Embedded Systems Development:
From Functional Models to Implementations

Alberto Sangiovanni-Vincentelli, Haibo Zeng, Marco Di Natale
and Peter Marwedel

Alberto Sangiovanni-Vincentelli
UC Berkeley, Berkeley, USA, e-mail: alberto@eecs.berkeley.edu

Haibo Zeng
McGill University, Montreal, Canada, e-mail: haibo.zeng@mcgill.ca

Marco Di Natale
Scuola Superiore Sant’Anna, Pisa, Italy, e-mail: marco.dinatale@sssup.it

Peter Marwedel
Technische Universität Dortmund, Dortmund, Germany, e-mail: peter.marwedel@tu-dortmund.de
Contents

Embedded Systems Development: From Functional Models to Implementations ........................................ v
Alberto Sangiovanni-Vincentelli, Haibo Zeng, Marco Di Natale and Peter Marwedel
Preface ............................................................................................................................................. xiii

1 Introduction: Modeling, Analysis and Synthesis of Embedded Software and Systems ................................................. 1
Alberto Sangiovanni-Vincentelli, Haibo Zeng, Marco Di Natale and Peter Marwedel
1.1 Recommended Reading ................................................. 4
1.2 Model-Based Design and Synthesis ................................................. 7
1.3 Model-Driven Design, Integration and Verification of Heterogeneous Models ................................................. 8
1.4 Component-Based Design and Real-Time Components ................................................. 9
1.5 Timing Analysis and Time-Based Synthesis ................................................. 11

Part I Model-Based Design and Synthesis

2 Modeling, Analysis, and Implementation of Streaming Applications for Hardware Targets ......................................... 15
Kaushik Ravindran, Arkadeb Ghosal, Rhishikesh Limaye, Douglas Kim, Hugo Andrade, Jeff Correll, Jacob Kornerup, Ian Wong, Gerald Wang, Guang Yang, Amal Ekbal, Mike Trimborn, Ankita Prasad, Trung N Tran
2.1 Introduction ................................................. 16
2.2 Related Work ................................................. 18
2.3 DSP Design Module: Models and Analysis ................................................. 19
  2.3.1 Static DataFlow ................................................. 20
  2.3.2 SDF: Properties and Analysis ................................................. 20
## Contents

2.3.3 Extensions for Cyclo-Static Data Rates and Parameterization ........................................... 22

2.4 DSP Design Module: Implementation Flow ................................................................. 23
  2.4.1 Design Environment ................................................................. 23
  2.4.2 Implementation Strategy ............................................................. 26
  2.4.3 Glue Design and IP Integration ................................................. 27
  2.4.4 I/O Integration ................................................................. 27

2.5 OFDM Transmitter & Receiver Case Study .............................................................. 28
  2.5.1 Transmitter and Receiver Overview ........................................... 29
  2.5.2 Hardware Implementation ......................................................... 30
  2.5.3 Design Exploration ................................................................. 31
  2.5.4 Extensions ................................................................. 32

2.6 Summary ................................................................. 32

3 Dataflow-based, Cross-platform Design Flow for DSP Applications .................................. 35
  Zheng Zhou, Chung-Ching Shen, William Plishker, and Shuvra S. Bhattacharyya
  3.1 Introduction ................................................................. 36
  3.2 Background ................................................................. 37
    3.2.1 CFDF Dataflow Model ......................................................... 37
    3.2.2 Lightweight Dataflow ......................................................... 38
    3.2.3 The Targeted DIF Framework ............................................. 39
  3.3 From simulation to implementation ............................................................. 39
    3.3.1 Step 1: System Formulation .................................................. 40
    3.3.2 Step 2: System Validation and Profiling .................................... 42
    3.3.3 Step 3: System Optimization .................................................. 42
    3.3.4 Step 4: System Verification and Instrumentation ....................... 43
    3.3.5 Determining Buffer Sizes ..................................................... 44
    3.3.6 Discussion ................................................................. 45
  3.4 Case Study 1 - CPU/GPU ................................................................. 45
    3.4.1 Simulation ................................................................. 46
    3.4.2 Implementation ................................................................. 48
  3.5 Case Study 2 - Multicore PDSP ................................................................. 51
    3.5.1 Simulation ................................................................. 52
    3.5.2 Implementation ................................................................. 53
  3.6 Summary ................................................................. 57

Part II Model-Driven Design, Integration and Verification of Heterogeneous Models

4 Model-Driven Design of Software Defined Radio Applications Based on UML .................. 61
  Jair Gonzalez, Renaud Pacalet
  4.1 Introduction ................................................................. 61
  4.2 Related Work ................................................................. 62
  4.3 Proposed Design Methodology ................................................................. 63
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>DiplodocusDF</td>
<td>64</td>
</tr>
<tr>
<td>4.4.1</td>
<td>SDR Waveform Notations</td>
<td>65</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Target Architecture Notations and Mapping</td>
<td>66</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Performance Requirements Notations</td>
<td>67</td>
</tr>
<tr>
<td>4.5</td>
<td>Code Generation</td>
<td>67</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Model Extension Constructs</td>
<td>68</td>
</tr>
<tr>
<td>4.5.2</td>
<td>DiplodocusDF Translation Semantics</td>
<td>68</td>
</tr>
<tr>
<td>4.6</td>
<td>Runtime Environment</td>
<td>69</td>
</tr>
<tr>
<td>4.7</td>
<td>DiplodocusDF example: Welch Periodogram Detector</td>
<td>70</td>
</tr>
<tr>
<td>4.8</td>
<td>Conclusions</td>
<td>73</td>
</tr>
<tr>
<td>5</td>
<td>On Integrating EAST-ADL and UPPAAL for Embedded System Architecture Verification</td>
<td>75</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>75</td>
</tr>
<tr>
<td>5.2</td>
<td>Related Work</td>
<td>76</td>
</tr>
<tr>
<td>5.3</td>
<td>EAST-ADL and Timing Extension</td>
<td>77</td>
</tr>
<tr>
<td>5.3.1</td>
<td>EAST-ADL Core and Behavior Model</td>
<td>78</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Function Behavior Semantics</td>
<td>78</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Timing Model</td>
<td>79</td>
</tr>
<tr>
<td>5.4</td>
<td>Timed Automata and UPPAAL</td>
<td>80</td>
</tr>
<tr>
<td>5.5</td>
<td>EAST-ADL and Timed Automata Relationship</td>
<td>81</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Mapping Scheme</td>
<td>82</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Usage and Automation Considerations</td>
<td>84</td>
</tr>
<tr>
<td>5.5.3</td>
<td>System Verification</td>
<td>85</td>
</tr>
<tr>
<td>5.6</td>
<td>Brake-by-Wire Case Study</td>
<td>85</td>
</tr>
<tr>
<td>5.7</td>
<td>Discussion</td>
<td>87</td>
</tr>
<tr>
<td>6</td>
<td>Schedulability Analysis at Early Design Stages with MARTE</td>
<td>89</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>89</td>
</tr>
<tr>
<td>6.2</td>
<td>Overview of the Optimum Process</td>
<td>91</td>
</tr>
<tr>
<td>6.3</td>
<td>The Schedulability Model</td>
<td>92</td>
</tr>
<tr>
<td>6.4</td>
<td>Detailed Optimum Methodology</td>
<td>93</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Modeling Language Description</td>
<td>94</td>
</tr>
<tr>
<td>6.4.2</td>
<td>The Optimum Models</td>
<td>95</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Conformance to the Formal Schedulability Model</td>
<td>99</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Software Architecture Exploration Phase</td>
<td>99</td>
</tr>
<tr>
<td>6.5</td>
<td>Application on an Automotive Case Study</td>
<td>101</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Workload model</td>
<td>101</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Generation of the Architecture Model</td>
<td>103</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Schedulability Analysis Results</td>
<td>104</td>
</tr>
<tr>
<td>6.6</td>
<td>Related Works</td>
<td>105</td>
</tr>
<tr>
<td>6.7</td>
<td>Conclusions and Future Work</td>
<td>105</td>
</tr>
</tbody>
</table>
Part III Component-Based Design and Real-Time Components

7 Early Time-Budgeting for Component-Based Embedded Control Systems ........................................... 109
Manoj G. Dixit and S. Ramesh and Pallab Dasgupta
7.1 Introduction ......................................................... 109
7.2 Time-Budgeting Methodology ................................. 111
  7.2.1 Formalization of Component Time-Budgeting ........ 112
  7.2.2 Component Time-Budget Computation .................. 114
7.3 RDP Constraint Computation Methods ....................... 115
  7.3.1 Emptiness and Universality Check Method .......... 115
  7.3.2 Bounded Response Constraint Extraction Method .... 116
  7.3.3 Corner Point Constraint Extraction Method .......... 118
7.4 Case Studies ....................................................... 120
7.5 Conclusion and Future Work .................................... 121

8 Contract-Based Reasoning for Component Systems with Rich Interactions ..................................... 123
Susanne Graf, Roberto Passerone and Sophie Quinton
8.1 Introduction ......................................................... 124
8.2 Design methodology ............................................. 125
  8.2.1 Contract framework ..................................... 127
  8.2.2 Reasoning within a contract framework ............. 129
8.3 Circular Reasoning in Practice ................................ 131
  8.3.1 The L0 framework ..................................... 131
  8.3.2 The L1 framework ..................................... 132
  8.3.3 Relaxed circular reasoning ............................. 135
8.4 Conclusion and Future Work .................................... 136

9 Extracting End-to-end Timing Models from Component-Based Distributed Embedded Systems ............... 137
Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin
9.1 Introduction ......................................................... 137
9.2 Background and Research Problem ........................... 138
  9.2.1 The Rubus Concept ..................................... 138
  9.2.2 Problem Statement: Linking of Distributed Chains .... 140
9.3 End-to-end Timing Model ......................................... 142
  9.3.1 System Timing Model ................................... 142
  9.3.2 System Linking Model ................................. 143
9.4 Extraction of End-to-end Timing Model ....................... 143
  9.4.1 Proposed Solution .................................... 144
  9.4.2 Extraction of End-to-end Timing Model in Rubus-ICE .. 146
9.5 Related Work ....................................................... 146
9.6 Conclusion ......................................................... 149
Contents

Part IV Timing Analysis and Time-Based Synthesis

10 Distributed Priority Assignment in Real-Time Systems .......................... 153
    Moritz Neukirchner, Steffen Stein, Rolf Ernst
    10.1 Introduction ................................................. 153
    10.2 Related Work ............................................... 154
    10.3 System Model & Admission Control concept ................................. 155
    10.4 Self-Configuration Strategy ..................................... 156
    10.5 The Local Improvement Target .................................... 158
    10.6 Distributed Self-Configuration Algorithm ............................... 159
    10.7 Evaluation .................................................... 162
        10.7.1 Number of Feasible Priority Assignments ...................... 163
        10.7.2 Runtime .................................................. 164
    10.8 Conclusion .................................................... 165

11 Exploration of Distributed Automotive Systems using Compositional Timing Analysis .......................... 167
    Martin Lukasiewycz, Michael Glaß, Jürgen Teich, and Samarjit Chakraborty
    11.1 Introduction .................................................. 167
    11.2 Design Space Exploration Model ..................................... 168
        11.2.1 Model Description ......................................... 168
        11.2.2 Binary Encoding ......................................... 171
    11.3 Compositional Timing Analysis ....................................... 173
        11.3.1 Timing Model .............................................. 173
        11.3.2 Dependency-based Fixed-Point Iteration ...................... 175
        11.3.3 Fine-grained Fixed-Point Iteration ........................ 177
    11.4 Experimental Results ............................................ 178
        11.4.1 Automotive Case Study .................................... 178
        11.4.2 Design Space Exploration Results ............................ 179
        11.4.3 Timing Analysis Results .................................. 180
    11.5 Concluding Remarks ............................................. 181

12 Design and Evaluation of Future Ethernet AVB-based ECU Networks ........................................ 183
    Michael Glaß, Sebastian Graf, Felix Reimann, and Jürgen Teich
    12.1 Future Communication Media for ECU Networks ......................... 183
    12.2 Related Work .................................................. 184
    12.3 Fundamentals .................................................. 186
    12.4 VPC Model ..................................................... 190
        12.4.1 AVB Scheduling ........................................... 191
        12.4.2 Overall Ethernet AVB Model ................................ 193
    12.5 Case Study ..................................................... 194
    12.6 Conclusion ..................................................... 197

List of Figures
Contents

References ................................................................. 203
Index ................................................................. 219
This book is an edited collection of contributions in selected topics related to embedded systems modeling, analysis and synthesis. Most contributions are extended versions of papers that originally appeared throughout several workshops organized in the context of the Embedded Systems Week and Real-Time Systems Symposium in the last months of 2011. The workshops targeted topics and challenges related to the use of models for the design, analysis, and synthesis of Embedded Systems. Problems and solutions were discussed for different stages in the development process and apply to the system-level view, as well as to the design, analysis and synthesis of components and subsystems and the behaviors therein. These workshops were the WSS, Workshop on Software Synthesis, the TiMoBD, Time Analysis and Model-Based Design, and the SOMRES, Workshop on Synthesis and Optimization Methods for Real-Time Embedded Systems.

As workshop organizers and editors of this book, we believe that these are very special times for embedded and cyber-physical systems researchers and developers. A time of opportunity, because of the need and emergence of feature-rich, complex, distributed systems and the need to tame their complexity in new ways, including the adoption of model-based development, new analysis methods and design synthesis techniques, and true component-based development, in which functional and platform assemblies are correct-by-construction.

This book collects contributions on different topics, including system and software models, innovative architectures (including OS and resource managers), formal methods, model checking and analysis techniques, software synthesis, system optimization and real-time networks, with the ambitious objective of providing useful insights and innovative ideas on how to solve very complex problems throughout the entire (model-based) development cycle. Contrary to other books on the subject, we attempt at reconciling the two communities of Model-Based and Model-Driven Design, which often operate in independent ways, with only a few fortunate exceptions.

Regardless of the workshop organization, the selected paper have been organized according to their topics and divided in chapters that better fit the stages in the development process rather than an abstract classification based, for example, on lan-
guages, algorithmic solutions or methods. The intended audience includes of course the general community of embedded systems researchers, but we believe several topics and contributions should be also of interest for developers, tool vendors and development process experts. Several contributions are provided by industry developers and researchers and refer to upcoming commercial products, methods and tools. The applicability of most other results is demonstrated by use cases and/or project experiences.

We would like to acknowledge all authors, the workshop audiences that provided constructive feedback and interesting discussion, which eventually found their way into improved and new content and the assistant editors at Springer.

Berkeley, Montreal, Pisa, Dortmund, March 2013

Alberto Sangiovanni-Vincentelli,
Haibo Zeng,
Marco Di Natale,
Peter Marwedel
Chapter 8
Contract-Based Reasoning for Component Systems with Rich Interactions

Susanne Graf, Roberto Passerone and Sophie Quinton

Abstract In this paper we propose a rule unifying circular and non-circular assume-guarantee reasoning and show its interest for contract-based design and verification. Our work was motivated by the need to combine, in the top-down methodology of the FP7 SPEEDS project, partial tool chains for two component frameworks derived from the HRC model and using different refinement relations. While the L0 framework is based on a simple trace-based representation of behaviors and uses set operations for defining refinement, the more elaborated L1 framework offers the possibility to build systems of components with complex interactions. Our approach in L1 is based on circular reasoning and results in a method for checking contract dominance which does not require the explicit composition of contracts. In order to formally relate results obtained in L0 and L1, we provide a definition of the minimal concepts required by a consistent contract theory and propose abstract definitions which smoothly encompass hierarchical components. Finally, using our relaxed rule for circular reasoning, we show how to use together the L0 and L1 refinement relations and as a result their respective tool chains.

Susanne Graf
VERIMAG/CNRS, 2 avenue de Vignate, 38610 Gières, France.
e-mail: susanne.graf@imag.fr

Roberto Passerone
DISI/University of Trento, via Sommarive 5, 38123 Trento, Italy.
e-mail: roberto.passerone@unitn.it

Sophie Quinton
IDA/TU Braunschweig, Hans-Sommer-Straße 66, 38106 Braunschweig, Germany.
e-mail: quinton@ida.ing.tu-bs.de
8.1 Introduction

Contract and interface frameworks are emerging as the formalism of choice for system designs that require large and distributed teams, or where the supply chain is complex [242, 64, 65]. This style of specification is typically employed for top-down design of systems of components, where the system under design is built by a sequence of decomposition and verification steps. In this paper we present and study some distinctive features of contract theories for frameworks in which the interaction between components is “rich”, i.e., more complex than the usual input/output (I/O) communication. One such component framework is BIP [26] which allows multi-party synchronizations scheduled according to priorities. In addition, we show how to combine results obtained using different contract refinement relations.

Our work has its practical motivation in the component framework HRC [29, 32, 64, 65] (standing for Heterogeneous Rich Components) defined in the FP7 IP project SPEEDS [255], which has been reused in the FP7 STREP project COMBEST [60] and the ARTEMIS project CESAR [51]. The HRC model defines component properties in terms of extended transition systems and provides several composition models, ranging from low-level semantic composition to composition frameworks underlying the design tools used by system designers. More precisely, HRC is organized around two abstraction levels called L0 and L1 and describing respectively the core level and the analysis tool level of HRC [210]. That is, L0 determines the expressive power of the entire model and there exist translations from L1 models to L0. On the other hand, L1 extends the core model with concepts such as coordination mechanisms — the rich interactions mentioned in the title. Analysis tools can then take advantage of these additional concepts to make system descriptions more concise and therefore verification more efficient.

Our objective is to allow combined use of synchronous tools like Simulink [269] for L0 and synchronization-based tools like BIP for L1, which have complementary strengths. For example, Simulink is very convenient for modeling physical dynamic systems or streaming applications. In contrast BIP, which encompasses rich interactions, is well adapted for describing the dynamic behavior of sets of components depending on available resources for memory, energy, communication bandwidth etc. We are interested in this paper in the relation between the L0 and L1 contract frameworks as we want to use verification results established in L1 for further reasoning within L0. The presence of rich interactions in L1 makes contract composition problematic and leads us to focus instead on circular reasoning, which allows a component and its environment to be refined concurrently — each one relying on the abstract description of its context — and entails an interesting rule for proving dominance, i.e., refinement between contracts. In order to relate L0 and L1, we define a generic contract framework that uses abstract composition operators and thus encompasses a variety of interaction models, including those for L0 and L1. Finally, we show how to use a relaxed rule for circular reasoning to combine partial tool chains for both frameworks into a complete tool chain for our methodology.

To the best of our knowledge, this is the first time that a rule combining different refinement relations is proposed and used to unify two contract frameworks.
While circular reasoning has been extensively studied, e.g. in [6, 173], existing work focuses on finding sufficient conditions for soundness of circular reasoning while we focus on how to use circular reasoning in a contract-based methodology. Non-circular assume-guarantee reasoning is also a topic of intense research focused on finding a decomposition of the system that satisfies the strong condition imposed on at least one of its components [59]. Finally, our contract frameworks are related to interface automata [2]. Since de Alfaro and Henzinger’s seminal paper many contract and interface theories have been developed for numerous frameworks (see e.g. [153, 282, 71, 229, 231, 230] to name just a few). However these theories focus on composition of contracts while we strive to avoid that and furthermore they do not handle rich interactions. Examples include [154, 230] based on modal I/O automata and [282] defining relational interfaces for capturing functional dependencies between inputs and outputs of an interface. Preliminary versions of our contract framework appeared in [224, 109] but did not address the question of combining results obtained for different refinements.

This paper is structured as follows: Section 8.2 describes our design and verification methodology as well as generic definitions of component and contract framework. It then discusses sufficient reasoning rules for establishing dominance without composing contracts. Section 8.3 presents how the proposed approach is applied to the \( L_0 \) and \( L_1 \) frameworks. In particular it shows how their different satisfaction relations may be used together using relaxed circular reasoning and discusses practical consequences of this result. Section 8.4 concludes the paper. The proofs of all theorems presented in this paper are presented in [102].

### 8.2 Design Methodology

Our methodology is based on an abstract notion of component. We characterize a component \( K \) by its interface defined as a set \( \mathcal{P} \) of ports which describe what can be observed by its environment. We suppose given a global set of ports \( \text{Ports} \), which all sets of ports in the following are subsets of. In addition, components are also characterized by their behavior. At this level of abstraction, we are not concerned with how behaviors are represented and develop our methodology independently of the particular formalism employed. Interactions (potentially complex) between components are expressed using the concept of glue operator [254]. A glue defines how the ports of different components are connected and the kind of synchronization and data exchange that may take place. We denote the composition of two components \( K_1 \) and \( K_2 \) through a glue \( gl \) as \( gl\{K_1, K_2\} \). The glue must be defined on the union of the ports \( \mathcal{P}_1 \) and \( \mathcal{P}_2 \) of the components.

In order to separate the implementation phase of a component from its integration into the system under design, we use contracts [32, 30, 224]. A contract for a component \( K \) describes the interface \( \mathcal{P} \) of \( K \), the interaction between \( K \) and its environment \( E \), the expected behavior of \( E \), called the assumption \( A \) of the contract, and the expected behavior of \( K \), called the guarantee \( G \). Assumptions and guarantees
are in turn expressed as components, defining the interface and the behavior that are considered acceptable from the environment and from the component. Thus, formally, a contract $C$ for an interface $P$ is a triple $(A, gl, G)$, where $gl$ is a glue operator on $P \cup P_A$ for some $P_A$ disjoint from $P$; the assumption $A$ is a component with interface $P_A$; and the guarantee $G$ is a component with interface $P$. Note that the interface of the environment is implicitly defined by $gl$. Graphically, we represent contracts as in Figure 8.1.

$$C = (A, gl, G)$$

![Fig. 8.1: A contract $(A, gl, G)$ for an interface $P$](image)

The proof of conformance, decomposition and dominance is shown in Figure 8.2.

$$K_1 \models \mathcal{C}_1 \quad K_2 \models \mathcal{C}_2 \quad K_3 \models \mathcal{C}_3$$

$satisfaction$

$$gl\{A, gl\{K_1, K_2, K_3\}\} \preceq \varphi$$

![Fig. 8.2: Proof of $gl\{A, gl\{K_1, K_2, K_3\}\} \preceq \varphi$](image)
From a macroscopic point of view, we adopt a top-down design and verification methodology (see Figure 8.2) in which global requirements are pushed progressively from the top-level system to the low-level atomic components. As usual, this is just a convenient representation; in real life, the final picture is always obtained in several iterations alternatively going up and down the hierarchy [213]. While the refinement relation between a specification and an implementation is at the core of component-based design, in contract-based design refinement takes different forms depending on whether it relates a system to a specification, two contracts or an implementation to a contract. In this paper we use a methodology which divides the design and verification process into three steps corresponding to these three forms of refinement.

We assume that the system $K$ under construction has to realize a global requirement $\varphi$ together with an environment on which we may have some knowledge, expressed by a property $A$. Both $\varphi$ and $A$ are expressed w.r.t. the interface $\mathcal{P}$ of $K$. We proceed as follows: (1) define a contract $\mathcal{C} = (A, gl, G)$ for $\mathcal{P}$ such that $gl\{A, G\}$ conforms to $\varphi$; (2) decompose $K$ as subcomponents $K_i$ connected through a glue operator $gl_I$ and provide a contract $\mathcal{C}_i$ for each of them; possibly iterate this step if needed; (3) prove that whenever a set of implementations $K_i$ satisfy their contracts $\mathcal{C}_i$, then their composition satisfies the top-level contract $\mathcal{C}$ (dominance) — and thus guarantee $\varphi$; (4) provide such implementations.

The correctness proof for a particular system is therefore split into 3 phases: conformance (denoted $\ll$) of the system defined by the top-level contract $\mathcal{C}$ to $\varphi$; dominance of $\mathcal{C}$ by the composition of the set of contracts $\{\mathcal{C}_i\}$ through $gl_I$; and satisfaction (denoted $|=\) of each $\mathcal{C}_i$ by the corresponding implementation $K_i$. Thus, conformance relates closed systems, dominance relates contracts, while satisfaction relates components to contracts.

The assumption of $\mathcal{C}_1$ is represented as one component $A_1$ while in the actual system $K_1$ will be used in the context of three components, namely $K_2$, $K_3$ and $A$. Thus, we need to relate the actual glues $gl$ and $gl_I$ to the glue $gl_1$ of $\mathcal{C}_1$. In other words, we need a glue $gl_{E_1}$ to compose $K_2$, $K_3$ and $A$ as well as an operation $\circ$ on glues such that $gl \circ gl_I = gl_1 \circ gl_{E_1}$. In most cases, $\circ$ cannot simply be composition of functions and has to involve some flattening of the system.

### 8.2.1 Contract Framework

To summarize, we consider a component framework that smoothly supports complex composition operators and hierarchical components. The elements of the component framework are as follows:

**Definition 1 (Component framework).** A component framework is defined by a tuple $(\mathcal{K}, GL, \circ, \cong)$ where:

- $\mathcal{K}$ is a set of components. Each component $K \in \mathcal{K}$ has as interface a set of ports, denoted $\mathcal{P}_K$ and subset of our global set of ports $\text{Ports}$. 
• GL is a set of glues. A glue is a partial function \(2^\mathcal{K} \rightarrow \mathcal{K}\) transforming a set of components into a new composite component. Each \(gl \in GL\) is defined on a set of ports \(S_{gl}\), called support set, and defines a new interface \(\mathcal{P}_{gl}\) for the new component, called exported interface. \(K = gl([K_1, ..., K_n])\) is defined if \(K_1, ..., K_n \in \mathcal{K}\) have disjoint interfaces, \(S_{gl} = \bigcup_{i=1}^{n} \mathcal{P}_{K_i}\), and \(\mathcal{P}_{K} = \mathcal{P}_{gl}\).

• | is a partial operator on GL, called flattening, to compose glues. \(gl \circ gl'\) is defined if \(\mathcal{P}_{gl} \subseteq S_{gl}\). Its support set is \(S_{gl} \setminus \mathcal{P}_{gl} \cup S_{gl'}\) and its interface is \(\mathcal{P}_{gl}\).

• \(\equiv\subseteq \mathcal{K} \times \mathcal{K}\) is an equivalence relation between components.

We simplify our notation by writing \(gl([K_1, ..., K_n])\) instead of \(gl([K_1, ..., K_n])\). The equivalence relation \(\equiv\) is typically used for relating composite components with their semantics given as an atomic component. More importantly, \(|\) must be coherent with \(\equiv\) in the sense that \(gl\{gl'([\mathcal{K}_1], \mathcal{K}_2) \equiv (gl \circ gl')\{\mathcal{K}_1 \cup \mathcal{K}_2\}\) for any sets of components \(\mathcal{K}\) such that all terms are defined.

After formalizing generic properties required from a component framework, we now define the relations used in the methodology for dealing with contracts. Satisfaction is usually considered as a derived relation and chosen as the weakest relation implying conformance and preserved by composition. We loosen the coupling between satisfaction and conformance to obtain later stronger reasoning schemata for dominance. Furthermore, we propose a representation of satisfaction as a set of refinement under context relations denoted \(\sqsubseteq_{A,gl}\) and such that \(K \sqsubseteq_{A,gl} G\) iff \(K \models (A, gl, G)\).

**Definition 2 (Contract framework).** A contract framework is defined by a tuple \((\mathcal{K}, GL, |, \equiv, \equiv, \models)\) where:

1. \((\mathcal{K}, GL, |, \equiv)\) is a component framework.
2. \(\equiv\subseteq \mathcal{K} \times \mathcal{K}\) is a preorder called conformance relating components having the same interface.
3. \(\models\) is a relation called satisfaction between components and contracts s.t.: the relations \(\sqsubseteq_{A,gl}\) defined by \(K \sqsubseteq_{A,gl} G\) iff \(K \models (A, gl, G)\) are preorders; and, if \(K \models (A, gl, G)\) then \(gl\{A, K\} \equiv gl\{A, G\}\).

Our definition of satisfaction emphasizes the fact that \(\models\) can be seen as a set of refinement relations where \(K \sqsubseteq_{A,gl} G\) means that \(K\) refines \(G\) in the context of \(A\) and \(gl\). The condition which relates satisfaction and conformance ensures that the actual system \(gl\{A, K\}\) will conform to the global requirement \(\varphi\) discussed in the methodology because \(\equiv\) is transitive and \(gl\{A, G\} \equiv \varphi\).

**Example 1.** Typical notions of conformance for labeled transition systems are trace inclusion and its structural counterpart simulation. For these, satisfaction is usually defined as the weakest relation implying conformance.

\[K \models (A, gl, G) \equiv gl\{K, A\} \subseteq gl\{G, A\}\]

Dominance is a key notion for reasoning about contracts rather than using refinement between components. Proving that a contract \(C\) dominates \(C'\) means showing...
that every component satisfying $\mathcal{C}$ also satisfies $\mathcal{C}'$. However, a dominance check involves in general not just a pair of contracts: a typical situation would be the one depicted in Figure 8.2, where a set of contracts $\{\mathcal{C}_i\}_{i=1}^n$ are attached to disjoint interfaces $\{\mathcal{P}_i\}_{i=1}^n$. Besides, a glue $gl_I$ is defined on $P = \bigcup_{i=1}^n \mathcal{P}_i$ and a contract $\mathcal{C}$ is given for $P$. In this context, a set of contracts $\{\mathcal{C}_i\}_{i=1}^n$ dominates a contract $\mathcal{C}$ w.r.t. a glue $gl_I$ if any set of components satisfying contracts $\mathcal{C}_i$, when composed using $gl_I$, makes a component satisfying $\mathcal{C}$.

**Definition 3 (Dominance).** Let $\mathcal{C}$ be a contract on $P$, $\{\mathcal{C}_i\}_{i=1}^n$ a set of contracts on $P_i$ and $gl_I$ a glue such that $S_{gl_I} = \bigcup_{i=1}^n \mathcal{P}_i$ and $P = P_{gl_I}$. Then $\{\mathcal{C}_i\}_{i=1}^n$ dominates $\mathcal{C}$ with respect to $gl_I$ iff for all components $\{K_i\}_{i=1}^n$:

$$ (\forall i : K_i \models \mathcal{C}_i) \implies gl_I\{K_1, \ldots, K_n\} \models \mathcal{C} \tag{8.1} $$

Note that this formal definition of dominance does not help establishing dominance in practice because looking at all possible components satisfying a contract is not realistic. What we need is a sufficient condition that refers to assumptions and guarantees, rather than components. One such condition is when the composition of the low-level guarantees $G_i$ satisfies the top-level contract $\mathcal{C}$ and furthermore each low-level assumption $A_i$ is discharged by the abstraction of its environment defined by the guarantees of the other components. Formally:

$$ \left\{ \begin{array}{l}
  gl_I\{G_1, \ldots, G_n\} \models \mathcal{C} \\
  \forall i : gl_{E_i}\{A, G_1, \ldots, G_{i-1}, G_{i+1}, \ldots, G_n\} \models \mathcal{C}_i^{-1}
\end{array} \right. \tag{8.1} $$

where for any contract $\mathcal{C}_i = (A_i, gl_i, G_i)$ we use the notation $\mathcal{C}_i^{-1}$ to denote the contract $(G_i, gl_i, A_i)$.

In the next subsection, we provide two rules which indeed make the previous condition sufficient for establishing dominance: one is similar to circular assume-guarantee reasoning and the other one deals with preservation of satisfaction by composition. This result is particularly significant because one can check dominance while avoiding composition of contracts, which is impossible in the general case and leads to state explosion in most concrete contract frameworks.

### 8.2.2 Reasoning within a Contract Framework

We use here the representation of satisfaction as a set of refinement under context relations $\sqsubseteq_{A, gl}$ where $K \sqsubseteq_{A, gl} G$ if and only if $K \models (A, gl, G)$. The usual non-circular assume-guarantee rule reads as follows in our context:

1 One may also need to ensure that the assumptions of the low-level contracts are indeed satisfied in the actual system. This is achieved by strengthening the definition with:

$$ \forall E \text{ on } A, \text{ if } E \models (G', gl', A') \text{ then } E \models (G, gl, A) $$
where $E \sqsubseteq A$ denotes that for any component $G$ and $gl$ such that $\sqsubseteq_{G,gl}$ is defined $E \sqsubseteq_{G,gl} A$. This rule relates the behavior of $K$, when composed with the abstract environment $A$, to the behavior of $K$, when composed with its actual environment $E$. However it is quite limited as it imposes a very strong condition on $E$. Hence the following rule which is commonly referred to as circular reasoning.

$$K \sqsubseteq_{A,gl} G \land E \sqsubseteq A \implies K \sqsubseteq_{E,gl} G$$

Note that $E$ and $K$ may symmetrically rely on each other. For a given contract framework, this property can be proven by an induction based on the semantics of composition and refinement. Unfortunately, circular reasoning is not sound in general. In particular it does not hold for parallel composition with synchronizations (as in Petri nets or process algebras) or instantaneous mutual dependencies between inputs and outputs (as in synchronous formalisms). The following example illustrates one possible reason for the non validity of circular reasoning.

**Example 2.** Consider a contract framework where components are labeled transition systems and composition is strong synchronization between corresponding labels and interleaving of others denoted $\parallel$. Define conformance as simulation and satisfaction as the usual relation defined in Example 1. The circular reasoning rule translates into: if $K \parallel A$ is simulated by $G \parallel A$ and $E \parallel G$ is simulated by $A \parallel G$ then $K \parallel E$ is simulated by $G \parallel E$. In the example of Figure 8.3, both $G$ and $A$ forbid a synchronization between $b_K$ and $b_E$, since they can rely on respectively $G$ and $A$ to forbid their actual occurrence. But obviously, the composition $K \parallel E$ now allows a synchronization between $b_K$ and $b_E$.

![Diagram](image)

Fig. 8.3: $K \parallel A \not\equiv G \parallel A$ and $E \parallel G \not\equiv A \parallel G$ but $K \parallel E \not\equiv G \parallel E$.

Note that this satisfaction relation can be strengthened to obtain a more restrictive relation for which circular reasoning is sound. This is the approach taken for the L1 contract framework in Section 8.3.2, where we need circular reasoning to avoid composition of contracts.

\[^2\] Note that non-determinism is another reason here for the non validity of circular reasoning.
A second rule which is used for compositional reasoning in most frameworks is:

\[ I \subseteq S \quad \text{then} \quad I \parallel E \subseteq S \parallel E. \]

It states that if an implementation \( I \) refines its specification \( S \) then it refines it in any environment \( E \). The equivalent of this rule for satisfaction is more complex as refinement here relates closed systems.

**Definition 4.** Satisfaction \( \models \) is preserved by composition iff for any component \( E \), \( gl \) such that \( S_{gl} = P_E \cup P \) for some \( P \) such that \( P \cap P_E = \emptyset \) and \( gl_E, E_1, E_2 \) such that \( E = gl_E \{ E_1, E_2 \} \), the following holds for any components \( I, S \) on \( P \):

\[ I \subseteq_{E, gl} S \implies gl_1 \{ I, E_1 \} \subseteq_{E_2, gl_2} gl_1 \{ S, E_1 \} \]

where \( gl_1 \) and \( gl_2 \) are such that \( gl \circ gl_E = gl_2 \circ gl_1 \).

We now have the ingredients to formalize our sufficient condition for dominance. This condition reduces a dominance proof to a set of satisfaction checks, one for the refinement between the guarantees and \( n \) for discharging individual assumptions.

**Theorem 1.** Suppose that circular reasoning is sound and satisfaction is preserved by composition. If \( \forall i \exists gl_{E_i} : gl \circ gl_{I_i} = gl_{I_i} \circ gl_{E_i} \) then to prove that \( \{ C_i \}_{i=1}^n \) dominates \( C \) w.r.t. \( gl \), it is sufficient to prove that condition (1) holds.
set of behaviors all the identity traces. Composition can then be taken as the intersection of the sets of behaviors of the components, together with the glue. To make this work, we must also equalize the ports of all trace sets using inverse projection \( \text{proj}^{-1}_{\mathcal{P}, \mathcal{P}} \), which extends behaviors over \( \mathcal{P}_1 \) with the appropriate additional ports of \( \mathcal{P} \). If we denote the interface of the composite as \( \mathcal{P}_{gl} \), and if \( \mathcal{K} = \{K_1, \ldots, K_n\} \) is a set of components such that \( \mathcal{P}_1, \ldots, \mathcal{P}_n \) are pairwise disjoint, then a glue \( gl \) for \( \mathcal{K} \) is a component \( K_{gl} \) defined on the ports \( \mathcal{P} = \mathcal{P}_{gl} \cup (\bigcup_{i=1}^{n} \mathcal{P}_i) \), and:

\[
K = gl\{K_1, \ldots, K_n\}
= \text{proj}_{\mathcal{P}_{gl}, \mathcal{P}} \left( K_{gl} \cap \text{proj}^{-1}_{\mathcal{P}_1, \mathcal{P}} (K_1) \cap \cdots \cap \text{proj}^{-1}_{\mathcal{P}_n, \mathcal{P}} (K_n) \right)
\]

The definition of \( \circ \) is straightforward: since glues are themselves components, their composition follows the same principle as component composition. Finally, the \( \equiv \) relation on \( \mathcal{K} \) is taken as equality of sets of traces.

In the L0 model there exists a unique maximal component satisfying a contract \( \mathcal{C} \), namely \( M_{\mathcal{C}} = G \cup \neg A \), where \( \neg \) denotes the operation of complementation on the set of all behaviors over ports \( \mathcal{P}_A \). A contract \( \mathcal{C} = (A, G) \) is in canonical form when \( G = M_{\mathcal{C}} \). Every contract has an equivalent contract in canonical form, which is obtained by replacing \( G \) with \( M_{\mathcal{C}} \). The operation of computing a canonical form is well defined, since the maximal implementation is unique, and it is idempotent. It is easy to show that \( K \models \mathcal{C} \) if and only if \( K \subseteq M_{\mathcal{C}} \).

The L0 contract framework has strong compositional properties, which derive from its simple definition and operators [30]. The theory, however, depends on the effectiveness of certain operators, complementation in particular, which are necessary for the computation of canonical forms. While the complete theory can be formulated without the use of canonical forms, complementation remains fundamental in the definition of contract composition, which is at the basis of system construction. Circular reasoning is not sound for contracts which are not in canonical form (Example 2 is a counter-example in that case). This is a limitation of the L0 framework, since working with canonical forms could prove computationally hard.

### 8.3.2 The L1 Framework

L1 composition is based on interactions, which involve non-empty sets of ports. An interaction is defined by the components that synchronize when it takes place and the ports through which these components synchronize. Interactions are structured into connectors which are used as a mechanism for encapsulation: only these connectors appear at the interface of a composite component. This enables to abstract the behavior of a component in a black-box manner, by describing which connector is triggered but not exactly which interaction takes place. Furthermore L1 is expressive enough to encompass synchronous systems.
**Definition 5.** An atomic component on a interface $\mathcal{P}$ is defined by an LTS $K = (Q, q^0, 2^\mathcal{P}, \rightarrow)$, where $Q$ is a set of states, $q^0$ an initial state and $\rightarrow \subseteq Q \times 2^\mathcal{P} \times Q$ is a transition relation.

Note that atomic components are labeled by sets of ports rather than ports because we allow several ports of a component to be triggered at the same time.

**Definition 6.** An interaction is a non-empty set of ports. A connector $\gamma$ is defined by a set of ports $S_\gamma$ called the support set of $\gamma$, a port $p_\gamma$ called its exported port and a set $\mathcal{I}(\gamma)$ of interactions in $S_\gamma$.

The notions of support set and exported port are illustrated in Figure 8.4, where connectors relate in a composition a set of inner ports (of the subcomponents) to an outer port (of the composite component). One should keep in mind that a connector $\gamma$, and thus the exported port $p_\gamma$, represents a set of interactions rather than a single interaction.

Typical connectors represent rendezvous (only one interaction, equal to the support set), broadcast (all the interactions containing a specific port called trigger) and also mutual exclusion (some interactions but not their union).

![Fig. 8.4: The role of connectors in a composition](image)

We now define glues as sets of connectors which may be used together in order to compose components.

**Definition 7.** A glue $gl$ on a support set $S_g$ is a set of connectors with distinct exported ports and with support sets included in $S_g$.

A glue $gl$ defines as exported interface $\mathcal{P}_{gl}$ the set $\{p_\gamma \mid \gamma \in gl\}$. Besides, $\mathcal{I}(gl)$ denotes the set of all interactions of the connectors in $gl$, i.e.: $\mathcal{I}(gl) = \bigcup_{\gamma \in gl} \mathcal{I}(\gamma)$. In Figure 8.4, $gl$ is composed of connectors $\gamma$ and $\gamma'$ and defines a composite component denoted $gl\{K_1, K_2\}$.

**Definition 8.** A component is either an atomic component or it is inductively defined as the composition of a set of components $\{K_i\}_{i=1}^n$ with disjoint interfaces $\{\mathcal{P}_i\}_{i=1}^n$ using a glue $gl$ on $\mathcal{P} = \bigcup_{i=1}^n \mathcal{P}_i$. Such a composition is called a composite component on $\mathcal{P}_{gl}$ and it is denoted $gl\{K_i\}_{i=1}^n$. 

So far, we have defined components and glues. Glues can be composed so as to allow flattening of components. Such a composition requires to handle hierarchical connectors built by merging connectors defined at different levels of hierarchy. The definition of the operator \( \circ \) used for this purpose is omitted here and can be found in [102]. Connectors whose exported ports and support sets are not related are called disjoint and need not be composed. The operator \( \circ \) is then easily extended to glues: the composition \( gl \circ gl' \) of two glues \( gl \) and \( gl' \) is obtained from \( gl \cup gl' \) by inductively composing all connectors which are not disjoint.

We can now formally define the flattened form of a component. This in turn will allow us to provide an equivalence relation between components based on the semantics of their flattened form. A component is called flat if it is atomic or of the form \( gl\{K_1,\ldots,K_n\} \), where all \( K_i \) are atomic components. A component that is not flat is called hierarchical. A hierarchical component \( K \) is of the form \( gl\{K_1,\ldots,K_n\} \) such that at least one \( K_i \) is composite. Thus, such a \( K \) can be represented as \( gl\{gl'\{s^1\},\ldots,gl'\{s^n\}\} \), where \( s^1 \) and \( s^n \) are sets of components.

Definition 9. The flattened form of a component \( K \) is denoted \( flat(K) \) and defined inductively as:

- if \( K \) is a flat component, then \( flat(K) \) is equal to \( K \).
- otherwise, \( K \) is of the form \( gl\{gl'\{s^1\},\ldots,gl'\{s^n\}\} \), and then \( flat(K) \) is the flattened form of \( (gl \circ gl')\{\bigcup s^i\} \).

Definition 10. The semantics \( \llbracket K \rrbracket \) of a flat component \( K = gl\{K_1,\ldots,K_n\} \) is defined as \( (Q,q^0,\mathcal{J}(gl),\rightarrow) \), where \( Q = \prod_{i=1}^n Q_i \), \( q^0 = (q^0_1,\ldots,q^0_n) \), and \( \rightarrow \) is such that: given two states \( q^1 = (q^1_1,\ldots,q^1_n) \) and \( q^2 = (q^2_1,\ldots,q^2_n) \) in \( Q \) and an interaction \( \alpha \in \mathcal{J}(gl) \), \( q^1 \xrightarrow{\alpha} q^2 \) if and only if for all \( q^1_i, q^2_i \xrightarrow{\alpha_i} \), where \( \alpha_i = \alpha \cap \mathcal{P}_i \).

We use the convention that \( \forall q : q^0 \xrightarrow{0} q \) so components not involved in an interaction do not move. Thus the semantics of a flat component is obtained as the composition of its constituting LTS where labels are synchronized according to the interactions of \( \mathcal{J}(gl) \).

We then define equivalence \( \cong \) as follows: two components are equivalent if their flattened forms have the same semantics. Note that in practice one would prefer to define the semantics of a hierarchical component as a function of the semantics of its constituting components. In presence of encapsulation this requires to distinguish between closed and open systems and thus to provide two different semantics. Details can be found in [102].

We now have the ingredients for defining the L1 component framework and we focus on its contract framework.

Definition 11. \( K_1 \cong^{L1} K_2 \) if and only if \( \llbracket K_1 \rrbracket \) is simulated by \( \llbracket K_2 \rrbracket \).

Thus L1-conformance is identical to L0-conformance for components without non-observable non-determinism, and otherwise stronger. Note that in verification tools, in order to check trace inclusion efficiently, one will generally check simulation anyway. Satisfaction is defined as follows.
Definition 12. A component $K$ satisfies a contract $C = (A, gl, G)$ for $\mathcal{P}_K$, denoted $K \models \mathcal{L}^L_1 (A, gl, G)$, if and only if:

$$\{g\{K, A_{det}\} \preceq \mathcal{L}^L_1 g\{G, A_{det}\}, (q_K, q_A) \mathcal{R} (q_G, q'_A) \land \exists q'_K : q_K \xrightarrow{\alpha} q'_K \implies \exists q'_G : q_G \xrightarrow{\alpha} q'_G$$

where $A_{det}$ is the determinization of $A$, $\mathcal{R}$ is the relation on states proving that $g\{K, A_{det}\} \preceq \mathcal{L}^L_1 g\{G, A_{det}\}$ and $\alpha \in 2^{\mathcal{P}_K}$ is such that $\exists \alpha' \in \mathcal{I}(gl) : \alpha \subseteq \alpha'$.

Thus $\models \mathcal{L}^L_1$ strengthens the satisfaction relation used in the $\mathcal{L}^0$ framework by: 1) determinizing $A$; 2) requiring every transition of $K$ to have a counterpart in each related state of $G$ — unless it is structurally forbidden by $gl$ — but the target states of the transition need to be related only if the environment allows this transition. As a consequence, $\models \mathcal{L}^L_1$ allows circular reasoning.

8.3.3 Relaxed Circular Reasoning

We have presented in the previous sections two contract frameworks developed in the SPEEDS project. We show now how we use their respective tool chains together. Unifying the $\mathcal{L}^0$ and $\mathcal{L}^1$ component frameworks is quite straightforward. Nevertheless, we have introduced two different notions of satisfaction: $\models \mathcal{L}^0$ and $\models \mathcal{L}^1$ where the second one is strictly stronger than the first one. To combine results based on $\mathcal{L}^0$ and $\mathcal{L}^1$, we propose a rule called relaxed circular reasoning for two (possibly different) refinement relations:

$$K \sqsubseteq^1_{A, gl} G \land E \sqsubseteq^2_{G, gl} A \implies K \sqsubseteq^1_{E, gl} G$$

This rule generalizes circular and non-circular reasoning by not restricting $\sqsubseteq^2_{G, gl}$ to refinement under context $\sqsubseteq^1_{G, gl}$ or refinement in any context $\sqsubseteq^1$. Depending on which relation is the most restrictive it can be used in two different ways:

1. If the first relation allows circular reasoning and is stronger than the second one (i.e. $K \sqsubseteq^1_{A, gl} G \implies K \sqsubseteq^2_{A, gl} G$) then our new rule relaxes circular reasoning by requiring $E \sqsubseteq^2_{G, gl} A$ rather than $E \sqsubseteq^1_{G, gl} A$.
2. Symmetrically, if the first relation does not allow circular reasoning and refinement in any context $\sqsubseteq^1$ is stronger than the second one then this rule relaxes non circular reasoning by requiring $E \sqsubseteq^2_{G, gl} A$ rather than $E \sqsubseteq^1 A$.

Interestingly, relaxed circular reasoning can be used both ways for $\mathcal{L}^0$- and $\mathcal{L}^1$-satisfaction. First it leads to a relaxed sufficient condition for dominance in $\mathcal{L}^1$.

Theorem 2. $K \sqsubseteq^L_1 G \land E \sqsubseteq^L_0 A$ implies $K \sqsubseteq^L_1_{E, gl} G$.

Theorem 3. If $\forall i \exists g|_{E_i} : gl \circ gl_i = gl_i \circ g|_{E_i}$ the following is sufficient to prove that $C$ dominates $\{g|_i\}_{i=1..n}$ w.r.t. $gl$: 


\[
\begin{align*}
\{ \text{gl}_1(G_1, \ldots, G_n) \} & \models^{L1} \mathcal{C}^0 \\
\forall i : \text{gl}_{E^i}(A, G_1, \ldots, G_{i-1}, G_{i+1}, \ldots, G_n) & \models^{L0} \mathcal{C}^{-1}_i
\end{align*}
\]

In that case, checking that contracts \( \{ \mathcal{C}_i \}_{i=1}^n \) L1-dominate a contract \( \mathcal{C} \) requires one L1-satisfaction check and \( n \) L0-satisfaction checks. This is particularly interesting since checking L0-satisfaction may be achieved by using other tools or approaches (that may not need circular reasoning). Moreover, dominance can be established more often as L1-satisfaction is stronger than L0-satisfaction. Second:

**Theorem 4.** \( K \sqsubseteq_{A, gl}^{L0} G \land E \sqsubseteq_{G, gl}^{L1} A \) implies \( K \sqsubseteq_{E, gl}^{L0} G \).

This result made it possible in SPEEDS to incorporate results from tools checking L0-satisfaction with results obtained through L1-dominance (implemented by a set of L1-satisfaction checks), thus building a complete tool chain.

### 8.4 Conclusion and Future Work

The work presented in this paper has been motivated by the necessity of combining contract-based verification tools and corresponding results for two component frameworks L0 and L1 defined in the context of the European SPEEDS project. In particular, we were interested in using dominance results established in L1 — and which cannot be obtained using the L0 refinement relation — for further reasoning in L0. To that purpose, we have presented an abstract notion of contract framework for a given component framework that defines three different notions of refinement, that is, conformance, dominance and satisfaction. We show how to derive these notions from refinement of closed systems and refinement under context and we provide a methodology for compositional and hierarchical verification of global properties.

We have studied circular reasoning as a powerful means for proving dominance. As circular reasoning does not always hold for usual notions of refinement, we provide proof rules for dominance relying on a relaxed notion of circular reasoning based on two notions of refinement. We have then shown that our abstract framework is general enough to represent both L0 and L1 as specific instances and proved that the L0 and L1 refinement relations satisfy the condition for relaxed circular reasoning.

This approach was applied to only simple case studies in the SPEEDS project and should therefore rather be seen as a proof of concept. The practical relevance of such an approach is that it opens up ways of connecting tools that work at different levels of abstraction, and relate their results to prove stronger properties. In addition, our results relax the requirements on the tools, since circular reasoning would not be needed at the L0 level.

**Acknowledgements** This work was supported in part by the EU projects COMBEST (n. 215543) and ArtistDesign (n. 214373).
References

18. AUTOSAR Consortium: AUTOSAR Technical Overview, Version 2.2.2. AUTOSAR – AU-
http://autosar.org
C., Sangiovanni-Vincentelli, A., Sentovich, E., Suzuki, K., Tabbara, B. (eds.): Hardware-
lishers, Norwell, MA, USA (1997)
A.L.: Metropolis: An integrated electronic system design environment. IEEE Computer
10.1109/TSE.2004.9. URL http://dx.doi.org/10.1109/TSE.2004.9
for real-time systems design. In: Proceedings of the 32nd EUROMICRO Conference on
Software Engineering and Advanced Applications, EUROMICRO ’06, pp. 108–117. IEEE
URL http://dx.doi.org/10.1109/EUROMICRO.2006.14
24. Bartolini, C., Lipari, G., DiNatale, M.: From functional blocks to the synthesis of the ar-
chitectural model in embedded real-time applications. In: Proceedings of the 11th IEEE
Real Time on Embedded Technology and Applications Symposium, RTAS ’05, pp. 458–
URL http://dx.doi.org/10.1109/RTAS.2005.24
25. Bartolini, C., Lipari, G., Natale, M.D.: From Functional Blocks to the Synthesis of the Ar-
chitectural Model in Embedded Real-time Applications. In: IEEE Real-Time and Embedded
27. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: M. Bernardo, F. Corradini
Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-
Concurrency and Petri Nets, LNCS, vol. 3098, pp. 87–124. Springer Berlin / Heidelberg
(2004)
ple viewpoint contract-based specification and design. In: F.S. de Boer, M.M. Bonsangue, 
S. Graf, Willem-Paul de Roever (eds.) Formal Methods for Component-based Objects and Objects, 6th 
International Symposium (FMCO 2007), Amsterdam, The Netherlands, October 24–26, 2007, 
Revised Papers, Lecture Notes in Computer Science, vol. 5382, pp. 200–225. Springer Verlag
30. Benveniste, A., Caillaud, B., Passerone, R.: A generic model of contracts for embedded sys-
tems. Rapport de recherche 6214, Institut National de Recherche en Informatique et en
Automatique (2007)
32. Benvenuti, L., Ferrari, A., Mangeruca, L., Muzzi, E., Passerone, R., Sofronis, C.: A contract-
based formalism for the specification of heterogeneous systems. In: Proceedings of the For-
um on Specification, Verification and Design Languages (FDL08), pp. 142–147. Stuttgart, 
Tampere, Finland (2008)
References

76. Douglass, B.: Real Time UML. Pearson Education (2009)
References 209


129. The Institute of Electrical and Electronics Engineers, Inc.: IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks (2011)


References