Automata for Service Contracts *

Davide Basile, Pierpaolo Degano, and Gian-Luigi Ferrari

{basile,degano,giangi}@di.unipi.it
Dipartimento di Informatica, Università di Pisa, Italy

Abstract. A novel approach to the formal description of service contracts is presented in terms of automata. We focus on the basic security property of guaranteeing that in the multi-party composition of principals each individual gets his requests satisfied, so that the overall composition reaches its goal. Depending on whether requests are satisfied synchronous or asynchronous, we construct an orchestrator that at static time either yields composed services enjoying the required properties or detects the individuals responsible for possible violations. To do that we resort to techniques from Control Theory and Operational Research.

1 Introduction

In the Service Oriented Computing paradigm, and in the Cloud as well, services are offered, composed and invoked by different individuals, possibly malicious. Many security issues then arise, e.g. on the usage of resources, the guarantees on the results provided, the respect of the users' privacy, the correctness of the way services interact with the customer and among them. Indeed, APIs for service management, orchestration and monitoring are integral part of the security of service-based systems. For instance, business processes can exploit these APIs to dynamically include add-on services to manage the relationships between the business process and its environment. Hence, weak programming methodologies can expose an organization to several security threats pertaining to confidentiality, integrity, availability, and accountability. Also, the secure orchestration of services becomes more problematic in the age of cloud computing when organizations are dependent on the 24/7 availability of services obtained by assembling together a variety of heterogeneous services. Below, we study a model to describe and prescribe when a group of services will safely cooperate for solving a task.

Several proposals have been put forward in the literature, among which we only mention a few of those in the framework of the λ-calculus [7, 12, 6], of process calculi [16, 17, 1], of non-standard logics [10, 3]. In section 2 we provide a comparison between the mentioned proposals and our model. A basic notion common to these approaches is that of *contract*, i.e. the interfaces published by services. Of course, this notion differs under many aspects in the various proposals. For us, a contract specifies what a service is going to offer and what in turn it requires. Two or more services can then compose and

* This work has been partially supported by the MIUR project *Security Horizons* and IST-FP7-FET open-IP project *ASCENS*

cooperate to achieve a common goal. The composition however may fail, because the contracts involved do not agree. For example, my contract says that "I give you a glass of wine and then I wait to be paid 5€" — as a regular expression: $\overline{wine}.five$, where an over-lined action denotes an offer. Your contract instead says that "You will drink a glass of wine and then you either pay 5€ or 10¢" — as a regular expression $wine.(\overline{five} + \overline{ten})$. The composition of the two contracts succeeds if you pay me 5€, while it fails if you dislike my wine and give me 10¢, only. *Circular* contracts are a special case [5, 4]: they arise when all the participants start by asking something, so exposing themselves to the risk of not getting back anything, if a partner is dishonest. A typical example is: "Alice gives Bob a toy if Bob gives her a bike", and "Bob gives Alice a bike if she gives back a toy". Circular contracts have been studied within Propositional Contract Logic, an extension of intuitionistic logic, proposed in [10]. Some models for a significant fragment of Propositional Contract Logic have been studied, based on suitably extended process algebras [11, 8], Event Structures [5] and Petri Nets [4].

We represent contracts, possibly circular, as a special kind of finite state automata, called Contract Automata (CA), endowed with composition operators. The "hostelry" contract discussed above is depicted in Figure 1. More precisely, we assume to have offers (over-lined), and requests (represented as non over-lined offers, disjoint from them), just as in I/O [21] and Interface Automata [2]. Then, the alphabet of a CA consists of tuples, each element of which records the activity of a single participant in the contract. In a tuple there is either a single offer (or a single request), or there is a pair of request-offer that match in pair; all the other elements of the tuple contain the null symbol $-$, meaning that the corresponding individuals stay idle.

Contract Automata can be composed, by making the cartesian product of their states and of the labels of the joined transitions, with the additional possibility of labels recording matching request-offer — actually, we have two different operations of composition: we will come back on them later on. An additional requirement we pose on the execution of a contract, i.e. of the run of the corresponding composed automaton, is that each participant has to complete all its activities and reach a state, designed to be final. Since our automata can obviously have loops, we have then to suitably take care of the ways these loops are taken in executions.

Consider again the example of drinking wine above. A word, i.e. a sequence of interactions between the involved services, accepted by the compositions of the two contracts is $(\overline{wine}, wine)(five, \overline{five})$, while the following word is also accepted by the contract automaton but it is not in agreement (see below): $\sigma = (\overline{wine}, wine)(-, \overline{ten})(five, -)$. Note that the tuples contain enough information to single out why a word leads to failure. More importantly, we can detect which (sub-)contract or individual, i.e. which (sub-)service made a wrong move. In the second trace of the oversimplified example above, the customer does not pay the due price to the innkeeper, and is therefore liable.

Besides describing contracts as automata, our main results concern the definition of:

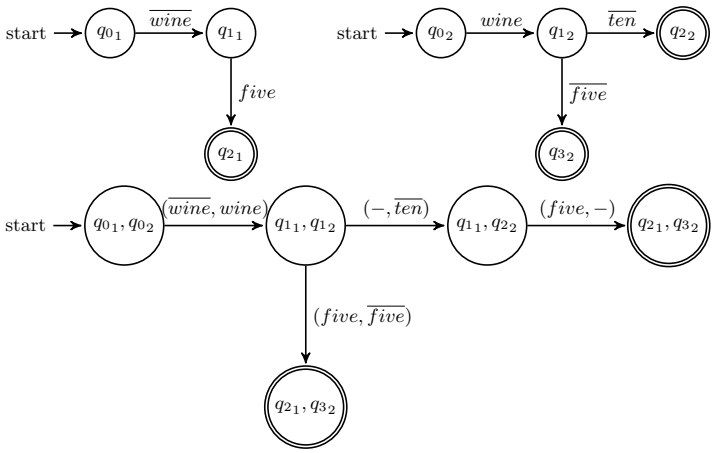

Fig. 1. On the first row from left to right we have $\mathcal{A}_1, \mathcal{A}_2$, on the second row $\mathcal{A}_3$

1. new, natural ways of composing contracts involving multiple parties, in accordance with two orchestration policies discussed below, and following the lines of component-based software engineering [21, 2];
2. properties of CA that guarantee a group of contracts to agree under all circumstances (*agreement* or, in a more liberal fashion *weak agreement*). We also define when they agree in some specific cases but not always (they *admit* agreement or weak agreement). The definition of these and other properties becomes here natural, because of our language based approach;
3. static techniques to detect which individual in a contract is *liable*, i.e. the one that is responsible for leading a composition to a failing, insecure state. While finding the individuals liable for a violation of agreement is not hard (inspecting the automaton plus a little calculation suffice), things become much more intricate when we consider weak agreement, that turns out to be context-sensitive, in language-theoretic terms. We attack here this last problem in a novel manner by resorting to optimization techniques borrowed from Operational Research;
4. an orchestrator that composes services so as to enforce the properties mentioned in item 2. For agreement we can define a (most permissive) controller, which is again a CA, in the style of Control Theory [14]. For weak agreement instead the controller is much harder to define.

Composition of contracts As anticipated, we have two ways of composing services. The first one, denoted by $\otimes$, considers the case when a service S joins a group of services already clustered as a single orchestrated service S'. In this case S can only accept the still available offers (requests, respectively) of S' and vice versa. In other words, there is no interaction of S with the individual components of the orchestration S', but only with S' as a whole. E.g., suppose that a thirsty fellow, gets in the inn, wants to drink and also offers to pay the other customer's

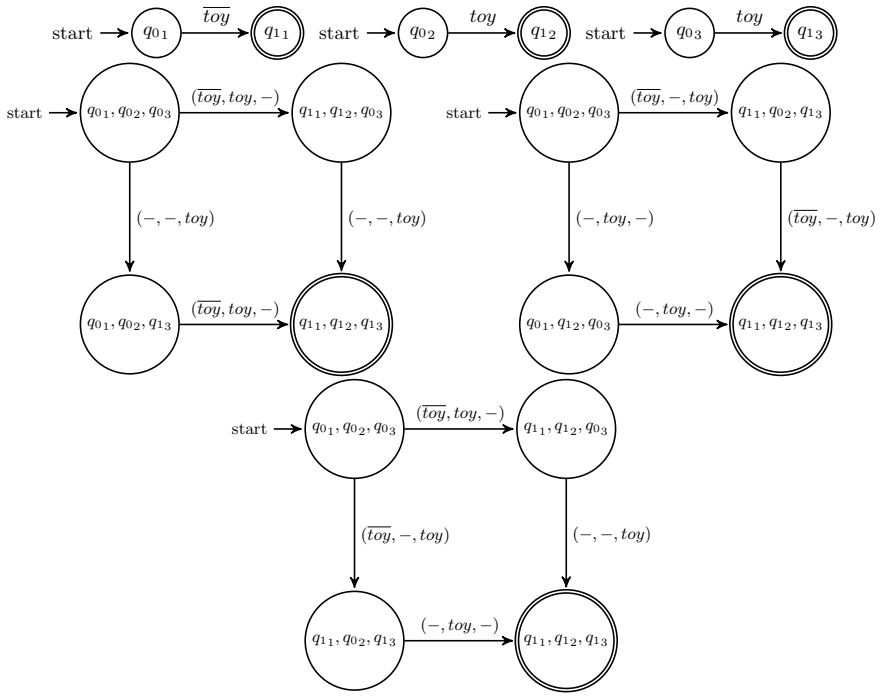

Fig. 2. From left to right, starting from the first row the contract automata of Alice, Bob and Jill, the contract automata $(Alice \otimes Bob) \otimes Jill$, $(Alice \otimes Jill) \otimes Bob$. On the third row the contract automaton $Alice \boxtimes Bob \boxtimes Jill$.

glass of wine: $wine.\overline{five}.\overline{five}$. The (automaton representing the) composition of the three parties will accept also the following word (compare it with the word σ above): $\sigma' = (\overline{wine}, wine, -)(-, \overline{ten}, -)(-, -, wine)(five, -, \overline{five})(-, -, \overline{five})$. Its intended meaning is that the first customer drinks, pays 10¢, the thirsty guy asks for wine and pays twice 5€ without getting anything — the innkeeper does not complain. The point is that the last customer has no means to break the business activity represented by the handshake $(\overline{wine}, wine)$ between the first one and the innkeeper.

This is not the case with the second operation of composition, $\boxtimes$, that puts instead the components of S at the same level of those of S'. Any matching request-offer $(-, ..., -, \overline{a}, ..., a, ..., -, ...)$ of either contracts can be split, and its offer $\overline{a}$ and request a are re-combined with a corresponding matching request or offer. Back to the example above, we will now have, e.g., the following sequence: $\sigma'' = (\overline{wine}, -, wine)(-, wine, -)(-, \overline{ten}, -)(five, -, \overline{five})(-, -, \overline{five})$, quenching the thirsty fellow and still making the innkeeper happy. We believe that this second operation of composition is essential to model business processes in dynamically changing environments.

Example 1. To clarify the two operations of composition we turn to consider a different example. Figure 2 illustrates a *children's playground contract* [11]. Alice, Bob and Jill bring their old toys and play a toy swap. Alice offers a toy while Bob and Jill perform the same request for the toy of Alice. In the product $(Alice \otimes Bob) \otimes Jill$ the toy is assigned to Bob who first enters in the composition with Alice, no matter if $Jill$ performs the same move. Equally, in the product $(Alice \otimes Jill) \otimes Bob$ the toy is assigned to Jill. In the last row we have the product $\mathcal{A}_1 \boxtimes (\mathcal{A}_2 \boxtimes \mathcal{A}_3) = (\mathcal{A}_1 \boxtimes \mathcal{A}_2) \boxtimes \mathcal{A}_3 = \mathcal{A}_1 \otimes \mathcal{A}_2 \otimes \mathcal{A}_3$ which represents dynamic orchestration: no matter who between Jill or Bob enters the composition with Alice first, the toy will be assigned to the first participant who makes the move.

Agreement and Weak Agreement We focus again on the hostelry example, the two last words σ' and σ'' have the property that they contain a request $wine$ not fulfilled. This is enough to qualify them as modelling business interactions not in agreement. Instead, the single word $(\overline{wine}, wine)(five, \overline{five})(-, \overline{five})$ accepted by the composition of the innkeeper and the thirsty guy contains *no* pending requests, and so we say that the two contracts are in *agreement*, in spite of the presence of an additional payment. Also, call *safe* a CA accepting only words that lead to an agreement.

Example 2. Let $\mathfrak{A}$ denote the language of all the possible strings in agreement, and let $\mathscr{L}(\mathcal{A})$ denote in the usual way the language of the contract automaton $\mathcal{A}$. The contract automaton $\mathcal{A}_3$ of Figure 1 admits agreement since $\mathscr{L}(\mathcal{A}_3) \cap \mathfrak{A} = (\overline{wine}, wine)(five, \overline{five})$. The contract automaton $Alice \otimes Bob$ of Figure 2 is safe since $\mathscr{L}(Alice \otimes Bob) = (\overline{toy}, toy) \subseteq \mathfrak{A}$.

Suppose now that a Fairy Godmother $\overline{magics}.\overline{five}$ replaces the thirsty fellow in the threefold composition of the hostelry contract (of either kind). We will have the following $(\overline{wine}, wine, -)(-, \overline{ten}, -)(five, -, -)(-, -, \overline{magics})(-, -, \overline{five})$, among other accepted words. All requests are fulfilled now, but not through a handshake: this is an example of *weak agreement*. This is a typical behaviour in loosely-coupled organizations, in which the components perform their task asynchronously. As done above, a CA only accepting words in weak agreement is called *weakly safe*.

Example 3. Consider the simple scenario of the playground swap where Alice and Bob want to share their bike and toy. Both do not trust each other, and before providing their offers they first require the corresponding requests. This is a common scenario in contracts for service oriented computing, where each participant first requires the satisfaction of its requests before providing the corresponding offers. The regular expressions for the contract automata are: $Alice = toy.\overline{bike}, Bob = bike.\overline{toy}$. The composition of contracts is: $Alice \otimes Bob = (toy, -)(\overline{bike}, bike)(-, \overline{toy}) + (-, bike)(toy, \overline{toy})(\overline{bike}, -)$. In both possible behaviours the participants fail on exchanging both the bike and the toy, hence $\mathscr{L}(Alice \otimes Bob) \cap \mathfrak{A} = \emptyset$ and the composition does not admit agreement. The circularity in the requests/offers is solved by weakening the notion of agreement,

allowing a request to be performed on credit if in the future the corresponding offer will take place. Let $\mathfrak{W}$ denote the set of all the strings in weak agreement. We have $\mathscr{L}(Alice \otimes Bob) \subseteq \mathfrak{W}$, hence the composition is weakly safe.

Checking (agreement and) safety is easily done in linear time by verifying whether all the arcs in (a path from the initial to a final state of) the given automaton are labeled by offers or by matching activities. It turns out that composing two safe/weakly safe automata with $\otimes$ is indeed safe/weakly safe, while their composition through $\boxtimes$ is an automaton that admits words in agreement/weak agreement, but also others in which some requests are left unsatisfied.

Example 4. Let $\mathcal{A}_1 = \overline{toy} + bike \otimes toy + \overline{bike}$ and $\mathcal{A}_2 = \overline{toy}$. We have that both contracts are safe but $\mathcal{A}_1 \boxtimes \mathcal{A}_2$ is not, because $(-, toy, \overline{toy})(bike, -, -) \in \mathscr{L}(\mathcal{A})$.

Controller and Liability We now move to consider a notion of *controller* for contracts, i.e. a mechanism that constrains composition to yield safe and weakly safe automata, only. In both cases we took advantage of some techniques borrowed from Control Theory [14]. As for safety, we construct a CA accepting (all and only) words in agreement, the intersection of which with the CA in hand guarantees to obtain a safe automaton. There is however a subtle point: there might be some offers or some matchings that prevent later on a request to be matched by an offer. One has therefore to detect those individual components that cause this unwanted behaviour, and that are therefore *liable*. The controller will then prune the controlled CA by removing the liable transitions (and that part of the automaton reachable through them, only). Although in a highly distributed setting one cannot assume to have a controller enforcing also malicious individuals a good behaviour, the design of a safe controlled automaton may help in synthetising the skeleton of the entire business process.

Example 5. In Figure 3 we have a simple selling scenario between two contract automata Alice and Bob. In this case Bob is willing to pay only the toy. For computing the controller $K_{\mathcal{A}}$ all the request transitions are removed, i.e. $((q_{21}, q_{22}), (-, pay), (q_{21}, q_{32}))$. After that all the hanged transitions[1] are removed and, after cleaning the automaton from all the unreachable states, the $K_{\mathcal{A}}$ is obtained. We have that the set of liable participants is $\{1, 2\}$, hence both Alice and Bob are potentially liable if the match transition with label $(\overline{bike}, bike)$ is performed. Indeed this behaviour is not prescribed by the agreement. Note that if we would consider as liable transitions only the requests, then the set of liable participants becomes erroneously $\{2\}$, since the participant 1 is also responsible of the divergence from the agreement.

We now introduce further properties that characterise the coordination aspects of contracts. We say that two contracts are *collaborative* if some requests of one meet the offers of the other, and are *competitive* if both have an offer that satisfies the same request.

[1] The hanged transitions are those which are never taken in a run of the automaton for recognizing a word.

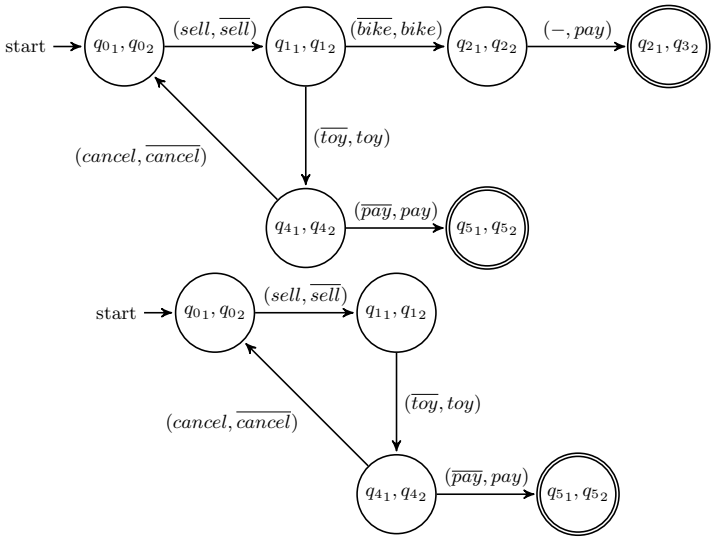

Fig. 3. Starting from the first row: $\mathcal{A}, K_{\mathcal{A}}$

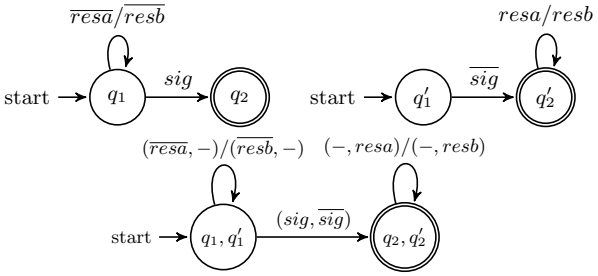

Fig. 4. From left to right: $\mathcal{A}_1, \mathcal{A}_2$, below: $\mathcal{A}_1 \otimes \mathcal{A}_2$

Example 6. In Figure 2 we have that $Alice, Bob \otimes Jill$ is a pair of collaborative contracts which is not competitive, indeed there is a match on the toy action but no participant interferes in this match. In example 4 the pair $\mathcal{A}_1, \mathcal{A}_2$ is competitive since $\mathcal{A}_2$ interferes with $\mathcal{A}_1$ on the toy offer.

It is easy to see that (any) composition of two non-collaborative unsafe contract is unsafe, and that safety is only preserved under the composition $\boxtimes$ of non-competitive contracts. Things become more complicated as soon as we consider weak agreement, because the language of such words is proved to be context-sensitive, and not context-free.

Example 7. Let $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_1 \otimes \mathcal{A}_2$ be the automata depicted in Figure 4: $\mathcal{A}_1$ produces two types of resources with the offers $\overline{resa}, \overline{resb}$ and terminates with the request sig. The contract $\mathcal{A}_2$ starts by sending the signal sig and then collects all the resources produced by $\mathcal{A}_1$

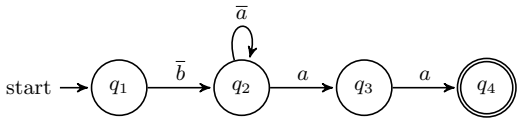

Fig. 5. A simple automaton. All words in weak agreement require the loop to be taken more than twice.

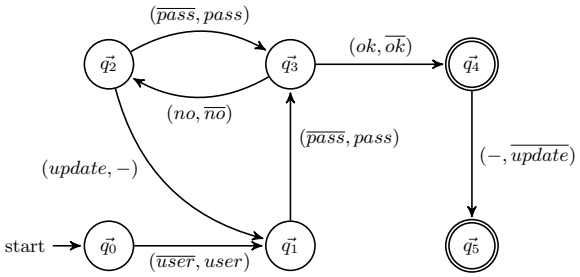

Fig. 6. A simple service of authentication

Then we have that $S = \mathfrak{W} \cap \mathscr{L}(\mathcal{A}_1 \otimes \mathcal{A}_2) \neq \emptyset$ which is not a context-free language. Indeed let $L = \{(\overline{resa}, -)^*(\overline{resb}, -)^*(sig, \overline{sig})(-, resa)^*(-, resb)^*\}$. We have that $S \cap L$ is not context-free (apply the pumping lemma), and since L is regular S is not context-free.

Still the notion is decidable, but finding at static time, e.g., the liable individuals becomes quite hard, mainly because CA have loops. Assume that our CA has, among other activities different from $a, \overline{a}$, a single loop containing an offer $\overline{a}$ (by one individual), followed by two requests a outside the loop (by another); a simple example is in Figure 5. Clearly, it admits a word in weak agreement, provided that the loop is taken at least twice. In the general case, one has a constraint system, the solutions of which will tell us how many times a loop has to be taken for guaranteeing that there is (at least) a word in weak agreement. It has the form of a system of bilinear Diophantine equations, that can be solved by optimization techniques. We are currently working on a more tractable presentation of this problem, and we feel that it can be solved in (high) polynomial time, due to the structure of the underlying regular automata.

Example 8. The contract automaton $\mathcal{A}$ depicted in Figure 6 represents a simple service of authentication. If the password is wrong the user can try again or update the account. Before storing the updated password, the service checks if the client will actually use the new data or if it will need a new update, i.e. it failed to enter the new data. In the first case, after the client is logged the update will be stored permanently.

Let $c_1 = \{(\vec{q_0}, (\overline{pass}, pass), \vec{q_3}), (\vec{q_3}, (no, \overline{no}), \vec{q_2}), (\vec{q_2}, (update, -), \vec{q_1})\}$ and let $c_2 = \{(\vec{q_3}, (no, \overline{no}), \vec{q_2}), (\vec{q_2}, (\overline{pass}, pass), \vec{q_3})\}$ be the two shortest cycles of the

automaton, in that no other sub-cycles are strictly included in c_1, c_2. One has to determine how many times, say x_{c_1} and x_{c_2}, the cycles c_1, c_2 must be walked through a run for recognizing a word in weak agreement. Since here we have a run accepting a word in weak agreement without using c_1, c_2, a solution is $x_{c_1} = 0, x_{c_2} = 0$. As a matter of fact all the solutions must satisfy the inequalities $x_{c_1} \leq 1, x_{c_2} \geq 0$.

Weak safety is analogous to safety, and there are results on when it is preserved under compositions $\otimes$ and $\boxtimes$, similar to those mentioned above.

2 Conclusions and related work

We formally described service contracts as a novel class of automata. Briefly, the benefits of our approach are:

- a compositional model for orchestration/choreographies of services;
- two composition operators of automata, reflecting one when a participant joins an existing orchestration without a global reconfiguration, while the other requires a global re-orchestration;
- different notions of proper composition of contracts, namely agreement and weak agreement, and safety and weak safety;
- a notion of liability and procedures to check it, for the different properties of contract composition;
- the concepts of collaborative and competitive contract automata, and the properties of their composition;
- an unexpected connection with results from Control Theory, Optimization Research;
- a model that may help analysing incomplete contract automata and synthesising missing services so to reach an orchestration that ensures agreement.

Related work The problem of formalizing contracts for service oriented computing, specifying and verifying the properties of a good composition has been addressed in literature using many different formalisms.

In [16, 1] behavioural contracts are expressed via suitable process algebras, where the interactions between services are modeled via I/O channels. Two different choice operators, namely internal and external choice, describe how two services interact. The internal choice requires the other party to be able to synchronize, via external choice, with all the possible branches. In our formalism the internal choice can be represented as a branching of requests. If one of these requests is not fulfilled, then the system is not able to synchronize with this internal action. This approach is extended to a multi party version by exploiting the π-calculus in [17]. The above papers focus on formalising the notion of progress of interactions. In our model we consider stronger properties: with agreement/weak agreement we require that all the requests of the contracts are satisfied, while for the property of progress it suffices that a subset of contracts meets their requests to be able to continue their interactions.

An extension of Intuitionistic Logic called Propositional Contract Logic (PCL) [10, 3] has been proposed for modeling contracts with circular offers/requests. A new operator called *contractual implication* ($\twoheadrightarrow$) is introduced for dealing with actions taken on credit. Intuitively, the contractual implication models actions taken on credit if in the future the obligations will be honoured. Our notion of *weak agreement* considers all the possible compositions of contracts where an agreement is reached only if the actions are taken on credit. We are developing a decision procedure to check if a contract automaton admits weak agreement. In the worst case this problem is NP-Hard (it can be reduced to an optimization problem of a linear integer system), but in practice it seems to be tractable.

In [8, 9] a participant may behave dishonestly, and processes and contracts are two separate entities, unlike ours. A process can fulfill its duty by obeying its contract or it becomes *culpable* otherwise — and become honest again by performing later on the prescribed actions.

We also do not assume participants to be honest, and our notion of *liability* is different than culpability. It is inspired by Control Theory [14] and expressed in language-theoretic terms. Possible misbehaviours occur in the composed contract automaton far before they become obvious. Hence computing at static time the set of all the possible liable participants is a non trivial task. Again, modeling weak agreement as an optimization problem helps in detecting liable participants, as well.

Session types are studied, among others, in [15, 19] where *global types* represent a formal specification of a choreography of services in terms of their interactions. The projection of a safe global type to its components yields safe *local type*, which are terms of a process algebra similar to [16]. From given safe local types, in [20] a choreography is synthesized, as a safe global type. In [18] local types are proved to correspond to communicating machines (CM) [13], that are finite state automata similar to ours. The main difference between the two is that CM interact through FIFO buffers, hence a participant can receive an input only if it was previously enqueued, while Contract Automata can offer/request on credit. This flexible communication mechanism allows us to model interactions with circularity features. Additionally, local types are roughly our principal automata, while global types result from our composition operators.

The I/O and the Interface Automata [21, 2] have been introduced in the field of Component Based Software Engineering and are quite similar to Contract Automata. However, our operators of composition track each participant differently than the above. Moreover we do not allow input enabled operations and non-linear behaviour (i.e. broadcasting offers to every possible request).

References

1. Acciai, L., Boreale, M., Zavattaro, G.: Behavioural contracts with request-response operations. Sci. Comput. Program. 78(2), 248–267 (2013)
2. de Alfaro, L., Henzinger, T.A.: Interface automata. In: ESEC / SIGSOFT FSE. pp. 109–120. ACM (2001)

3. Bartoletti, M., Cimoli, T., Giamberardino, P.D., Zunino, R.: Contract agreements via logic. In: Carbone, M., Lanese, I., Lluch-Lafuente, A., Sokolova, A. (eds.) ICE. EPTCS, vol. 131, pp. 5–19 (2013)
4. Bartoletti, M., Cimoli, T., Pinna, G.M.: Lending petri nets and contracts. In: Arbab, F., Sirjani, M. (eds.) FSEN. Lecture Notes in Computer Science, vol. 8161, pp. 66–82. Springer (2013)
5. Bartoletti, M., Cimoli, T., Zunino, R.: A theory of agreements and protection. In: Basin, D.A., Mitchell, J.C. (eds.) POST. Lecture Notes in Computer Science, vol. 7796, pp. 186–205. Springer (2013)
6. Bartoletti, M., Degano, P., Ferrari, G.L.: Planning and verifying service composition. Journal of Computer Security 17(5), 799–837 (2009)
7. Bartoletti, M., Degano, P., Ferrari, G.L., Zunino, R.: Call-by-contract for service discovery, orchestration and recovery. In: Wirsing, M., Hölzl, M.M. (eds.) Results of the SENSORIA Project, Lecture Notes in Computer Science, vol. 6582, pp. 232–261. Springer (2011)
8. Bartoletti, M., Tuosto, E., Zunino, R.: Contract-oriented computing in co2. Sci. Ann. Comp. Sci. 22(1), 5–60 (2012)
9. Bartoletti, M., Tuosto, E., Zunino, R.: On the realizability of contracts in dishonest systems. In: Sirjani, M. (ed.) COORDINATION. LNCS, vol. 7274, pp. 245–260. Springer (2012)
10. Bartoletti, M., Zunino, R.: A logic for contracts. In: Cherubini, A., Coppo, M., Persiano, G. (eds.) ICTCS. pp. 34–37 (2009)
11. Bartoletti, M., Zunino, R.: A calculus of contracting processes. In: LICS. pp. 332–341. IEEE Computer Society (2010)
12. Basile, D., Degano, P., Ferrari, G.L.: Secure and unfailing services. In: Malyshkin, V. (ed.) PaCT. Lecture Notes in Computer Science, vol. 7979, pp. 167–181. Springer (2013)
13. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)
14. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
15. Castagna, G., Dezani-Ciancaglini, M., Padovani, L.: On global types and multi-party session. Logical Methods in Computer Science 8(1) (2012)
16. Castagna, G., Gesbert, N., Padovani, L.: A theory of contracts for web services. ACM Trans. Program. Lang. Syst. 31(5) (2009)
17. Castagna, G., Padovani, L.: Contracts for mobile processes. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR. LNCS, vol. 5710, pp. 211–228. Springer (2009)
18. Deniélou, P.M., Yoshida, N.: Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M.Z., Peleg, D. (eds.) ICALP (2). Lecture Notes in Computer Science, vol. 7966, pp. 174–186. Springer (2013)
19. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: Necula, G.C., Wadler, P. (eds.) POPL. pp. 273–284. ACM (2008)
20. Lange, J., Tuosto, E.: Synthesising choreographies from local session types. In: Koutny, M., Ulidowski, I. (eds.) CONCUR. Lecture Notes in Computer Science, vol. 7454, pp. 225–239. Springer (2012)
21. Lynch, N.A., Tuttle, M.R.: Hierarchical correctness proofs for distributed algorithms. In: Schneider, F.B. (ed.) PODC. pp. 137–151. ACM (1987)