

Towards a development approach for dynamic-integrative systems

Dirk Niebuhr
TU Kaiserslautern
Gottlieb-Daimler-Str.
D-67653 Kaiserslautern
+49-631-205-3366
niebuhr@informatik.uni-kl.de

Christian Peper
Fraunhofer IESE
Sauerwiesen 6
D-67661 Kaiserslautern
+49-631-707-219
peper@iese.fhg.de

Andreas Rausch
TU Kaiserslautern
Gottlieb-Daimler-Str.
D-67653 Kaiserslautern
+49-631-205-3360
rausch@informatik.uni-kl.de

ABSTRACT

The current progress in ubiquitous and pervasive computing not only imposes new requirements for the integration capabilities of the involved components, but also for the applied development process itself. To ensure a system's correctness and safety properties, a suitable development approach has to provide specification techniques, component models, and a common communication infrastructure to cope with the arbitrary appearance and disappearance of physical components at execution time. This paper suggests a first framework for the development of so called dynamic-integrative systems.

Categories and Subject Descriptors

D.2, D.2.1, D.2.2, D.2.4, D.2.6, D.2.11, D.2.12, D.3.1, F.3.1.

General Terms

Design, Reliability, Languages, Theory, Verification.

Keywords

System development, Development Techniques, Adaptive systems, Dynamic-integrative systems

1. INTRODUCTION

In these days the trend "everything, every time, everywhere" becomes more and more obvious: for example the internet can be accessed using mobile phones. Furthermore electronic assistants, so called "information appliances", like network-enabled PDAs, WAP-capable cell-phones and electronic books or tourist guides are available in stores all over the country.

The continuing progress of all IT-branches towards "smaller, cheaper, more powerful" mainly enables this trend. Moreover new developments in the field of materials science like midget-sensors, Organic Light Emitting Devices or electronic ink and the evolution in communications technology, especially in the wireless sector, contribute to embedding IT-components in nearly every industrial or everyday life object.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Conference'04, Month 1–2, 2004, City, State, Country.
Copyright 2004 ACM 1-58113-000-0/00/0004...\$5.00.

These IT-components, like for example a navigation system of a vehicle or a detection of a child integrated in a child safety seat, can be of different capabilities and size. Nevertheless they share the following characteristics.

IT-components

- are independent IT-systems consisting of hardware and software
- provide specific functionality with appropriate non-functional properties
- are capable of processing information and have the ability to communicate
- have strict demands for integration within a dynamic heterogeneous environment

Approaches supporting the development of such IT-components have been developed and successfully deployed over the past years like for example the Kobra-method for component-based software development [1]. Systems are no longer redeveloped from the scratch but composed of existing components ([2], [3]). Moreover a couple of various technologies exist to support dynamic integration of these components during execution time, e.g. Universal Plug-and-Play (UPnP) [4], Salutation-Framework [5], Jini Framework [6], Ronin-Agent-Framework [7] and CORBA Trading Service [8]. However a systematic development approach for those dynamic integratable IT-components and the dynamic integration of these components towards a system, providing a real added value to the user, is still a big challenge. This is why approaches supporting the compositional description of services become more and more relevant [9].

Our understanding of dynamic-integrative software-systems is that they are distributed applications systems, consisting of interacting IT-components. Further IT-components can be integrated into the applications system at any time and integrated IT-components can be removed from the system respectively they can fail as a result of a defect.

The integration of IT-components into a dynamic-integrative software-system and the removal of IT-components from a dynamic-integrative software-system have to be performed at execution time. A dynamic-integrative software-system has to guarantee, that only valid configurations of the system can be executed.

In order to achieve this, the correctness of a system has to be evaluated considering the integrated IT-components instead of

taking every condition for correctness for all possible configurations into account. If the resulting system is not correct, the foregoing correct configuration has to be executed. If it is not possible, to create a correct configuration, the system has to be halted.

The correctness of a dynamic-integrative software-system – which means the complete compliance of the realization and the requirements-specification – can not be analyzed formally because of its complexity. That is why we examine the correctness regarding some selected, exceptionally critical properties, which have to be specified using a formal language. Functional characteristics like liveness-properties ("some good thing eventually happens") have to be considered as well as non-functional characteristics like safety-properties ("some bad thing never happens") [10]. The approach does not focus on further non-functional characteristics, like for example security- and timing-properties. Existing approaches dealing with these issues, like the "Time-Triggered Architecture" [11], can be integrated into the approach presented in this paper.

The paper is structured as follows: in chapter 2 the sample application is introduced and the research topics, which are motivated by this application, are explained. Chapter 3 deals with the approach, we introduce in order to provide solutions to those research questions and finally chapter 4 gives some information what still can be done in the future after establishing the development approach.

2. RESEARCH TOPICS AND SAMPLE APPLICATION

The sample application we use to motivate and illustrate the proposed development approach for dynamic-integrative systems is a vehicle. For our sample we assume that after development and manufacturing of the vehicle a new type of child safety seat has been developed. This child safety seat has an integrated IT-component to be integrated into the car-system dynamically, although the seat was not known when the vehicle was manufactured. According to the presence respectively position of the child safety seat, the airbag and the window lift in the front respectively in the rear has to be deactivated. Furthermore the child lock has to be activated if the seat is placed in the rear. This behavior of the system can be seen in Figure 1 for the front and rear position of the child safety seat. Green color means, that the system is activated while orange color means, that the system must not be activated. In case of the child lock, the orange color means that the door is locked.

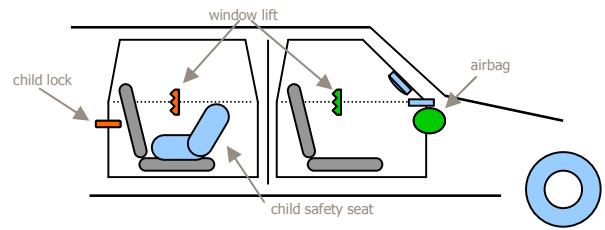
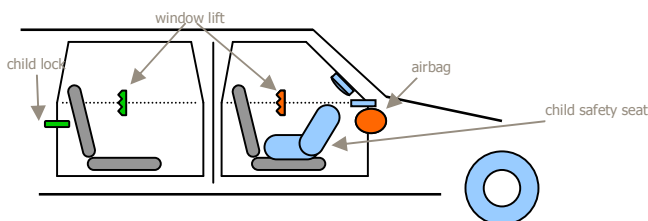


Figure 1. Behavior of the system in two different cases

The following research topics can be derived from this sample application:

- Correctness of a dynamic-integrative software-systems: If a child safety seat with an integrated chip for children detection is placed on the front passenger's seat, then the child safety seat could register itself at the on-board computer and cause the deactivation of the airbag and the window lift and furthermore the activation of the child lock. Nevertheless according to the requirements specification it still has to be possible to deactivate the child lock using a central switch (correctness of functional properties). Moreover the integration of the child safety seat requires a check of non-functional properties like for example safety-properties, too. The airbag for the front-seat passenger must not be deactivated, if the child safety seat is placed in the rear of the vehicle, for example (correctness of non-functional properties).
- Efficient dynamic integration and disintegration of IT-components: If a navigation system is integrated into a vehicle, the voice output of the navigation system could be used for additional acoustical output of critical system states like pressure drop in a tire or inflated temperature of the engine. Therefore the IT-component, which is controlling the display, has to be exchanged. Integration and exchange of an IT-component forces a complete update of the vehicles software nowadays. Because of the diversity of variants, only some selected configurations of a vehicle are tested and released. That is the reason, why it can even be necessary to exchange parts of the system, which are not affected by the change, like the control of the display in this example. This situation is not acceptable at all. Dynamic-integrative software-systems have to support the integration and disintegration of IT-components in an efficient way.
- Design of a logical architecture with dynamic integration properties: A co-worker arranged a meeting with a customer using his brand new UMTS-based PDA with an included visual telephone. He has never used this PDA in his car before. Although the PDA had not been developed in the past, when the car was manufactured, the PDA is recognized by the car and integrated into the system of the vehicle. This happens instantly, when the person enters the car in order to drive to the agreed meeting. The dynamically integrated PDA transmits the address of the customer to the navigation system of the vehicle. The navigation system can now calculate the best route and guide the driver.

The dynamic integration of new IT-components, which were not known at development time of the system, provides the possibility to improve the value of the whole system during the years. Dynamic integration can only be realized based on a shared and open logical architecture and an available and accepted universal integration platform.

3. A FRAMEWORK FOR THE DEVELOPMENT OF DYNAMIC-INTEGRATIVE SOFTWARE SYSTEMS

To assure the dynamic integrability of IT components, these components have to provide general communication and integration capabilities in addition to their specific functions. The integration is automated and thus invisible to the user. The functional and non-functional correctness of the system is guaranteed. Such dynamic integration properties are not considered in current development approaches and lead to the daily visible integration problems [9]. Reasons for this are deficits not of a technological, but of a methodical nature. Today, the specific functions of the developed components are supposed to be most urgent. The potential future environments of the component with its related and necessary integration properties are neglected.

Therefore, we suggest a methodic and scientifically well-founded approach for the development of dynamic-integrative software systems. The applicability in the industrial context will later be demonstrated by a prototypic integration platform with corresponding tool support. By application of case studies and empirical examinations, the feasibility of the approach shall be confirmed. By this, we can acquire a significant contribution to the establishment of a methodical basis for the development of dynamic-integrative software systems.

Using this approach, functional and non-functional correctness of dynamic-integrative software systems is more predictable. Integration properties are designed and incorporated at an early stage to cope with the integration difficulties of future software systems.

The development approach is based on the idea that IT components are dynamically integrated into a correct software system during execution time. For this purpose, a system performance specification is computed based on the specifications of the available components during execution time. Note, the notion performance specification is not restricted to time properties. A performance specification captures all functional and non-functional properties of a dynamic-integrated system. Based on such a performance specification, it can be checked whether the dynamic-integrative system satisfies the functional and non-functional requirements.

3.1 Static Requirements

As shown in Figure 2 initially a requirements specification has to be elaborated. A UML-based, graphical description technique extended by concepts for the specification of non-functional safety requirements may be used. Since safety properties typically consist of a combination of fine-granular functional and non-functional properties, a suitable composition mechanism must be available.

Further extensions have to be provided for the specification of variability and added value. Aiming at an automatic verification, this specification technique has to be formally founded.

Initially, the outlined approach expects that all requirements are fixed during development time and remain unchanged during execution time. This restriction is not a matter of principle and can be released in future research.

3.2 Static Design

Based on the requirement specification, it is possible to design a logical architecture (see S1 in Figure 2) by adaptation of existing component-based design approaches. This architecture consists of a set of logical component specifications and a logical integration specification. As for the requirements, a mathematical system model is needed for the semantic foundation of the applied specification techniques. Especially, a clear distinction between integration architecture and component interfaces is important.

To specify IT components, the existing UML-based interface specification techniques have to be extended and formally founded, also providing support for the precise modeling of non-functional properties of hardware and environment. Aspect-oriented approaches could offer a solution.

For the specification of the integration architecture, a suitable subset of the UML has to be selected, adapted and formally founded. With the resulting specification technique it must be possible to describe the logical architecture with all its possible physical architecture variants.

3.3 Static Verification

Applying the suggested integration approach to this logical architecture, it is already possible to compute a (static) system performance specification during development time (S2). By this, it can be checked during development time if the logical architecture satisfies the fixed requirements (S3, yes/no).

3.4 Dynamic Integration

In dynamic-integrative systems the availability of physical IT components can arbitrarily change during execution time: Integrated components can fail and new components can come along (D1). This means that the logical architecture is not realized by available physical IT components before execution time.

Therefore, a realization relation is dynamically searched which maps the properties of the currently available, physical IT components to the logical components (D2). During the integration step (D3) a physical integration specification is derived from this realization relation and the logical integration specification. Physical component description and physical integration specification together form the physical architecture.

Subsequently, the (dynamic) system performance specification is computed based on the physical architecture (D4) at execution time, too. It is used for the dynamic verification of the correctness of the integrated software system with regard to the requirements and for the evaluation of the integration solution (see below).

The determination of the realization relation between physical and logical system components, the computation of the physical architecture and the system performance specification, as well as

the verification, are automatically performed for each change in the physical constellation of IT components at execution time. Due to this automated processing, the specifications of both realization relation and system performance have to be formal and machine readable. For the computation of the realization relation and the system performance it is also necessary to support well-founded refinement and composition concepts in the formal system model.

3.5 Dynamic Verification

The functional and non-functional correctness of the system as described in the formal performance specification is verified with regard to the requirements at execution time.

The dynamic verification can be regarded as a special case of the static verification. Nevertheless, the dynamic verification method has to efficiently support its application at execution time. This means that the expressiveness of logical and physical component specifications has to be restricted in a sensible way.

In general, a number of different integration solutions can simultaneously satisfy the requirements. In such a case, the system performance specification can additionally be used to compare the matching integration solutions (D5). Based on the added value properties, which are part of the requirements, an objective function could be derived to evaluate and automatically select the best of these integration solutions.

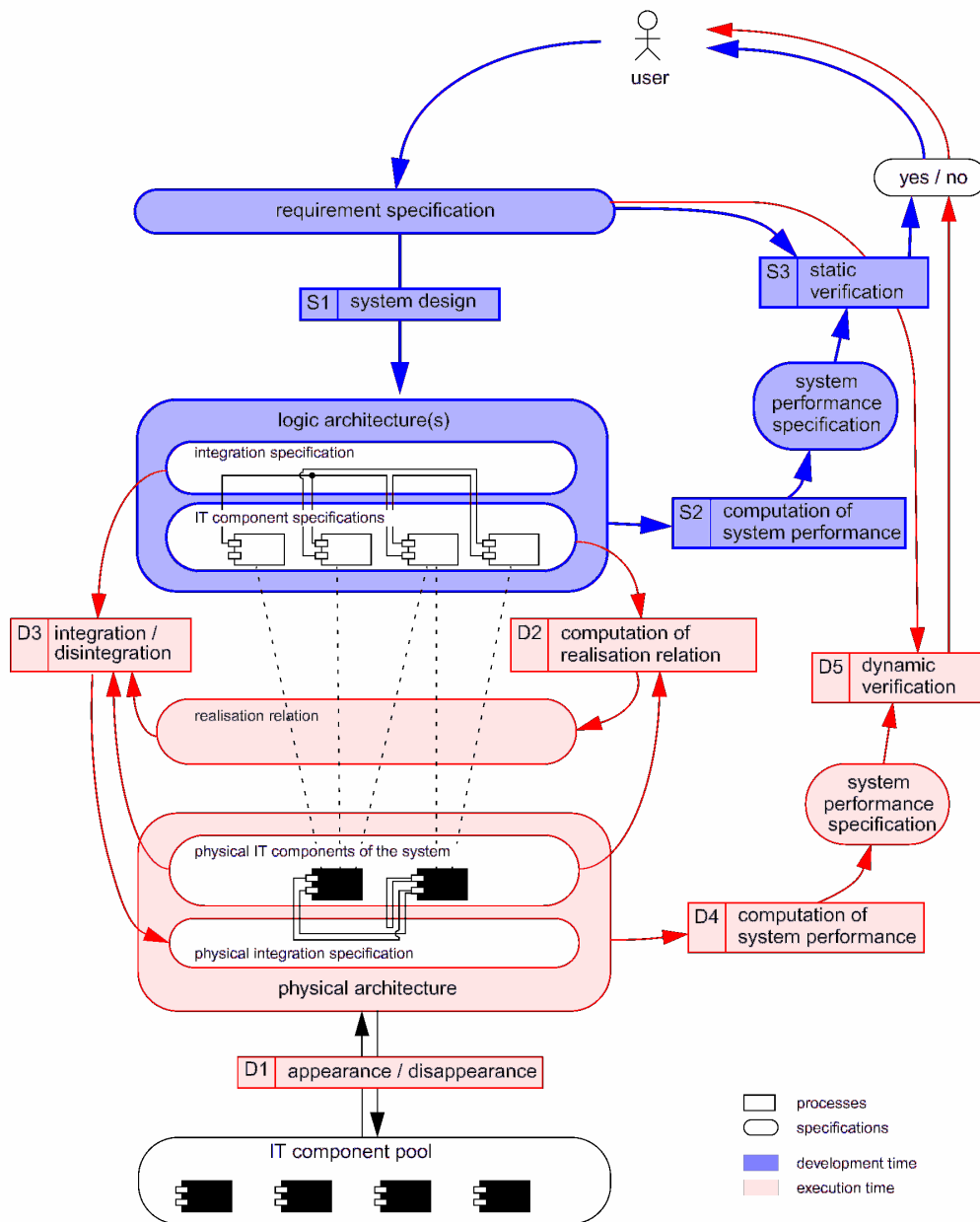


Figure 2. Basic model of the dynamic integration of software-systems

4. FURTHER WORK

In this paper we have motivated and presented a rough sketch of a development approach for dynamic-integrative system. This described specification techniques and formal foundations have to be designed and elaborated. A first step towards the needed specification techniques for the proposed IT-components has already been developed and applied [12].

Based on these experiences the proposed integration and verification techniques can be developed. An algorithm for the composition of the IT-components and a mechanism for verification of the system have to be developed.

Furthermore the presented development approach, the specification techniques under elaboration, and the required integration and verification mechanisms have to be realized with a development and execution platform for dynamic-integrative systems.

Such a platform is providing the central services used by all IT components, such as a look-up-service or an integration service. Note that some basic requirements have to be matched by all components participating in this platform, e.g., compatibility of communication hardware and protocols, or export of the offered functionality in a standardized format. Tool support for the development of dynamic-integrative software systems has also to be provided within the platform, because the involved specifications are automatically processed, e.g., in consistency checks or code generation.

5. REFERENCES

- [1] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, and J. Zettel. Component-Based Product Line Engineering with UML. The Component Software Series. Addison-Wesley, 2001. ISBN 0-201-73791-4. 2001.
- [2] Clemens Szyperski. Component Software: Beyond Object-Oriented Programming. Addison Wesley Publishing Company. 1997.
- [3] Klaus Bergner, Andreas Rausch, Marc Sihling, Alexander Vilbig. Putting the Parts Together – Concepts, Description Techniques, and Development Process for Componentware. Proceedings of the 33th Annual Hawaii International Conference on System Sciences, IEEE Computer Society. 2000.
- [4] Universal Plug and Play Device Architecture Reference Specification. Microsoft Corporation. 2004.
- [5] Salutation Architecture Specification (Part 1), version 2.1. The Salutation Consortium Inc. 1999.
- [6] John Relesh. UPnP, Jini and Salutation - A look at some popular coordination framework for fu-ture network devices. Technical Report, California Software Lab. <http://www.cswl.com/whiteppr/tech/upnp.html>. 1999.
- [7] Chen Harry. Ronin Agent Framework. <http://gentoo.cs.umbc.edu/ronin/>. 2004.
- [8] OMG. CORBA 3.0 – OMG Specification. 2002.
- [9] D. Chakraborty, A. Joshi, Y. Yesha, T. Finin. Service Composition for Mobile Environments. Journal on Mobile Networking and Applications (MONET), Special issue on Mobile Services. 2004.
- [10] Leslie Lamport. Proving the Correctness of Multiprocess Programs. IEEE Transactions on Software Engineering, Vol.3 No.2. 1977.
- [11] Hermann Kopetz, Günther Bauer. The Time-Triggered Architecture. Proceedings of the IEEE, Vol.91 No.1. 2003.
- [12] Dominic Hillenbrand. Spezifikation und Simulation eingebetteter Komponenten. To appear in 2004.