber: “Using Product Sets to Define Complex Product Decisions”, *9th International Software Product Line Conference (SPLC 2005)*, (Rennes, France, 2005).

[12] D. Nestor, L. O'Malley, A. Quigley, E. Sikora, S. Thiel: “Visualisation of Variability in Software Product Line Engineering”, *1st International Workshop on Variability Modelling of Software Intensive Systems (VaMoS-2007)*, Limerick, Ireland, 2007.

[13] I. Jacobson, M. Griss, P. Jonsson. Software Reuse. Architecture, Process and Organization for Business Success. Addison-Wesley, 1997.

[14] C. Hofmeister: “A General Model of Software Architecture Design Derived from Five Industrial Approaches”. 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 2005), Pittsburgh, PA, USA, 2005.

[15] G. Botterweck, D. Nestor, A. Preussner, C. Cawley, S. Thiel: “Towards Supporting Feature Configuration by Interactive Visualisation”, *Proceedings of 1st International Workshop on Visualisation in Software Product Line Engineering (ViSPLE 2007), collocated with the 11th International Software Product Line Conference (SPLC 2007)*, Kyoto, Japan, 2007.

[16] J. Bézivin, E. Breton, G Dupé, P. Valduriez. The ATL Transformation-based Model Management Framework, 2003. http://www.sciences.univ-nantes.fr/info/ recherche/Vie/RR/RR-IRIN2003-08.pdf

[17] IEEE 802.11p Task Group http://grouper.ieee.org/groups/802/11/Reports/tgp_update.htm

[18] Continuous Communications for Vehicles. http://www.calm.hu/

[19] T. Erl. “Service Oriented Architecture: Principles of Service Design” Prentice Hall, Service-Oriented Computing Series

[20] G. de Boer, P. Vogel. “Connecting the Vehicle with the Environment – Trends and Challenges” in Dagstuhl Seminar Proceedings: Mobile Computing and Ambient Intelligence: The Challenge of Multimedia, 2005.

[21] D. C. Schmidt. “Model-Driven Engineering” IEEE Computer, 39(2), 2006.

[22] R. Filman, T. Elrad, S. Clarke, M. Aksit “Aspect-Oriented Software Development” Addison-Wesley, 2004.

[23] S. Fritsch, A. Senart, S. Clarke. “Addressing Dynamic Contextual Adaptation with a Domain-Specific Language” in Workshop on Software Engineering of Pervasive Computing, Applications, Systems and Environments (SEPCASE) at ICSE 2007

[24] B. Emaus, “Hitchhiker's Guide to the Automotive. Embedded Software Universe”, SEAS’05 Workshop, 2005 http://www.inf.ethz.ch/personal/pretscha/events/seas05/bruce_emaus_keynote_050521.pdf

[25] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. H. Ho. “Hybrid Automata: An algorithmic approach to the specification and verification of hybrid systems”, Hybrid Systems, vol. 736 LNCS, pp. 209-229, Springer-Verlag, 1993.

[26] G. W. Hamilton. “Distillation: Extracting the Essence of Programs.” Proceedings of the International Workshop on Partial Evaluation and Program Manipulation, 61-70, 2007.

[27] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel and M. Löwe. Algebraic Approaches to Graph Transformation; Basic Concepts and Double Pushout Approach. In G. Rozenberg, editor, Handbook of Graph Grammars and Computing by Graph Transformation. World Scientific, 1997.

[28] D. L. Metayer, “Describing Software Architecture Styles Using Graph Grammars”, IEEE Trans. Software Eng. 24(7): 521–553. 1998.