Contracts for System Design

Albert Benveniste
Benoît Caillaud
Dejan Nickovic
Roberto Passerone
Jean-Baptiste Raclet
Philipp Reinkemeier
Alberto Sangiovanni-Vincentelli
Werner Damm
Thomas A. Henzinger
Kim G. Larsen
Editorial Scope

Topics

Foundations and Trends® in Electronic Design Automation publishes survey and tutorial articles in the following topics:

- System level design
- Behavioral synthesis
- Logic design
- Verification
- Test
- Physical design
- Circuit level design
- Reconfigurable systems
- Analog design
- Embedded software and parallel programming
- Multicore, GPU, FPGA, and heterogeneous systems
- Distributed, networked embedded systems
- Real-time and cyberphysical systems

Information for Librarians

Foundations and Trends® in Electronic Design Automation, 2018, Volume 12, 4 issues. ISSN paper version 1551-3939. ISSN online version 1551-3947. Also available as a combined paper and online subscription.
Contracts for System Design

Albert Benveniste
INRIA, Rennes, France
albert.benveniste@inria.fr

Benoît Caillaud
INRIA, Rennes, France
benoit.caillaud@inria.fr

Dejan Nickovic
Austrian Institute of Technology
Dejan.Nickovic@ait.ac.at

Roberto Passerone
University of Trento, Italy
roberto.passerone@unitn.it

Jean-Baptiste Raclet
IRIT, Toulouse, France
raclet@irit.fr

Philipp Reinkemeier
Offis, Oldenburg, Germany
Philipp.Reinkemeier@offis.de

Alberto Sangiovanni-Vincentelli
University of California at Berkeley
alberto@eecs.berkeley.edu

Werner Damm
Offis and University of Oldenburg, Germany
werner.damm@offis.de

Thomas A. Henzinger
IST Austria, Klosterneuburg
tah@ist.ac.at

Kim G. Larsen
Aalborg University, Denmark
kgl@cs.aau.dk
# Contents

1 Introduction 2
   1.1 Industrial context ........................................... 2
   1.2 Positive impact of contract-based design ................. 4
   1.3 A bird’s eye view of research in contracts ............ 7
   1.4 Contribution of this monograph ........................... 8

   2.1 Contract based design ....................................... 14
   2.2 A primer on contracts ....................................... 18

3 Positioning of this Monograph and Bibliographical Note 28
   3.1 Contracts in software engineering ........................... 28
   3.2 Contracts for (possibly cyber-physical) systems ......... 34

4 A Mathematical Meta-Theory of Contracts 37
   4.1 Components .................................................. 38
   4.2 Contracts .................................................... 40
   4.3 Refinement and conjunction ................................. 41
   4.4 Parallel composition ....................................... 43
   4.5 Quotient ..................................................... 49
   4.6 Making contract composition associative ................. 49
   4.7 Abstractions ................................................ 50
### 5 Assume/Guarantee Contracts

5.1 Synchronous A/G contracts with fixed alphabet .......................... 60
5.2 Dealing with variable alphabets ............................................. 66
5.3 Abstractions ................................................................. 67
5.4 Observers ................................................................. 68
5.5 Asynchronous dataflow A/G contracts ................................. 72
5.6 A/G contracts for Cyber-Physical systems ................... 73
5.7 Discussion ................................................................. 74
5.8 Bibliographical note ....................................................... 75

### 6 Synchronous Moore Interfaces and A/G Contracts

6.1 Introduction ................................................................. 81
6.2 An illustration example for Moore Interfaces .................... 82
6.3 A/G contract saturation via Moore Interfaces ................. 85
6.4 Moore Interfaces, seen as A/G contracts ......................... 89
6.5 Discussion ................................................................. 92

### 7 Rely/Guarantee Reasoning and A/G Contracts

7.1 A brief on rely/guarantee reasoning ................................. 94
7.2 Components for shared variable concurrency ................. 96
7.3 Contracts for shared variable concurrency ................... 101
7.4 Discussion ................................................................. 104

### 8 Interface Theories

8.1 Components as i/o-automata ............................................ 106
8.2 Interface Automata with fixed alphabet ......................... 108
8.3 Modal Interfaces with fixed alphabet .............................. 119
8.4 The approach by Gerald Lütgen, Walter Vogler et al. .... 145
8.5 Modal Interfaces with variable alphabet ....................... 147
8.6 Decomposing a contract as a composition of subcontracts 149
8.7 Modal interfaces as Assume / Guarantee contracts .......... 153
8.8 Bibliographical note ....................................................... 160

### 9 Scheduling Contracts

9.1 Introduction ................................................................. 169
Abstract

Recently, contract-based design has been proposed as an “orthogonal” approach that complements system design methodologies proposed so far to cope with the complexity of system design. Contract-based design provides a rigorous scaffolding for verification, analysis, abstraction/refinement, and even synthesis. A number of results have been obtained in this domain but a unified treatment of the topic that can help put contract-based design in perspective was missing. This monograph intends to provide such a treatment where contracts are precisely defined and characterized so that they can be used in design methodologies with no ambiguity. In particular, this monograph identifies the essence of complex system design using contracts through a mathematical “meta-theory”, where all the properties of the methodology are derived from a very abstract and generic notion of contract. We show that the meta-theory provides deep and illuminating links with existing contract and interface theories, as well as guidelines for designing new theories. Our study encompasses contracts for both software and systems, with emphasis on the latter. We illustrate the use of contracts with two examples: requirement engineering for a parking garage management, and the development of contracts for timing and scheduling in the context of the AUTOSAR methodology in use in the automotive sector.
1

Introduction

1.1 Industrial context

System companies such as automotive, avionics and consumer electronics enterprises are facing significant difficulties due to the exponentially raising complexity of their products coupled with increasingly tight demands on functionality, correctness, and time-to-market. The cost of being late to market or of imperfections in the products is staggering as witnessed by the recent recalls and delivery delays that system industries had to bear. Many challenges face the system community to deliver products that are reliable and effective. Table 1.1, albeit not recent, continues to be a telling example of the main causes and their share in the difficulties related to systems complexity.¹ This table highlights the importance of system integration, where corrections occur late in the design flow and are therefore very costly.

System specification and integration is particularly critical for Original Equipment Manufacturers (OEM) managing the integration and maintenance process with subsystems that come from different suppliers who use different design methods, different software architectures, and different hardware.

### 1.1. Industrial context

Table 1.1: Difficulties related to system complexity. The table displays, for each industrial sector, the percentage of tasks delayed and tasks causing delays, for the different phases of system design.

<table>
<thead>
<tr>
<th>Design task</th>
<th>Tasks delayed automotive</th>
<th>Tasks delayed automation</th>
<th>Tasks delayed medical</th>
</tr>
</thead>
<tbody>
<tr>
<td>System integration test, and verification</td>
<td>63.0%</td>
<td>56.5%</td>
<td>66.7%</td>
</tr>
<tr>
<td>System architecture design and specification</td>
<td>29.6%</td>
<td>26.1%</td>
<td>33.3%</td>
</tr>
<tr>
<td>Software application and/or middleware development and test</td>
<td>44.4%</td>
<td>30.4%</td>
<td>75.0%</td>
</tr>
<tr>
<td>Project management and planning</td>
<td>37.0%</td>
<td>28.3%</td>
<td>16.7%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Design task</th>
<th>Tasks causing delay automotive</th>
<th>Tasks causing delay automation</th>
<th>Tasks causing delay medical</th>
</tr>
</thead>
<tbody>
<tr>
<td>System integration test, and verification</td>
<td>42.3%</td>
<td>19.0%</td>
<td>37.5%</td>
</tr>
<tr>
<td>System architecture design and specification</td>
<td>38.5%</td>
<td>42.9%</td>
<td>31.3%</td>
</tr>
<tr>
<td>Software application and/or middleware development and test</td>
<td>26.9%</td>
<td>31.0%</td>
<td>25.0%</td>
</tr>
<tr>
<td>Project management and planning</td>
<td>53.8%</td>
<td>38.1%</td>
<td>37.5%</td>
</tr>
</tbody>
</table>


Platforms. In addition, even inside an OEM itself, complex systems involve a number of different aspects or viewpoints that are generally handled by
different teams using different paradigms and tools. Examples of aspects are system architecture, the functions the system should perform and the services it should deliver, its safety and reliability characteristics, its energy budget, its deployment on an embedded computing platform to name a few.

Contract-based design has as main goal solving the above issues in a rigorous framework.

1.2 Positive impact of contract-based design

Addressing the Complexity of Systems. Several approaches have been developed by research institutions and industry to cope with the exponential growth in systems complexity. Of particular interest to the development of embedded controllers and systems are layered design and component-based design (used, e.g., in the AUTOSAR\(^2\) standard in the automotive sector, and the ARINC\(^3\) standard in the avionic domain), model-based development (supported by important frameworks and tools such as SysML\(^4\) [208] and/or AADL [211] for architecture modeling, and Modelica [133] and Matlab-Simulink [168] for system modeling), virtual integration (Ptolemy [124] and Metropolis [100, 68]), and platform-based design [100, 114, 221]. There are two basic principles followed by these methods: abstraction/refinement and composition/decomposition. Abstraction and refinement are processes that relate to the flow of design between different layers of abstraction (vertical process) while composition and decomposition operate at the same level of abstraction (horizontal process). Layered design and model-based development focus on the vertical process while component-based design deals principally with the horizontal process. Platform-based design combines the two aspects in a unified framework.

While the above methods have been critical steps in breaking systems complexity, they do not by themselves provide the ultimate answer. Contracts are ideal tools to solidify both vertical and horizontal processes providing the theoretical background to support formal methods in system design. When design is being performed at a considered layer, implicit — and often hidden — assumptions regarding other layers (e.g., computing resources)

\(^2\)http://www.autosar.org/
\(^3\)https://www.aviation-ia.com/product-categories/arinc
\(^4\)http://www.omg.org/spec/SysML/
1.2. Positive impact of contract-based design

are typically invoked by the designer. Actual properties of these other layers, however, cannot be compared against these hidden assumptions. Similarly, when components or sub-systems are abstracted via their interfaces in component based design, it is generally not true that such interfaces provide sufficient information for other components to be safely implemented based on this sole interface. By pinpointing responsibilities and making hidden assumptions explicit, contract-based design provides the due discipline, concepts, and techniques to cope with this.

Another challenge for component-based design of embedded systems is to provide interface specifications that address behaviors, not only type properties of interfaces, and are rich enough to cover all phases of the design cycle. This calls for including non-functional characteristics as part of the component interface specifications, which is best achieved by using multiple viewpoints [40, 46, 42]. Contract-based design supports multiple viewpoints by giving a mathematically precise answer to what it means to fuse them.

**Addressing OEM-Supplier Chains and Managing Requirements.**
The management of responsibilities in and design processes across OEM-supplier chains is indeed the core target of contract-based design. By making the explication of implicit assumptions mandatory, contracts help assign responsibilities to a precise stake holder for each design entity. By supporting independent development of the different sub-systems while guaranteeing smooth system integration, they orthogonalize the development of complex systems. Contracts are thus adequate candidates for a technical counterpart of the legal bindings between partners involved in the distributed and concurrent development of a system.

Regarding requirement capture, efforts have been made by paying close attention to book-keeping activities such as the management of the requirement descriptions and corresponding traceability support (e.g., using commercial tools such as Doors⁵ in combination with Reqtify⁶) and by inserting, whenever possible, precise formulation and analysis methods and tools. Still, the need for basing requirement engineering on more solid bases is widely

---

⁶https://www.3ds.com/fr/produits-et-services/catia/produits/reqtify/
Introduction

acknowledged. Specifications used for procurement should be precise, unambiguous, and complete. Indeed, a recurrent reason for failures causing deep iterations across supply chain boundaries rests in incomplete characterizations of the conditions for use and environment of the system to be developed by the supplier, such as missing information about failure modes and failure rates, missing information on possible sources of interference through shared resources, and missing boundary conditions. This argument highlights the need of making assumptions on the design context explicit in OEM-supplier commercial contracts. The potentially highest value proposition of a systematic introduction of contracts indeed lies in requirement capture. Already the evaluation results of the industrial partners in the Integrated Project Speeds7 acclaim the use of contracts for the requirement capture phase to substantially increase the quality of requirements.

By systematically enforcing the explication of assumptions, systems understanding and thus system interface specifications are substantially improved. Thinking in terms of assumptions uncovers early potential incompatibilities, which otherwise would have only been found much later in integration stages. Furthermore, (i) the explication of assumptions significantly eases concurrent engineering; assumptions provide a natural way of communication between design teams; (ii) the quality improvements in requirements translates directly to improvement of test cases for requirement-based testing; and (iii) the effort spent in explicating assumptions translates directly to improvement of test cases for integration testing. Assumptions are easily integrated into industrial design flows for requirement capture, including tools for traceability and change management. Further, formalized contracts allow for a rigorous checking of otherwise easily overlooked inconsistencies between requirements. Formalized contracts allow for “playing out” contracts — a term coined by David Harel [149, 147] — i.e., executing formalized specifications by engines that systematically generate all behaviours possible under the current set of contracts. Such simulation based environments give strong support for checking the completeness of requirements. Finally, vectors for requirement based testing and virtual integration testing can be automatically derived from formalized contracts, again leading to a significant quality improvement. Observers can be automatically generated from

7http://cordis.europa.eu/project/rcn/79466_en.htm
formalized contracts and used in model-, software- and hardware-in-the-loop testing, or even integrated into execution platforms e.g. to diagnose failure situations. We will thus in this monograph elaborate in particular on the benefits of formalized contracts for requirement capture.

1.3 A bird’s eye view of research in contracts

The notion of contract is not new. It was first developed and promoted in the community of software engineering, and more specifically Model Driven Engineering. Actually, Design by Contract is a software engineering technique popularized by Bertrand Meyer [200, 201] following earlier ideas from Floyd-Hoare logic [234, 155]. Floyd-Hoare logic assigns meaning to sequential imperative programs in the form of triples of assertions consisting of a precondition on program states and inputs, a command, and a postcondition on program states and outputs. So far contracts consisting of pre/postconditions naturally fit imperative sequential programming. In situations where programs may operate concurrently, interference on shared variables can occur. Rely/Guarantee rules [159] were thus added to interface contracts. Rely conditions state assumptions about any interference on shared variables during the execution of operations by the system’s environment. Guarantee conditions state obligations of the operation regarding shared variables.

Despite early contributions by Abadi, Lamport, and Wolper [5, 3], developing contracts for Cyber-Physical Systems [236, 100] and Reactive Systems [146, 152, 142, 196], where mathematical behaviors are essential, boomed more recently in the 2000’s, when de Alfaro and Henzinger proposed and popularized so-called interface theories [105, 103, 8]. Since then, a number of models have been proposed that can be seen as instances of contract theories, either to address a specific technical aspect (e.g., function, timing, and resources), or by following different styles and approaches (Assume/Guarantee contracts or Interfaces).  

---

8Distributed physical systems complemented by computing systems.
9See the dedicated bibliographical notes in this monograph.
1.4 Contribution of this monograph

This wide diversity in the proposed approaches calls for a clarification of what the essence of a contract theory is. More specifically, we need an abstract and generic theory (a \textit{meta-theory}) of contracts or interfaces that abstracts away how contracts and actual designs are actually represented and still formally defines the following concepts:

\begin{itemize}
  \item \textit{Implementations} and \textit{environments} that conform to the contract; a contract is \textit{consistent} if it possesses legal implementations and \textit{compatible} if it possesses legal environments;
  \item contract \textit{refinement}, the proper notion of substitutability for contracts;
  \item \textit{Conjunction} of contracts, how to “fuse” different viewpoints;
  \item \textit{Parallel composition} of contracts, how composing (sub-)contracts attached to subsystems yields a system-level contract; the aim is that this parallel composition supports independent development, meaning in particular that composing legal implementations for each subcontract yields a legal implementation for the system-level contract;
  \item an additional, less essential but still useful concept, is that of \textit{quotient}, which is the adjoint of the notion of parallel composition; how to “patch” an existing design to make it satisfy a new contract.
\end{itemize}

As the central contribution of this monograph, we thus propose a mathematical \textit{meta-theory of contracts} and specialize it to different existing contract theories and variations thereof. In addition to presenting a number of new results, the monograph has a tutorial value in explaining the role of contracts and interfaces in design. In this respect, we include extensive bibliographical notes with particular attention to the numerous results published since year 2000. Since a number of topics are addressed, we preferred to defer bibliographical studies to the different chapters for each different topic.

The monograph is organized as shown in the Figure 1.1, which shows a dependency map between the different chapters. Chapters 1, 2, and the concluding Chapter 12 address readers who may not be specialists in contracts nor on formal methods (except for the summary of results when describing
1.4. **Contribution of this monograph**

the organization of the paper). Chapter 3 is a wide scope discussion of the state of the art—details of recent results are not discussed. Chapter 4, which is a key contribution of this monograph, is more technical but is meant to be self-contained and should be readable by anyone having general skills in mathematics. Chapters 5 to 9 target readers enough exercised in formal methods and, for some parts, even researchers in the field. Some readers may be particularly interested in a particular contract framework and then concentrate on the corresponding chapter. Alternatively, she may be interested in links between frameworks. The two application Chapters 10 and 11 target a wider audience, although they rely on the technical material of previous chapters. A more detailed description of these chapters follows.

![Figure 1.1: Organization of the monograph and dependencies between chapters.](image-url)
Introduction

In Chapter 2 we first discuss the requirements on a theory of contracts, based on methodological considerations. In particular we stress the need to support different viewpoints on the system (e.g., operation, function, timing, energy, safety). Then we develop a primer on contracts by using a very simple example requiring only elementary mathematical background. The purpose of this simplistic example is to smoothly and informally introduce the different concepts and operations we need for a contract framework.

Chapter 3 presents a birds eye bibliography of the subject and explains the positioning of our work. So far the links and parallels between the two notions of contract in Object Oriented programming and contract or interface for system design were obscure. In this chapter we draw these two landscapes and pave the way for clarifying the (actually existing) links between these two notions of contract.

Chapter 4 is the cornerstone of this monograph: It presents a new vista on contracts. The so-called “meta-theory” of contracts is introduced and developed in detail. By meta-theory we mean the collection of concepts, operations, and properties that any formal contract framework should offer. Every concrete framework compliant with this meta-theory will inherit these generic properties. The principle of the meta-theory is the definition of a contract as two sets: correct implementations and legal environments. In doing so, we do not assume any particular way of specifying implementations or environments thus making the meta-theory applicable to any contract theory proposed in the literature. Architecture design is greatly facilitated if the framework used allows to re-structure in a different way a system architecture, while preserving its overall semantics (i.e., meaning). A mathematical formalization of this feature is by requiring that the composition operator supporting architecture modeling shall be associative: \( (M \times M') \times M'' = M \times (M' \times M'') \), illustrated in Figure 1.2. When applied to contracts, the same property is key in supporting independent development of subsystems by different suppliers with safe system integration. The meta-theory naturally leads, instead, to the consideration of a weaker notion of sub-associativity, involving the refinement for its definition — we prove that sub-associativity is sufficient for supporting independent development. Not all concrete contract frameworks possess an associative parallel composition; our results prove that sub-associativity nevertheless holds. We give a tight additional axiom for the
1.4. Contribution of this monograph

Figure 1.2: Illustrating associativity in architectures. The figures show two architectures using the same set of five components. Components are grouped into different subsystems in the top and bottom architectures. Associativity means that the two obtained architectures should possess identical semantics.

meta-theory ensuring that contract composition is associative and we show that this axiom holds for Assume/Guarantee contracts. We introduce the notion of quotient, which supports the practice of patching an existing system to make it satisfy different specifications; the quotient formalizes the concept of “minimal patch”. We finally show how abstraction techniques can be defined at the level of the meta-theory, thus specializing to any compliant contract theory. The meta-theory does not specify how components and contracts are effectively represented and manipulated. The subsequent series of chapters presents a panorama of major concrete contract frameworks.

Chapter 5 deals with Assume/Guarantee contracts [40, 46]. This framework is the most straightforward instance of the meta-theory. It presents pairs \((A, G)\) of assumptions and guarantees explicitly, \(A\) and \(G\) being both expressed as properties. This framework is flexible in that it allows for different styles of description of such properties — computational efficiency depends on the style adopted. In Chapter 6 we relate the Synchronous Interfaces [82]
to the Assume/Guarantee contracts. In Chapter 7 we analyse Rely/Guarantee reasoning [159] used in the area of software engineering and formal methods, to reason about concurrency. We show that this reasoning is also tightly related to Assume/Guarantee contracts. Chapter 8 develops the Interface theories [105, 19], in which assumptions and guarantees are specified by means of a single object: the interface. We revisit the notion of quotient for Modal Interfaces [226, 227], to make it the proper specialization of the notion of quotient following the meta-theory. We use this revisited quotient in a holistic methodology for automatically moving from system-level requirements to a set of subcontracts for the different suppliers. We ground on firm bases how Assume/Guarantee contracts can be emulated using Modal Interfaces. Chapter 9 develops a contract framework addressing schedulability analysis, a task involving resource aspects. This framework is subtle because the time and the computing resources both have a strong global flavor.

We complement the above chapters devoted to aspects of the theory with two illustration cases. In Chapter 10, we develop and study requirements for a simple parking garage. Its top-level specification comprises several viewpoints, each one consisting of a requirement table. We pay attention to responsibilities by properly identifying assumptions regarding the environment (context of use), and guarantees offered by the system if properly used. We then study the critical design step consisting in producing sub-contracts for each supplier, following an architecture of sub-systems that differs from the top-level architecture — a frequently encountered situation. We go beyond the state-of-the-art by proposing a synthesis method and algorithm, by which the sub-contracts are automatically derived, from the top-level contract and the (SysML-like) topological description of the sub-systems architecture. We discuss the use of contracts in formally establishing properties of the requirements such as consistency, compatibility, and completeness. Despite this being a simple example, it is yet much too complex to be dealt with by hand. A Proof of Concept tool was used to support our development. The contract framework used for this study is the Modal Interfaces.

Chapter 11, which is intended to present an industrially-relevant application, addresses a key part of the AUTOSAR development process in use in the automotive industry. AUTOSAR advocates a design methodology by which the functions, structured into tasks, are first designed independently of the
1.4. *Contribution of this monograph*

computing and communication infrastructure, assuming a virtual AUTOSAR run time environment. We study the key step by which time budgets are then allocated to tasks and computing resources are assigned. Lack of formal support in AUTOSAR methodology makes this step difficult today. We show the benefit of using contracts for this step. To this end, we develop an adaptation, called *scheduling contracts*, of the Assume/Guarantee contracts.

Finally the concluding chapter summarizes the lessons drawn from this work and analyzes the industrial situation.
References


References


References


References


Full text available at: http://dx.doi.org/10.1561/1000000053
References


Full text available at: http://dx.doi.org/10.1561/1000000053


References


Full text available at: http://dx.doi.org/10.1561/1000000053


References


Full text available at: http://dx.doi.org/10.1561/1000000053
References


References


References


References


References


References


Full text available at: [http://dx.doi.org/10.1561/1000000053](http://dx.doi.org/10.1561/1000000053)
References


References


References


References


