Moving from Specifications to Contracts in Component-Based Design*

Sebastian S. Bauer[1], Alexandre David[2], Rolf Hennicker[1], Kim Guldstrand Larsen[2], Axel Legay[2,3], Ulrik Nyman[2], and Andrzej Wąsowski[4]

[1] Ludwig-Maximilians-Universität München, Germany
[2] Computer Science Department, Aalborg University, Denmark
[3] INRIA/IRISA, Rennes Cedex, France
[4] IT University of Copenhagen, Denmark

Abstract. We study the relation between specifications of component behaviors and contracts providing means to specify assumptions on environments as well as component guarantees. We show how a contract framework can be built in a generic way on top of any specification theory which supports composition and specification refinement. Our contract framework lifts refinement to the level of contracts and proposes a notion of contract composition on the basis of dominating contracts. Contract composition satisfies a universal property and can be constructively defined if the underlying specification theory is complete, i.e. it offers operators for quotienting and conjoining specifications. We illustrate our generic construction of contracts by moving a specification theory for modal transition systems to contracts and we show that a (previously proposed) trace-based contract theory is an instance of our framework.

1 Introduction

Over the years we have seen a remarkable growth of complexity and size of software systems. This growth has been possible due to rapid development in hardware and software technology. Development of software today uses strong abstraction and encapsulation principles, that allows componentizing systems into comprehensible units.

This rapid growth of size and complexity of systems has inspired intensive research into component-oriented design and analysis methods for software. In the domain of safety critical concurrent software a number of interface theories have been proposed to this end, starting with the seminal work of Alfaro and Henzinger [2] devoted to tracking communication errors in discrete systems, followed by numerous extensions addressing other errors, or other forms of abstraction [1,16,17]. These include abstract specification of discrete finite-state systems exploiting may/must modalities [18,20,21,23,26,33,34,36], specification

* Work partially supported by MT-LAB (a VKR Centre of Excellence), by an "Action de Recherche Collaborative" ARC (TP)I, and by the EU project ASCENS, 257414.

J. de Lara and A. Zisman (Eds.): FASE 2012, LNCS 7212, pp. 43–58, 2012.

of systems manipulating complex data [4,8,35], specification of real-time embedded systems and real-time communication protocols [3,11,14,24], specification of randomized and probabilistic systems [12], and modeling of resource usage [6,13]. This proliferation of results is both positive and negative. Positive since it is a sign of fast progress in the field. Negative, because many works appear similar, yet it is difficult to compare them.

We attempt to develop a synthesis of the existing work in a uniform common framework. Altogether these theories have led to a shared understanding of what are the main ingredients of a mature specification theory for behavioral components; namely notions of satisfaction and refinement, together with composition operators such as conjunction, parallel compositions, and quotients. Nevertheless, despite this agreement, and despite the algebraic similarity of many specification theories, no uniform meta-theory exists that would formalize the abstract structure to enable better comparability of work, and reuse of results, channeling proliferation into higher quality and impact.

Independently, a number of contract theories, based on assume-guarantee (AG) reasoning have been developed, with a similar aim of approaching the compositional design. Contract theories differ from specification theories in that they strictly follow the principle of separation of concerns. They separate the specification of assumptions from specification of guarantees, a choice largely inspired by early ideas on manual proof methods of Misra, Chandy [30] and Jones [22], along with the wide acceptance to pre-/post-condition style of specification in programming [29]. Contract theories exist for discrete systems [10,25,31] and probabilistic systems [15,37].

Even though the specification theory, and the contract theory research have similar objectives, it is not clear so far what the two approaches offer with respect to each other, and whether their development is making the others complementary, or superfluous. So our second goal is to understand not only the essential structure of specification theories, but also their relation to contract theories. All in all we set off to organize (somewhat) the field of compositional specification for behavioral components.

We define contracts as pairs, (A, G), where A is a specification of assumptions, and G is a specification of guarantees. This leads us to our hypothesis that most specification theories should have enough structure to be used as a basis of an associated contract theory with explicit assumptions and guarantees. Dually, we observe that contract theories tend to degenerate to specification theories in the following simple manner: a specification G is describing the same models as a contract (tt, G) — so a contract without any assumption. Thus any reasonably complete contract theory can be used as a specification theory.

We make this intuition formal by developing a meta-theory of specifications and contracts. First, in Sect. 2, we propose a simple and general axiomatization of specification theories, able to capture the algebraic structure of most of the current specification theories (some frameworks require small adaptation, because not all of them have been originally developed with a complete set of operators in mind). Second, we demonstrate in Sect. 3 how a contract framework

can be derived from a specification theory, using our abstract constructions. As a result we are able to instantiate "for free" a contract theory with good properties of contracts from any specification theory fulfilling our axioms.

Any such derived contract theory is automatically equipped with:

- An implementation and an environment semantics reflecting the set of interfaces and environments that satisfy the guarantees and assumptions of the contract, respectively.
- A refinement relation that allows to compare contracts in terms of sets of implementations and legal environments.
- A structural composition, which encapsulates contracts for two communicating components into one contract for the composition of the two.

These results follow automatically as soon as the specification theory is equipped with parallel composition, conjunction, and a quotient of parallel composition. A number of specification theories have been proposed recently that satisfy our assumptions. In the course of this paper, we illustrate our general constructions by moving two specification theories to two contract theories: a simple trace-based specification theory, in which specifications are represented as sets of runs or traces (inspired by Benveniste et al. [10]), and as a more detailed example, we use modal specifications [32] in Sect. 4 to derive so-called modal contracts. All proofs can be found in [5].

We would like to stress that there are many other specification theories that fit into our framework, for instance, timed specifications [11], which allow us to derive "for free" a contract theory for timed systems, which has not yet been proposed in the literature.

2 Specification Theories

In our study the abstract concept of a specification theory defines rudimentary properties that should be satisfied by any formal framework for component behavior specifications. Given a class $\mathscr{S}$ of specifications, a specification theory includes a composition operator $\otimes$ to combine specifications to larger ones.[1] Additionally, a specification theory must offer a refinement relation $\leq$ to relate "concrete" and "abstract" specifications, i.e. $S \leq T$ means that S refines T. To obtain a specification theory, refinement must be compositional in the sense that it must be preserved by the composition operator.

Formally, a *specification theory* is a triple $(\mathscr{S}, \otimes, \leq)$ consisting of a class $\mathscr{S}$ of specifications, a parallel composition operator $\otimes : \mathscr{S} \times \mathscr{S} \to \mathscr{S}$ and a reflexive and transitive refinement relation $\leq \subseteq \mathscr{S} \times \mathscr{S}$, such that for all $S, S', T, T' \in \mathscr{S}$,

$$\text{whenever } S' \leq S \text{ and } T' \leq T, \text{ then } S' \otimes T' \leq S \otimes T. \tag{A1}$$

[1] The composition operator is, in general, partial since it is not always syntactically meaningful to compose specifications, due to syntactic constraints. In this work, however, to avoid a lot of technicalities, we will restrict ourselves to total composition operators – though the theory is easily extendable to partial composition operators.

The refinement relation induces an equivalence relation $=$ on specifications, by $S = T$ if and only if $S \leq T$ and $T \leq S$. The composition operator is commutative and associative with respect to this equivalence relation.

Obviously, in a top-down design, the requirements for a specification theory support independent development of components. To a certain extent a specification theory supports also bottom-up design, where existing components can be reused as parts of a larger system architecture, as long as local refinements are correct and local specifications fit into the context.

Specification theories sometimes come along with an operator $/$, called *quotient*, which is dual to parallel composition and which allows to synthesize specifications: When given a requirement specification T of the overall system and a smaller specification S, then the quotient T/S is the most general specification such that $S \otimes (T/S) \leq T$. Formally, quotient is a partial operator $/ : \mathscr{S} \times \mathscr{S} \hookrightarrow \mathscr{S}$ that satisfies

$$T/S \text{ defined if and only if } \exists X \in \mathscr{S} : S \otimes X \leq T. \tag{A2}$$

$$\text{If } T/S \text{ defined, then } S \otimes (T/S) \leq T. \tag{A3}$$

$$\text{If } T/S \text{ defined, then } \forall X \in \mathscr{S} : S \otimes X \leq T \implies X \leq T/S. \tag{A4}$$

When two separate teams independently develop specifications that are intended to be realized by the same component, then it is useful to have a *conjunction* operator $\wedge$ that computes the most general specification that realizes both specifications (if this is possible). Formally, conjunction is a partial operator $\wedge : \mathscr{S} \times \mathscr{S} \hookrightarrow \mathscr{S}$ such that

$$S \wedge T \text{ defined if and only if } \exists X \in \mathscr{S} : X \leq S \text{ and } X \leq T. \tag{A5}$$

$$\text{If } S \wedge T \text{ defined, then } S \wedge T \leq S \text{ and } S \wedge T \leq T. \tag{A6}$$

$$\text{If } S \wedge T \text{ defined, then } \forall X \in \mathscr{S} : X \leq S \text{ and } X \leq T \implies X \leq S \wedge T. \tag{A7}$$

When a specification theory supports quotient as well as conjunction, then we call it a *complete* specification theory.

Example 1. *As our running example we revisit the contract framework of Benveniste et al. [10], for two reasons: first, it uses a simple trace-based language to represent behavior of components, and specification operators boil down to simple set operations which we believe helps to understand the abstract requirements of specification theories; second, we will show that in fact our general constructions applied to this trace-based specification theory exactly results in the contract framework (in a simplified version) described in [10].*

In this simple theory, a global set $\mathbb{P}$ of ports is assumed over which components can communicate by reading and writing port values. The class of specifications consists of all (possibly empty) subsets of $\mathcal{R}(\mathbb{P})$ which is the set of all runs over $\mathbb{P}$ where each run assigns a history of values to the ports in $\mathbb{P}$. For example, a run could be a function $\rho : \mathbb{R}_{\geq 0} \to (\mathbb{P} \to \mathbb{V})$ from the time domain $\mathbb{R}_{\geq 0}$ to a valuation $\mathbb{P} \to \mathbb{V}$ of the ports, for some value set $\mathbb{V}$.

In this setting, refinement is simply defined by set inclusion, composition and conjunction is intersection (they are the same since we are dealing with a single

global signature). Note that conjunction is total, as the empty set is also a specification. For any two specifications T and S, the dual operation to composition, quotient, is defined by $T/S =_{def} T \cup \neg S$, where $\neg A =_{def} \mathcal{R}(\mathbb{P}) \setminus A$. Notice that indeed quotient is the maximal specification X such that S composed with X refines T, i.e. $S \cap X \subseteq T$.

In the following we will see that if we apply the general constructions of our contract framework to the trace-based case we will obtain the contract framework of Benveniste et al. [10].

3 Building a Contract Framework

For the development of our abstract contract framework, we assume to be given a specification theory $(\mathscr{S}, \otimes, \leq)$ as defined in the previous section.

3.1 Contracts and Their Semantics

On top of the specification theory we define a notion of a contract which explicitly distinguishes between assumption and guarantee specifications.

Definition 1. *A* contract *is a pair (A, G) where $A, G \in \mathscr{S}$ are two specifications.*

In a contract (A, G), the specification A expresses the assumption on the environment of the component, whereas the specification G describes the guarantee of any component implementation to the environment given that the environment respects the assumption A. For the definition of implementation correctness, we use a notion of *relativized refinement* which is derived from the refinement relation of the underlying specification theory.

Definition 2. *Relativized refinement is the ternary relation in $\mathscr{S} \times \mathscr{S} \times \mathscr{S}$ defined as follows: for all $S, E, T \in \mathscr{S}$,*

$$S \leq_E T \quad \textit{if and only if} \quad \forall E' \in \mathscr{S} : E' \leq E \implies S \otimes E' \leq T \otimes E'.$$

$S \leq_E T$ intuitively means that S refines T if both are put in any context E' that refines E; in particular, $S \otimes E \leq T \otimes E$. The following lemma summarizes properties of relativized refinement that are easy consequences of the definition.

Lemma 1. *Relativized refinement is a preorder, and for all $S, E, E', T \in \mathscr{S}$, whenever $S \leq_E T$ and $E' \leq E$ then $S \leq_{E'} T$.*

The *implementation semantics* of a contract (A, G) is given by the set of all specifications that satisfy the contract guarantee G *under the assumption* A:

$$[\![C]\!]_{\text{impl}} = \{I \in \mathscr{S} \mid I \leq_A G\}.$$

This is a significant generalization of pure specification theories where it is usually assumed that implementations must literally satisfy the specification. The

environment semantics of the contract (A, G) consists of all (environment) specifications for (or users of) the component satisfying the assumption A of the contract:

$$[\![C]\!]_{\text{env}} = \{E \in \mathscr{S} \mid E \leq A\}.$$

In summary, the semantics of a contract is given by both implementation semantics and environment semantics. Two contracts are *semantically equivalent*, if they have the same (implementation and environment) semantics.

Example 2. *In our trace-based example the relativized refinement $S \leq_E T$ can be easily shown to be equivalent to $S \cap E \subseteq T$; note that all specifications describe sets of runs over the same global set of ports $\mathbb{P}$.*

Our first result is a direct consequence of the definition of a contract and contract semantics: Whenever one has a correct environment and a correct implementation of a contract, then their composition is a refinement of the composition of assumption and guarantee of the contract.

Theorem 1. *Let $C = (A, G)$ be a contract. For all $E, I \in \mathscr{S}$, if $E \in [\![C]\!]_{\text{env}}$ and $I \in [\![C]\!]_{\text{impl}}$ then $E \otimes I \leq A \otimes G$.*

The implementation semantics of a contract in general depends on both the assumption A and the guarantee G. However, if the implementation semantics of (A, G) is independent of the assumption A, we say that the contract (A, G) is in normal form.

Definition 3. *A contract $C = (A, G)$ is* in normal form *if for all specifications $I \in \mathscr{S}$, $I \leq_A G$ if and only if $I \leq G$.*

It may be the case that a contract (A, G) can by transformed into a semantically equivalent contract (A, G^{nf}) in normal form by weakening of G to G^{nf}. In the examples considered here the underlying specification theories are powerful enough to allow such a weakening for any contract (A, G).

Example 3. *For a contract (A, G) in our trace-based example, a semantically equivalent contract in normal form (see [10]) is given by $(A, G \cup \neg A)$. It is indeed in normal form according to our definition since for any specification I, $I \cap A \subseteq G$ if and only if $I \subseteq G \cup \neg A$.*

3.2 Refinement of Contracts

Next, we turn to the question how contracts can be refined. We follow here a standard approach inspired by notions of behavioral subtyping [28] that a contract C' refines another contract C if C' admits less implementations than C, but more legal environments than C.

Definition 4. *Let C and C' be two contracts. The contract C' refines the contract C (is stronger than C), written $C' \preceq C$, if $[\![C']\!]_{\text{impl}} \subseteq [\![C]\!]_{\text{impl}}$ and $[\![C']\!]_{\text{env}} \supseteq [\![C]\!]_{\text{env}}$.*

The refinement relation between contracts is reflexive and transitive. Obviously, two contracts C, C' are semantically equivalent if and only if $C' \preceq C$ and $C \preceq C'$. The following theorem characterizes contract refinement by contra-/covariant (relativized) refinement of corresponding assumptions and guarantees.

Theorem 2. *Let (A, G) and (A', G') be two contracts. Then $(A', G') \preceq (A, G)$ if and only if $A \leq A'$ and $G' \leq_A G$.*

An immediate consequence is that whenever two contracts (A, G), (A', G') are in normal form, then $(A', G') \preceq (A, G)$ if and only if $A \leq A'$ and $G' \leq G$.

Example 4. *Refinement of contracts (A, G) by (A', G') is called dominance in [10] (not to be mixed up with our notion of dominance later on), and is defined by $A \subseteq A'$ and $G' \subseteq G$ which matches our definition of contract refinement if contracts are in normal form. For the other cases we have achieved a more thorough (weaker) definition of refinement which we would suggest to use for the trace-based approach as well.*

3.3 Composition of Contracts

When implementations I_1 and I_2 of individual components are composed, their composition is only semantically meaningful if the contracts, say C_1, C_2, of the single components fit together. This mean that there exists a 'larger' contract C which subsumes C_1 and C_2 such that (1) the composition $I_1 \otimes I_2$ is a correct implementation of C, and (2) each correct environment of C controls the single implementations in such a way that they mutually satisfy the assumptions of the single contracts. Inspired by [31] we call such a contract C a dominating contract for C_1 and C_2.

Definition 5. *Let C, C_1, and C_2 be contracts.* C dominates C_1 and C_2 *if the following two conditions are satisfied:*

1. *Any composition of correct implementations of C_1 and C_2 results in a correct implementation of the contract C:*
   - $\forall I_1 \in [\![C_1]\!]_{\text{impl}} : \forall I_2 \in [\![C_2]\!]_{\text{impl}} : I_1 \otimes I_2 \in [\![C]\!]_{\text{impl}}$
2. *For any correct environment of the contract C_1, the composition with a correct implementation of the C_1 (C_2) results in a correct environment of C_2 (C_1). Formally, for all $E \in [\![C]\!]_{\text{env}}$,*
   - $\forall I_1 \in [\![C_1]\!]_{\text{impl}} : E \otimes I_1 \in [\![C_2]\!]_{\text{env}}$,
   - $\forall I_2 \in [\![C_2]\!]_{\text{impl}} : E \otimes I_2 \in [\![C_1]\!]_{\text{env}}$.

We say that two contracts C_1, C_2 are dominatible *if there exists a contract C dominating C_1, C_2.*

A composition of two contracts C_1 and C_2 is a strongest dominating contract for C_1 and C_2.

Definition 6. *A contract C is called* contract composition *of the contracts C_1 and C_2 if*

1. *C dominates C_1 and C_2,*
2. *for all contracts C', if C' dominates C_1 and C_2 then $C \preceq C'$.*

Contract compositions, if they exist, are unique up to semantic equivalence of contracts. We will now turn to the questions (1) whether two contracts are dominatible and (2) whether the composition of two contracts exists and, if so, whether it can be *constructively* defined. For this purpose we generally assume in the following that any contract has a normal form, i.e. for any $C = (A, G)$ there exists a semantically equivalent contract $C^{nf} = (A^{nf}, G^{nf})$ which is in normal form. Due to the definition of environment semantics, without loss of generality, we can always assume in the following that $A^{nf} = A$.

We consider first question (1), for which the following lemma is useful. It follows directly from the definition of a dominating contract.

Lemma 2. *Two contracts C_1 and C_2 are dominatible if and only if their normal forms C_1^{nf} and C_2^{nf} are dominatible.*

The next theorem provides a characterization of dominatability. The idea is that there must be an environment under which implementations of the single contracts can be adapted to meet each others assumptions.

Theorem 3. *Let $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ be two contracts with normal forms $C_1^{nf} = (A_1, G_1^{nf})$ and $C_2^{nf} = (A_2, G_2^{nf})$ respectively. C_1 and C_2 are dominatible if and only if $\exists E \in \mathscr{S} : G_1^{nf} \otimes E \leq A_2$ and $G_2^{nf} \otimes E \leq A_1$.*

We now turn to question (2) from above. For this we assume from now on a complete specification theory (recall that such a theory has quotient and conjunction) over which contracts are constructed.

Definition 7. *Let $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ be two contracts with normal forms $C_1^{nf} = (A_1, G_1^{nf})$ and $C_2^{nf} = (A_2, G_2^{nf})$ respectively. $C_1 \boxtimes C_2$ is defined if and only if C_1 and C_2 are dominatible and then*

$$C_1 \boxtimes C_2 =_{def} ((A_1/G_2^{nf}) \wedge (A_2/G_1^{nf}), G_1^{nf} \otimes G_2^{nf}).$$

Note that, by Lemma 2, $C_1 \boxtimes C_2$ is semantically equivalent to $C_1^{nf} \boxtimes C_2^{nf}$. The next lemma shows that $C_1 \boxtimes C_2$ is indeed well-defined.

Lemma 3. *Let C_1 and C_2 be two contracts with normal forms as in Def. 7. $(A_1/G_2^{nf}) \wedge (A_2/G_1^{nf})$ is defined if and only if $\exists E \in \mathscr{S} : G_1^{nf} \otimes E \leq A_2$ and $G_2^{nf} \otimes E \leq A_1$, if and only if C_1 and C_2 are dominatible.*

The next theorem answers question (2) from above.

Theorem 4. *If the contracts C_1 and C_2 are dominatible, then $C_1 \boxtimes C_2$ is (up to semantic equivalence) the composition of C_1 and C_2.*

The next statements deal with the relationship between contract composition and contract refinement. First, dominance is preserved under refinement of individual contracts.

Theorem 5. *Let C_1, C_1', C_2, C_2', C be contracts such that $C_1' \preceq C_1$ and $C_2' \preceq C_2$. If C dominates C_1 and C_2, then C dominates also C_1' and C_2'.*

Second, contract refinement is preserved under contract composition, thus our contract framework satisfies itself the requirements of a specification theory of Sect. 2 if we admit *partial* composition (which has been disregarded in Sect. 2 just for reasons of simplicity).

Theorem 6. *Let C_1, C_2, D_1, D_2 be contracts such that C_1 and C_2 are dominatible. If $D_1 \preceq C_1$ and $D_2 \preceq C_2$ then $D_1^{nf} \boxtimes D_2^{nf} \preceq C_1^{nf} \boxtimes C_2^{nf}$.*

Example 5. *In [10], contract composition is defined by*

$$(A_1, G_1) \boxtimes (A_2, G_2) = ((A_1 \cap A_2) \cup \neg(G_1 \cap G_2), G_1 \cap G_2).$$

Note that the assumption can be reformulated to $(A_1 \cup \neg G_1 \cup \neg G_2) \cap (A_2 \cup \neg G_1 \cup \neg G_2)$, and since the contracts (A_1, G_1) and (A_2, G_2) are in normal form we have $A_1 \cup \neg G_1 = A_1$ and $A_2 \cup \neg G_2 = A_2$. Hence we get $(A_1 \cup \neg G_2) \cap (A_2 \cup \neg G_1)$ as assumption which, all in all, exactly matches our definition of $\boxtimes$ for contracts.

4 Modal Contracts

To illustrate our general constructions for moving from a specification theory to contracts, we consider a well-established specification theory based on modal transition systems that has gained considerable interest in recent years, as it nicely supports loose specifications together with stepwise refinement. Modal transition systems [27] are labeled transition systems with two types of transition relations: *may* transitions model optional (allowed) behavior that need not be implemented in a refinement, and *must* transitions model required behavior. In [32] a complete specification theory for modal specifications (which correspond to deterministic modal transition systems) has been defined, which allows us to get *modal contracts* for free. Modal contracts have been defined already in [19,31] and we will comment on the differences in the next section.

We briefly sketch the specification theory for modal specifications, for a thorough introduction see [32]. A modal specification (MS) is formally defined as a tuple $S = (St, s_0, \Sigma, \dashrightarrow, \longrightarrow)$ where St is the set of states, $s_0 \in St$ is the initial state, Σ is the set of actions, and $\dashrightarrow, \longrightarrow \subseteq St \times \Sigma \times St$ are the may and must transition relation, respectively, such that $\longrightarrow \subseteq \dashrightarrow$. Any MS is required to be deterministic: for all states $s, s', s'' \in St$ and all actions $\alpha \in \Sigma$, if $(s, \alpha, s'), (s, \alpha, s'') \in \dashrightarrow$ then $s' = s''$. In the following, we usually write $s \overset{\alpha}{\dashrightarrow} s'$ for $(s, \alpha, s') \in \dashrightarrow$, and similarly for must transitions.

We consider a simple component-based system consisting of two components: component *Server* with contract (A_{Server}, G_{Server}) over the action set $\Sigma_{Server} = \{msg, secret\_msg, auth, send\}$ (i.e. both A_{Server}, G_{Server} have the set of actions Σ_{Server}), and a component *User* with contract (A_{User}, G_{User}) over set of actions $\Sigma_{User} = \{auth, send\}$. The two contracts can be seen in Fig. 1(a)–(d). May transitions are drawn with dashed arrows, and must transitions with solid arrows. May transitions underlying must transitions are not drawn for simplicity.

The contract (A_{Server}, G_{Server}) intuitively expresses the following protocol: First, the environment can issue a message (msg) that is then sent by the server

to the user (*send*). Second, the environment can also issue a secret message (*secret_msg*), that is only sent to the user if the server receives an authentication code from the user (*auth*). More precisely, the assumption A_{Server} formulates the following requirements on the environment:

- The authentication code may always be received.
- New messages (secret or not) are only allowed to be sent in the initial state.
- Once a message is received, the environment must be ready to accept the sending of the server.
- Once a secret message is received, the authentication code must be received.

The contract (A_{User}, G_{User}) for the user component is simpler: The guarantee is that the messages can always be received from the server, however, the sending of the authentication code may not be possible. The assumption A_{User} always allows the actions *auth* and *send*, without any specific order.

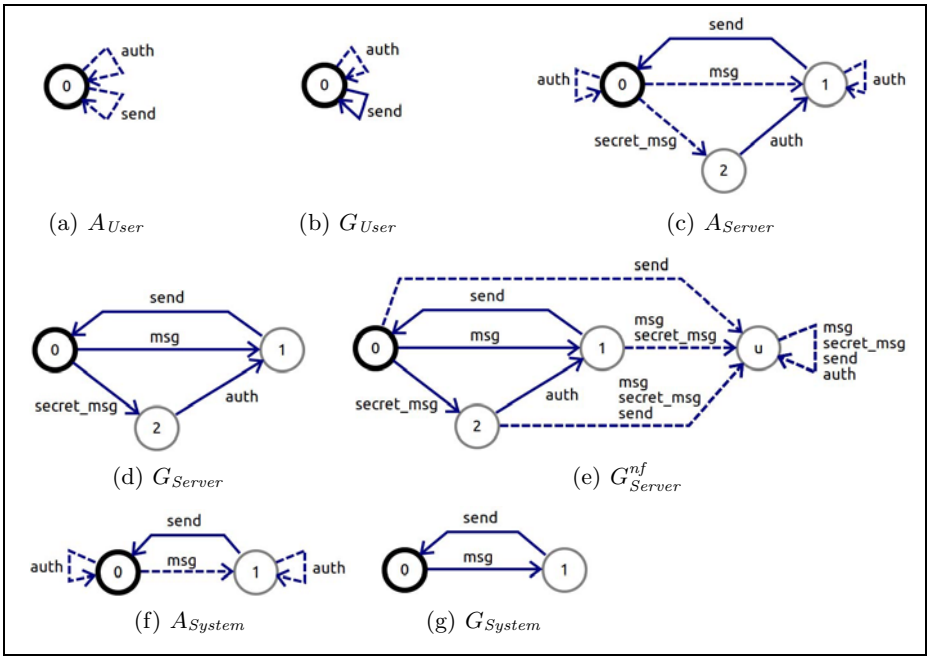

Fig. 1. Modal contracts for a simple message system

Before we discuss how these two modal contracts are composed, we first have to discuss the underlying specification theory, so refinement together with all the specification operators for MS. Refinement of MS is defined as follows: an MS S refines another MS T, written $S \leq_m T$, if they have the same set of actions Σ and if there exists a relation $R \subseteq St_S \times St_T$ such that $(s_0, t_0) \in R$ and for

all $(s,t) \in R$ and all $\alpha \in \Sigma$, whenever $s \overset{\alpha}{\dashrightarrow} s'$ then there exists $t \overset{\alpha}{\dashrightarrow} t'$ and $(s',t') \in R$, and whenever $t \overset{\alpha}{\longrightarrow} t'$ then there exists $s \overset{\alpha}{\longrightarrow} s'$ and $(s',t') \in R$. For instance, in Fig. 1, G_{User} is a refinement of A_{User}, i.e. $G_{User} \leq_m A_{User}$.

$\otimes$	$s_2 \overset{\alpha}{\longrightarrow} s_2'$	$s_2 \overset{\alpha}{\dashrightarrow} s_2'$
$s_1 \overset{\alpha}{\longrightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\longrightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} (s_1',s_2')$
$s_1 \overset{\alpha}{\dashrightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} (s_1',s_2')$

$/$	$s_2 \overset{\alpha}{\longrightarrow} s_2'$	$s_2 \overset{\alpha}{\dashrightarrow} s_2',\ s_2 \not\overset{\alpha}{\longrightarrow}$	$s_2 \not\overset{\alpha}{\dashrightarrow}$
$s_1 \overset{\alpha}{\longrightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\longrightarrow} (s_1',s_2')$	$(s_1,s_2) \in \lightning$	$(s_1,s_2) \in \lightning$
$s_1 \overset{\alpha}{\dashrightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} \mathsf{u}$
$s_1 \not\overset{\alpha}{\dashrightarrow}$			$(s_1,s_2) \overset{\alpha}{\dashrightarrow} \mathsf{u}$

$\wedge$	$s_2 \overset{\alpha}{\longrightarrow} s_2'$	$s_2 \overset{\alpha}{\dashrightarrow} s_2'$	$s_2 \not\overset{\alpha}{\dashrightarrow}$
$s_1 \overset{\alpha}{\longrightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\longrightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\longrightarrow} (s_1',s_2')$	$(s_1,s_2) \in \lightning$
$s_1 \overset{\alpha}{\dashrightarrow} s_1'$	$(s_1,s_2) \overset{\alpha}{\longrightarrow} (s_1',s_2')$	$(s_1,s_2) \overset{\alpha}{\dashrightarrow} (s_1',s_2')$	
$s_1 \not\overset{\alpha}{\dashrightarrow}$	$(s_1,s_2) \in \lightning$		

Fig. 2. Transition relations for the specification operators $\otimes$, $/$, $\wedge$ on MS

The specification operators composition, quotient and conjunction are described hereafter and we assume that the involved MS always have the same set of actions. Composition of MS ($\otimes$) is defined by synchronizing on shared actions. The rules of $\otimes$ for MS can be seen in Fig. 2; note that only the synchronization of two must transition yields a must transition, in all other cases it yields a may transition.[2]

The two missing operators quotient and conjunction need some more involved definition, because both are partial operators. During quotient as well as conjunction, inconsistencies may arise, however, that does not mean that the whole result is inconsistent; we instead apply a pruning operator ρ to remove all those inconsistent states. More precisely, given an MS S with a set of inconsistent states $\lightning \subseteq St$, the pruned version $\rho(S)$ gives the largest MS which refines S but no state of $\rho(S)$ is related (in the sense of refinement) to an inconsistent state in $\lightning$. The formal definition of pruning can be found in [32].

With pruning at hand, we can define quotient S_1/S_2 (as the dual operator to composition) and conjunction $S_1 \wedge S_2$, as shown in Fig. 2. The set $\lightning$ models in both cases the set of inconsistent states, and in the definition of quotient, the state u is a new *universal* state in which, for every action, there is a looping may transition to the same state u.

[2] The notation $s \not\overset{\alpha}{\dashrightarrow}$ means that there does not exist s' such that $s \overset{\alpha}{\dashrightarrow} s'$, and similar for must transitions.

For writing down contracts based on MS, it is useful to be able to handle dissimilar set of actions when applying specification operators, see [32]. Given an MS S over the set of actions Σ, and a larger set of actions $\Sigma' \supseteq \Sigma$,

- the strong extension of S, written $S_{\uparrow\Sigma'}$, adds for each new action $a \in \Sigma' \setminus \Sigma$ a may and a must loop with that action to all states in S.
- Similar, the weak extension of S, written $S_{\Uparrow\Sigma'}$, adds for each new action (only) a may loop (for all new actions) to all states.

The specification operators are, for the general case, extended to MS with dissimilar sets of actions as follows. If S and T are two MS with sets of actions Σ_S and Σ_T, respectively, and $\Sigma = \Sigma_S \cup \Sigma_T$, then $S \otimes T$ is defined by $S_{\uparrow\Sigma} \otimes T_{\uparrow\Sigma}$, $S \wedge T$ is defined by $S_{\Uparrow\Sigma} \wedge T_{\Uparrow\Sigma}$, and T/S is defined by $T_{\Uparrow\Sigma}/S_{\uparrow\Sigma}$.

Relativized refinement (see Def. 2), induced by modal refinement, can be shown to be equivalent with the following direct definition: If S, T, E are MS over the same set of actions Σ, then $S \leq_E T$ *if and only if* there exists a relation $R \subseteq St_S \times St_E \times St_T$ such that $(s_0, e_0, t_0) \in R$, and for all $(s, e, t) \in R$, all $\alpha \in \Sigma$,

1. if $s \overset{\alpha}{\dashrightarrow} s'$ and $e \overset{\alpha}{\dashrightarrow} e'$ then there exists $t \overset{\alpha}{\dashrightarrow} t'$ such that $(s', e', t') \in R$,
2. if $t \overset{\alpha}{\longrightarrow} t'$ and $e \overset{\alpha}{\dashrightarrow} e'$ then there exists $s \overset{\alpha}{\longrightarrow} s'$ such that $(s', e', t') \in R$.

Every modal contract can be transformed to an equivalent contract in normal form, by weakening the guarantee by the assumption. It turns out that there is a direct definition of a so-called *weakening operator*, that exactly does what we are looking for: $I \leq_A G$ if and only if $I \leq A \triangleright G$, where $A \triangleright G$ is the weakening of G by A. Formally, if A and G are two MS over the same set of actions Σ, then $A \triangleright G$ is defined to be the MS $((St_A \times St_G) \cup \{\mathsf{u}\}, (a_0, g_0), \Sigma, \dashrightarrow, \longrightarrow)$ where u is a fresh state (the universal state), and where the transition relations are defined as shown in the table in Fig. 3.

$\triangleright$	$g \overset{\alpha}{\longrightarrow} g'$	$g \overset{\alpha}{\dashrightarrow} g'$	$g \not\overset{\alpha}{\dashrightarrow}$
$a \overset{\alpha}{\dashrightarrow} a'$	$(a, g) \overset{\alpha}{\longrightarrow} (a', g')$	$(a, g) \overset{\alpha}{\dashrightarrow} (a', g')$	
$a \not\overset{\alpha}{\dashrightarrow}$	$(a, g) \overset{\alpha}{\dashrightarrow} \mathsf{u}$	$(a, g) \overset{\alpha}{\dashrightarrow} \mathsf{u}$	$(a, g) \overset{\alpha}{\dashrightarrow} \mathsf{u}$

Fig. 3. Rules for weakening ($\triangleright$)

Coming back to the example, the contract (A_{Server}, G_{Server}) is obviously not in normal form, but with the weakening operator at hand, we can transform the contract to the semantically equivalent contract $(A_{Server}, G^{nf}_{Server})$ where $G^{nf}_{Server} =_{def} A_{Server} \triangleright G_{Server}$, see Fig. 1(e). As one can see, the normalized contract has lots of additional transitions, and it is often better to draw non-normal form contracts which are usually considerably smaller. Note that (A_{User}, G_{User}) is already in normal form.

We can finally compose our two contracts. As the watchful reader might have already noticed, A_{Server} is expecting the user to answer in any case with the authentication code once a secret message is received. But G_{User} does not provide the authentication code because it may be the case that he/she is not aware of the code. Thus we have an inconsistency arising here, and as a result of applying quotient and conjunction while building the new (weakest) assumption $A_{System} = (A_{Server}/G_{User}) \wedge (A_{User}/G^{nf}_{Server})$, one can see in Fig. 1 that – as expected – the environment is not allowed to issue a secret message anymore. The resulting guarantee G_{System} of the composed contract has been slightly simplified by leaving out some may transitions to a universal state (as in G_{Server}) but the overall contract (A_{System}, G_{System}) is obviously semantically equivalent to $(A_{System}, G^{nf}_{Server} \otimes G_{User})$.[3] Our theory in Sect. 3 now tells us that (A_{System}, G_{System}) is indeed the strongest contract that dominates the contract of the server and the user.

5 Conclusion, Related Work, and Future Work

This paper studies the relationship between specifications of component behaviors and contracts. The general framework for contracts is inspired by the work of Benveniste et al. [10]. They have chosen a trace-based approach to represent interfaces which (as we have shown) is a specification theory and an instance of our proposed abstract contract framework. The idea to equip a specification with implementation and environment semantics has been used by the authors already in [7] where UML protocol state machines were considered as specifications of component interfaces.

Modal contracts have already been introduced and investigated in several previous works, including [19,31]. Raclet and Goessler [19] propose an implementation semantics that is slightly different to ours. In their paper, an implementation I satisfies a contract (A, G) if $A \wedge I \leq_m G$ whenever $A \wedge I$ is defined, which is in fact equivalent to our definition of contract satisfaction, but only for implementations (that are modal specifications where the must and may transition relations coincide). Our satisfaction relation is more powerful as it works for arbitrary modal specifications. Refinement and composition is only syntactically defined, without any semantic considerations as we do it in this paper, hence they lack the universal property for contract compositions. In [31], Quinton and Graf define an abstract framework of contracts which however tends to be technically overloaded due to the integration of complex composition operators. Besides this difference, the satisfaction relation of contracts is the same as in our work. Our notion of (semantic) dominance is inspired by their (syntactical) definition, but still their work lacks of a careful discussion about dominance and the universal property of contract composition. In summary, in comparison

[3] This "inverse" operation to normalizing contracts is in fact useful when drawing larger specifications, and can be automatically applied to a (composed) contract to reduce its number of states and transitions while remaining semantically equivalent.

to both works [19,31], we consider our work as "more semantical" as implementation and environment semantics of contracts are carefully taken into account for the definition of contracts and contract operators.

There are various directions for future work. As an example, we have simplified our setup in this work by ignoring compatibility and consistency issues between interfaces, although we are convinced that they can be integrated without problems. Another major objective is to implement our modal contract theory in the MIO Workbench [9].

References

1. Aarts, F., Vaandrager, F.: Learning I/O Automata. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 71–85. Springer, Heidelberg (2010)
2. de Alfaro, L., Henzinger, T.A.: Interface automata. In: FSE, pp. 109–120. ACM Press (2001)
3. de Alfaro, L., Henzinger, T.A., Stoelinga, M.I.A.: Timed Interfaces. In: Sangiovanni-Vincentelli, A.L., Sifakis, J. (eds.) EMSOFT 2002. LNCS, vol. 2491, pp. 108–122. Springer, Heidelberg (2002)
4. de Alfaro, L., da Silva, L.D., Faella, M., Legay, A., Roy, P., Sorea, M.: Sociable Interfaces. In: Gramlich, B. (ed.) FroCos 2005. LNCS (LNAI), vol. 3717, pp. 81–105. Springer, Heidelberg (2005)
5. Bauer, S.S., David, A., Hennicker, R., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Moving from specifications to contracts in component-based design. Tech. Rep. 1201, LMU Munich, Germany (January 2012)
6. Bauer, S.S., Fahrenberg, U., Juhl, L., Larsen, K.G., Legay, A., Thrane, C.R.: Quantitative Refinement for Weighted Modal Transition Systems. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 60–71. Springer, Heidelberg (2011)
7. Bauer, S.S., Hennicker, R.: Views on Behaviour Protocols and Their Semantic Foundation. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO 2009. LNCS, vol. 5728, pp. 367–382. Springer, Heidelberg (2009)
8. Bauer, S.S., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: A Modal Specification Theory for Components with Data. In: FACS 2011. LNCS. Springer, Heidelberg (2011)
9. Bauer, S.S., Mayer, P., Schroeder, A., Hennicker, R.: On Weak Modal Compatibility, Refinement, and the MIO Workbench. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 175–189. Springer, Heidelberg (2010)
10. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple Viewpoint Contract-Based Specification and Design. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2007. LNCS, vol. 5382, pp. 200–225. Springer, Heidelberg (2008)
11. Bertrand, N., Legay, A., Pinchinat, S., Raclet, J.-B.: A Compositional Approach on Modal Specifications for Timed Systems. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM 2009. LNCS, vol. 5885, pp. 679–697. Springer, Heidelberg (2009)
12. Caillaud, B., Delahaye, B., Larsen, K.G., Legay, A., Pedersen, M.L., Wasowski, A.: Constraint markov chains. Theor. Comput. Sci. 412(34), 4373–4404 (2011)
13. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Resource Interfaces. In: Alur, R., Lee, I. (eds.) EMSOFT 2003. LNCS, vol. 2855, pp. 117–133. Springer, Heidelberg (2003)

14. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed I/O automata: a complete specification theory for real-time systems. In: HSCC, pp. 91–100. ACM (2010)
15. Delahaye, B., Caillaud, B., Legay, A.: Probabilistic contracts: a compositional reasoning methodology for the design of systems with stochastic and/or non-deterministic aspects. Formal Methods in System Design 38(1), 1–32 (2011)
16. Doyen, L., Henzinger, T.A., Jobstman, B., Petrov, T.: Interface theories with component reuse. In: EMSOFT, pp. 79–88. ACM Press (2008)
17. Emmi, M., Giannakopoulou, D., Păsăreanu, C.S.: Assume-Guarantee Verification for Interface Automata. In: Cuellar, J., Sere, K. (eds.) FM 2008. LNCS, vol. 5014, pp. 116–131. Springer, Heidelberg (2008)
18. Godefroid, P., Jagadeesan, R.: On the Expressiveness of 3-Valued Models. In: Zuck, L.D., Attie, P.C., Cortesi, A., Mukhopadhyay, S. (eds.) VMCAI 2003. LNCS, vol. 2575, pp. 206–222. Springer, Heidelberg (2002)
19. Goessler, G., Raclet, J.B.: Modal contracts for component-based design. In: SEFM, pp. 295–303. IEEE Computer Society (2009)
20. Grumberg, O., Lange, M., Leucker, M., Shoham, S.: *Don't Know* in the μ-*Calculus*. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 233–249. Springer, Heidelberg (2005)
21. Huth, M., Jagadeesan, R., Schmidt, D.A.: Modal Transition Systems: A Foundation for Three-Valued Program Analysis. In: Sands, D. (ed.) ESOP 2001. LNCS, vol. 2028, pp. 155–169. Springer, Heidelberg (2001)
22. Jones, C.B.: Development methods for computer programs including a notion of interference. Ph.D. thesis, Oxford University Computing Laboratory (1981)
23. Larsen, K.G.: Modal Specifications. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407, pp. 232–246. Springer, Heidelberg (1990)
24. Larsen, K.G., Legay, A., Traonouez, L.-M., Wąsowski, A.: Robust Specification of Real Time Components. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 129–144. Springer, Heidelberg (2011)
25. Larsen, K.G., Nyman, U., Wąsowski, A.: Interface Input/Output Automata. In: Misra, J., Nipkow, T., Karakostas, G. (eds.) FM 2006. LNCS, vol. 4085, pp. 82–97. Springer, Heidelberg (2006)
26. Larsen, K.G., Nyman, U., Wąsowski, A.: Modal I/O Automata for Interface and Product Line Theories. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 64–79. Springer, Heidelberg (2007)
27. Larsen, K.G., Thomsen, B.: A modal process logic. In: LICS. IEEE Computer Society (1988)
28. Liskov, B., Wing, J.M.: A behavioral notion of subtyping. ACM Trans. Program. Lang. Syst. 16(6), 1811–1841 (1994)
29. Meyer, B.: Applying "design by contract". IEEE Computer 25(10), 40–51 (1992)
30. Misra, J., Chandy, K.M.: Proofs of networks of processes. IEEE Trans. Software Eng. 7(4), 417–426 (1981)
31. Quinton, S., Graf, S.: Contract-based verification of hierarchical systems of components. In: SEFM, pp. 377–381. IEEE Computer Society (2008)
32. Raclet, J.B., Badouel, E., Benveniste, A., Caillaud, B., Legay, A., Passerone, R.: A modal interface theory for component-based design. Fundam. Inform. 108(1-2), 119–149 (2011)
33. Raclet, J.B., Badouel, E., Benveniste, A., Caillaud, B., Passerone, R.: Why are modalities good for interface theories? In: ACSD, pp. 119–127. IEEE Computer Society (2009)

34. Sassolas, M., Chechik, M., Uchitel, S.: Exploring inconsistencies between modal transition systems. Software and System Modeling 10(1), 117–142 (2011)
35. Tripakis, S., Lickly, B., Henzinger, T.A., Lee, E.A.: A theory of synchronous relational interfaces. ACM Trans. Program. Lang. Syst. 33(4), 14 (2011)
36. Wei, O., Gurfinkel, A., Chechik, M.: Mixed Transition Systems Revisited. In: Jones, N.D., Müller-Olm, M. (eds.) VMCAI 2009. LNCS, vol. 5403, pp. 349–365. Springer, Heidelberg (2009)
37. Xu, D.N., Gössler, G., Girault, A.: Probabilistic Contracts for Component-Based Design. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 325–340. Springer, Heidelberg (2010)