Design for Testability for Highly Reconfigurable Component-Based Systems

Andrea Baldini, Paolo Prinetto

Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24, Torino, Italy
{baldini, prinetto}@polito.it

Giovanni Denaro, Mauro Pezzè

Università degli Studi di Milano Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Via Bicocca degli Arcimboldi 8, Milano, Italy
{denaro, pezze}@disco.unimib.it

Abstract

Highly-reconfigurable component-based systems, i.e., systems that are built form existing components and are distributed in many versions and configurations, are becoming increasingly popular.

The design and verification of such systems presents new challenges. In this paper we propose a design approach that facilitates analysis and testing of different configurations by identifying and tracking relations among requirements, logic components and resources. The approach proposed in the paper allows for easily identifying different dependencies among resources, components and requirements and thus spotting the tests that must be re-executed to assure the desired level of quality.

1 Introduction

Highly-Reconfigurable Component-Based (HRCB) systems are built starting from reusable hardware and software components [5]. In such systems, components can be substituted, added or eliminated to obtain different configu-

[1] This work has been partially founded by the Italian Government in the context of the QUACK project (QUACK: A Platform for the Quality of New Generation Integrated Embedded Systems.)

rations of the same system. The set of systems based on a common subset of core components is often referred to as a *family of systems*.

HRCB systems are increasingly used in many application domains [1]. A notable example of such systems is the board systems of new generation cars, that combine several hardware devices, e.g., DVD, GSM and GPS devices, and provide many complex software services, e.g., Internet facilities, car alarm monitoring and integrated control of all available devices. Some of these components are produced by third parties (for example, the GSM devices are generally not proprietary). Different car models use different configurations. For example, base models may not include GPS and Internet facilities. Configurations are obtained by changing, adding or updating subsets of components.

Design and verification of HRCB systems present new challenges [4,3]. The many different evolving configurations and the often stringent quality requirements cannot be addressed by completely testing each different configuration. New techniques are required to identify subsets of tests to be re-executed to assure the quality of new configurations.

This position paper frames the problem and suggests a preliminary solution for designing such systems that records different dependencies among requirements, logical components and resources to allow for easily identifying tests that must be re-executed when changing a resource, a component, or a requirement. The technique proposed in the paper is presented referring to the design of modern consoles of top-of-the-line cars.

2 Highly Reconfigurable Component-Based Systems

Many commercial systems are often available in many different versions and configurations. We can refer to such classes of systems as system families. A system family includes many different systems that provide a common set of core functionalities, but differ in the number and combination of provided functionalities, and in the accessibility to the offered functionalities.

Often systems that belong to the same family can be obtained from a common set of hardware and software components suitably selected and assembled. The reuse of components can greatly reduce costs and development time, but presents new design and verification challenges.
Main characteristics of HRCB systems are:

- the presence of a common pool of *components*,
- the availability of the components and the systems in many different versions and *configurations* and
- the presence of non-trivial both *functional and non-functional* requirements, e.g., performance, usability and real-time requirements.

Components

We use here a broad definition of component that includes both hardware (physical) and software components. A general definition is not straightforward, so we will rely on the common knowledge to give a meaning to the concept of components, but we will use the term usually at a very high abstraction level. A micro-chip can be a component, but also an entire display; similarly, a particular encoder implementing a specific algorithm can be a component, maybe directly a hardware one or a software component running on a generic processor.

Configurations

We can infer many ways to configure or reconfigure a HRCB system: We can add or eliminate a component in an existing system; We can substitute a particular component with one or more components; We can update components with new versions. For example, we can substitute a micro-chip to gain more processing power, or a lower power dissipation, or new features. We can have different versions of the same product with different displays, e.g., black and white for cheap products and color high-resolution for expensive ones. Even more complex, we can consider the case of an encoder, which we can implement as hardware device (this is faster but usually more expensive) or as software component. According to our definition this is just a substitution of component.

Functional and Non-functional Properties

One of the major challenges for HRCB systems is the presence of non-trivial functional and non-functional requirements. Many of such requirements are expressed in terms of properties for the whole system. This entails that any change in the configuration can impact on any of such properties. Good examples are represented by real-time requirements for system reaction or performance requirements for generic functionalities.

Such properties are not easy to verify at the system level, since they often cannot be simply inferred from the properties of the single components, but depend on several components of the system. Even if all components satisfy a specific requirement, e.g., a real-time constraint, this is not a sufficient condition for the same requirement to be satisfied at the system level. Even if a specific configuration satisfies a specific requirement, e.g., a performance requirement, this is not a sufficient condition for any other configuration to satisfy such requirement.

3 Motivating Example

This section presents a notable case of industrially relevant highly-configurable component-based systems that motivates our research: top-of-the-line automotive applications, such as modern consoles which group many controls of correlated devices.

Such applications have various orders of complexity: the interaction with the driver is guaranteed by a set of input and output devices, such as a display, a keyboard, audio input (microphone) and output; moreover, a set of now common high-level-car components are integrated, such as positioning system with navigation facilities, voice and data communications and vocal commands.

Such systems are reconfigurable and come usually as families of systems, meaning that different specific models are derived from the same general model.

From our direct industrial experience we can affirm that such systems are often produced by specialized companies that work for several different brands. Each client can possibly request different versions of its specific application, but the same project can cover more than one client, so that different configurations are sold under different brand names.

It is evident that such systems have all the features that characterize HRCB systems: they present a common pool of core components, they are highly reconfigurable and have many families of different possible configurations, and they certainly require non-trivial functional and non-functional properties, e.g., performance requirements.

There are several commercially available car consoles. The model described here conforms with industrial products of various manufacturers for the near future.

The list of all the possible functionalities is out of the scope of this paper, but we can present a set of representative functions usually performed by such classes of systems. From the informal requirements for a full-optional application, we can extract functions such as global positioning system (GPS) navigation facilities, radio and CD player, on-board computer, upgrade and maintenance facilities, safety mechanisms, TV and DVD player, integrated mobile phone, Internet access and browsing and presence of various technologies such as bluetooth for wireless connections.

The system includes several functionalities: GPS navigation based on road maps CD; radio and the CD player; on-board computer displaying planned arrival time, average consumption, maximum distance that can be covered with the remaining fuel; functions for upgrading and maintaining the software of the console by authorized people connected by wire or GPRS/UMTS; Safety signals; TV with Teletext and DVD players; GSM mobile phone; Internet access; Bluetooth wireless connectivity to mobile phones, PDA and notebook computers.

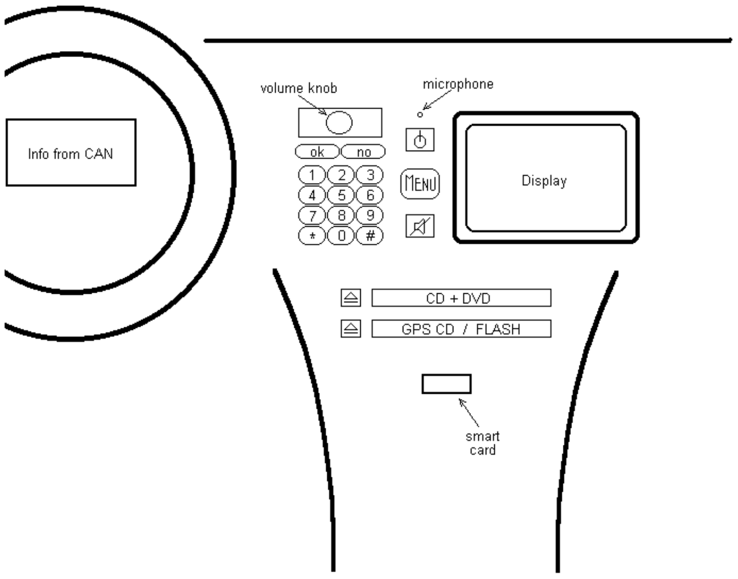


Fig. 1. Schematic external appearance

Figure 1 shows a conceptual schema of the interface as seen from the driver.

Readers interested in further details about this industrial case are referred to [1].

4 Approaching the Design of HRCB Systems

Traditional design and verification techniques are not tailored to the specific characteristics of HRCB systems and thus their use may result in excessive costs and unsatisfactory results.

This section outlines an approach for the design of HRCB systems. The proposed approach augments a classic software development approach, e.g., UML based, by adding design for testability functionalities.

According to IEEE, testability can be defined as: *(1) the degree to which a system or component facilitates the establishment of test criteria and performance of tests to determine whether those criteria have been met. (2) the degree to which a requirement is stated in terms that permit establishment of test criteria and the performance of tests to determine whether those criteria have been met* [2].

The process proposed here identifies different design steps particularly relevant for the design of HRCB systems, namely requirements analysis, identification of logic components, and resource allocation. The process defines a set of relations among different design levels given in the form of mappings that define the correspondence among levels. The mappings among levels facilitate the identification of functionalities to be tested and the identification of features shared between different system configurations that require additional testing.

The *requirement analysis* defines the requirements for the system. The resulting requirement specifications may be given in many different ways, e.g., by means of UML use-case diagrams. In this paper, we assume that the requirements are expressed as a list of items that may take the form of a numbered list.

The *logic components* are the system counterparts of a requirement [2], i.e., while a requirement describes a functionality as perceived by the drivers, a logic component represents the subsystem that implements such functionality. For example, the requirement *assisted navigation* is implemented by the logic component *satellite based navigation system.* Logic components can be associated with sets of constraints. In general, constraints are used to capture non-functional requirements. For example, for the *satellite based navigation system*, we may require real-time constraints or compliance with existing standards, such as UMTS standard.

Resources indicate entities at various abstraction levels, e.g., hardware devices, software modules, protocols, and standards. Logic components are mapped to resources, according to the functionalities to be satisfied and the constraints on logic components. Some logic constraints may involve several resources. For example timing requirements require the cooperation of different resources.

The mapping among requirements, logic components and resources is organized as a dependency tree.

The design process described above can be applied at all development stages, e.g., preliminary analysis of consistency of a given configuration. In this case, we consider all requirements that frame the configuration and merge the related dependency trees. This provides a global sketch of the system resources and their dependencies.

The presence of different levels (requirements, logic components, resources) and the mappings between them allow the identification of the elements to test, and help in the definition of quality figures. For example, mappings are useful to understand what is modified when a change in a configuration occurs, both in terms of elements and in terms of integration and relationships among them, and this allows for identifying test cases for regression testing.

5 Preliminary Application of the Technique

In this section we provide an early example of application of the methodology introduced in the former section referring to the preliminary analysis of the car console system presented in Section 3.

Figure 2 presents a subset of requirements for the car control system, given

[2] Logic component are elsewhere referred to as *features*, for example in the domain of telecommunication applications.

(i) GPS navigation for assisted navigation: it avoids drivers consulting road maps and searching for road signs, indicating directions to every destination stored in the proper road maps CD.

(ii) Audio and video reproduction: Radio and the CD player offer hi-fi quality sound.

(iii) On-board computer management: the on-board computer acquires data useful for the driver, from the possible arrival time and the average consumption to the maximum distance that can be covered with the remaining fuel.

(iv) System maintenance operations: authorized personnel can upgrade and maintain the software of the console, connecting a specific wire or directly via GPRS/UMTS connection.

(v) Car alarm monitoring: acoustic safety signals to remind the driver not to exceed the speed limit.

(vi) TV with Teletext and DVD players: the driver can always obtain actual news and entertainment for the passengers.

(vii) Handling of vocal communication: the GSM mobile phone offers full phone capabilities, complete with SMS messaging and phonebook.

(viii) Internet navigation facilities: Internet access allows additional information and connection to maintenance and emergency services.

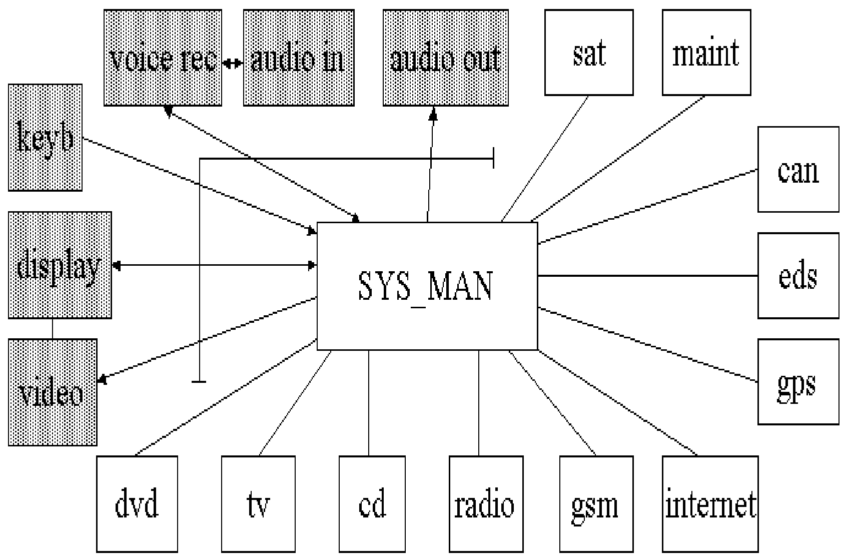
(ix) Wireless access to external devices: the Bluetooth wireless technology allows the driver to make effortless, wireless and instant connections to various communication devices, such as mobile phones, PDA and notebook computers.

Fig. 2. Few requirements for the car console system of Section 3

as a numbered list. This list conforms with the informal requirements already presented in Section 3. Numbers are used to define the mapping to the logic components.

Figure 3 shows a subset of logic components for the car console system. The diagram presents a short-hand notation for the logic components: sat is the satellite based navigation system, internet is the Internet browser and so on. Shadowed components represent logic subsystems for which we expect more interactions with the users. Some non-trivial constraints are indicated as example: the satellite based navigation system must be UMTS compliant to download additional information and maps, and the global positioning system (GPS) services, used by the satellite based navigation system, must give the driver a resolution on the position of 3 meters, meaning that the driver should perceive a 3 mt precision, independently from the specific technology.

Figure 4 shows the dependency trees for some of the requirements. Indentation indicates association between requirements, logic components and resources or dependence among resources. For instance, the first requirement,



```
list of constraints:
- sat -> UMTS compliance
- gps -> perceived resolution: 3 meters
- ...
```

Fig. 3. A subset of logic components and constraints for a car console system

assisted navigation, is implemented with the logic component *satellite based navigation system*, which *must be* UMTS compliant. Implementation is based on a GUI software component built on top of a renderer, which manages the physical display. Communication takes place via sockets using UMTS as protocol. A radiomobile antenna is required as physical resource for communication. The resources indicated in the trees are taken from a list not shown in this paper. The whole dependency tree is actually a forest of dependency trees, one for each initial requirement of the specific configuration.

The dependency trees represent the relationships between different levels and allow the identification of the tests that are necessary for each different version. If we modify a specific resource in a configuration, the dependency trees tell us which logic components and which requirements are affected by such change, and must be re-tested. For example, referring to Figure 4, if we modify the resource UMTS (e.g., adopting a new library that implements the UMTS services), the dependency trees tell us that only the satellite navigation system (SAT) and the Internet browser need regression testing. Conversely, the integrated phone subsystem is not affected by such modification. Furthermore, for the SAT subsystem, we need to re-test only the communication infrastructure (socket-based), while the GUI, the renderer and the display remain unaffected.

For resources, the issue is a bit more complex. When we want to integrate

(i) assisted navigation (requirement)
- satellite based navigation system - sat (logic component)
- UMTS compliance (constraint on the logic component)
  - · navigation GUI (resource)
    - renderer (resource)
      - display (resource)
  - · socket (resource)
    - UMTS (resource)
      - radiomobile antenna (resource)

(ii) handling of vocal communication (requirement)
- integrated phone - GSM (logic component)
  - · vocal interface (resource)
    - microphone (resource)
    - speaker (resource)
  - · GSM (resource)
    - radiomobile antenna (resource)

(iii) Internet navigation facilities (requirement)
- Internet browser - internet (logic component)
  - · Internet browser software component (resource)
    - socket (resource)
      - UMTS (resource)
        - radiomobile antenna (resource)

Fig. 4. Some dependency trees for the car console system

different components, we obtain many cases of presence of the same resource in different logic components. However we cannot merge directly the subtrees related to that particular resource, but we must first distinguish among different situations.

We have classified such situations as:

Sharing: the same resource is used by different logic components, but there is no direct dependency so the resource can be used by more than one component at the same time.

Dependency: the resource can be shared but its sharing impacts on the involved components, i.e., constraints are not violated but the behavior of one component can affect the behavior of the others sharing the same resource.

Conflict: the sharing of a resource represents an unacceptable dependency among components, i.e., some constraints are violated. A conflict must be solved as a design issue, e.g., adding resources.

Note that the expression *dependency* used above is not related to the expression *dependency tree*, even if from the merging of dependency trees such situations can be highlighted.

Sharing indicates that a change in that particular shared resource can affect the components sharing the resource, while dependency indicates that a change in any resource of a particular component could affect the components depending from the same resource. In the latter case, the changes in the behavior could cause the violation of a constraint of any of the sharing components, and the dependency could become a conflict.

The early individuation of such new conflicts in different configurations is possible only with a structure that contains information about resource mappings, such as dependency trees.

6 Conclusions

HRCB systems, i.e., systems distributed in different versions and configurations made of different subsets of components, are becoming increasingly used. This type of systems entail many new verification requirements that cannot be easily addressed by means of traditional design and testing techniques.

This paper proposes a design methodology to map requirements to logical components and resources, to increase testability of HRCB systems. The ideas presented in the paper are explained on a sample application in the field of the automotive applications, where the HRCB technology is rapidly spreading.

We are currently developing and experimenting the proposed technique in the context of the QUACK (A Platform for the Quality of New Generation Integrated Embedded Systems) project[3].

References

[1] Baldini, A., A. Benso, P. Prinetto, S. Mo and A. Taddei, *A uml process for system level functional test: an industrial perspective*, in: *Proceedings of the Sixth Biennial World Conference on Integrated Design and Process Technology (IDPT'02)* (2002), p. 48.

[2] *IEEE Standard Glossary of Software Engineering Technology ANSI/IEEE 610.12*, IEEE Press (1990).

[3] Liu, C. and D. Richardson, *Software components with retrospectors*, in: *Proceedings of the International Workshop on the Role of Software Architecture in Testing and Analysis (ROSATEA)*, 1998.

[4] Rosenblum, D., *Challenges in exploiting architectural models for software testing*, in: *Proceedings of the International Workshop on the Role of Software Architecture in Testing and Analysis (ROSATEA)*, 1998.

[5] Szyperski, C., "Component Software: Beyond Object-Oriented Programming," ACM Press and Addison-Wesley, New York, NY, 1998.

[3] QUACK is partially supported by MIUR, the Italian Ministry of University and Education through the Cofin Program