Interfaces of Roles

in Distributed Collaborative Systems

Eric Badouel[(a)] and Rodrigue Aimé Djeumen Djatcha[(b)]

(a) Inria Rennes-Bretagne Atlantique, Irisa, University of Rennes I,
Campus Universitaire de Beaulieu, 35042 Rennes Cedex, France
(b) Faculty of Sciences, University of Douala, BP 24157 Douala, Cameroon
eric.badouel@inria.fr djeumenr@yahoo.fr
This work was partially supported by ANR Headwork.

ABSTRACT. We address the problem of component reuse in the context of service-oriented programming and more specifically for the design of user-centric distributed collaborative systems modelled by Guarded Attribute Grammars. Following the contract-based specification of components we developp an approach to an interface theory for the roles in a collaborative system in three stages: we define a composition of interfaces that specifies how the component behaves with respect to its environement, we introduce an implementation order on interfaces and finally a residual operation on interfaces characterizing the systems that, when composed with a given component, can complement it in order to realize a global specification.

RÉSUMÉ. Nous abordons le problème de la réutilisation des composants dans le contexte de la programmation orientée services et plus spécifiquement pour la conception de systèmes collaboratifs distribués centrés sur l'utilisateur modélisés par des grammaires attribuées gardées. En suivant la démarche de la spécification contractuelle des composants, nous développons une approche de la théorie des interfaces pour les rôles d'un système collaboratif en trois étapes: on définit une composition d'interfaces qui spécifie comment le composant se comporte par rapport à son environnement, on introduit un ordre d'implémentation sur les interfaces et enfin une opération de résidus sur les interfaces qui caractérise les systèmes qui, lorsqu'ils sont composés avec un composant donné, peuvent le compléter afin de réaliser une spécification du système global.

KEYWORDS : Component Based Design, Service Oriented Programming, Interface, Role, Collaborative System, Guarded Attribute Grammars

MOTS-CLÉS : Conception à base de composants, Programmation orientée services, Interface, rôle, systèmes collaboratif, grammaires attribuées gardées

1. Introduction

We address the problem of component reuse in the context of service-oriented programming and more specifically for the design of user-centric distributed collaborative systems. The role of a specific user is given by all the services he or she offers to the environment. A role can be encapsulated by a module whose interface specifies the provided services the module exports and the required external services that it imports. Usually the modules in a service-oriented design are organized hierarchically. In contrast, modules in a distributed collaborative systems would often depend on each other (even though cyclic dependencies between services should be avoided). Moreover services that are currently activated can operate as coroutines and a service call can activate new services in a way that may depend on the user's choice of how to provide the service. We thus need a richer notion of interface for roles in a distributed collaborative system. In this paper we consider a very simple extension of the concept of interface obtained by adding a binary relation on the set of services indicating for each of the provided services the list of services that are potentially required to carry it out. This relation gives only *potential dependencies* because a user can provide a service in various ways and relying on a variety of external services. We motivate our presentation in the context of systems modelled by Guarded Attribute Grammars [4]. Possible extensions of this basic model of interface are mentioned in the concluding section. They would provide finer descriptions of the behaviour of a module in a Guarded Attribute Grammar specification.

Even if the objectives differ (service-oriented design versus verification) as well as the models used (user-centric collaborative systems versus reactive systems) we are largely inspired by the works that have been carried out on behavioural interfaces of communicating processes. Three main ingredients have been put forward in these studies which will serve as our guideline.

First, an interface is mainly used to formalize a contract-based reasoning for components. The idea is that a component of a reactive system [5] is required to behave correctly only when its environment does. The correctness of composition is stated in terms of a contract given by *assume-guarantee* conditions: the component should guarantee some expected behaviour when plugged into an environment that satisfies some properties. The principle of composition is however made subtle by the fact that each component takes part in the others' environment [1]. Safety and liveness properties, which are not relevant in our case, are crucial issues in this context and largely contribute to the complexity of the resulting formalisms. The underlying models of a component range from process calculi [2] to I/O automata and games [3]. These interface theories have also been extended to take some qualitative aspects (time and/or probability) into account.

Second, an interface is viewed as an abstraction of a component, a so-called *behavioural type*. Thus we must be able to state when a component satisfies an interface, viewed as an abstract specification of its behaviour. A relation of refinement, given by a pre-order $I_1 \leq I_2$, indicates that any component that satisfies I_2 also satisfies I_1. In the context of service-oriented programming we would say that interface I_2 *implements* interface I_1.

Third, a notion of *residual specification* has also proved to be useful. The problem was first stated in [6] as a form of equation solving on specifications. Namely, given a specification G of the desired overall system and a specification C of a given component we seek for a specification X for those systems that when composed with the component satisfies the global property. It takes the form of an equation $L \bowtie X \approx G$ where $\bowtie$

stands for the composition of specifications and $\approx$ is some equivalence relation. If $\approx$ is the equivalence induced by the refinement relation the above problem can better be formulated as a Galois connection [9] $G/L \leq X \iff G \leq L \bowtie X$ stating that the residual specification G/L is the smallest (i.e. less specific or more general) specification that when composed with the local specification is a refinement of the global specification. Since $L \bowtie -$ is monotonous (due to Galois connection) it actually entails that a component is an implementation of the residual specification if and only if it provides an implementation of the global specification when composed with an implementation of the local specification .

2. Roles in a Collaborative System Modelled by a Guarded Attribute Grammar

Guarded Attribute Grammars (GAG) [4] is a user centric model of collaborative work that puts emphasis on task decomposition and the notion of user's workspace. We assume that a workspace contains, for each service offered by the user, a repository that contains one artifact for each occurrence of a service call (that initiates a so-called *case* in the system). An artifact is a tree that records all the information related to the treatment of the case. It contains open nodes corresponding to pending tasks that require user's attention. In this manner, the user has a global view of the activities in which he or she is involved, including all relevant information needed for the treatment of the pending tasks.

Each *role* (played by some users) is associated with a grammar that describes the dynamic evolution of a case. A production of the grammar is given by a left-hand side, indicating a non-terminal to expand, and a right-hand side, describing how to expand this non-terminal. We interpret a production as a way to decompose a task, the symbol on the left-hand side, into sub-tasks associated with the symbols on the right-hand side. The initial tasks are symbols that appear in some left-hand side (they are *defined*) but do not appear on right-hand side of rules (they are not *used*). They correspond to the *services* that are *provided* by the role. Conversely a symbol that is used but not defined (i.e it appears on some right-hand side but on no left-hand side) is interpreted as a call to an external service. It should appear as a service provided by another role. Symbols that are both used and defined are internal tasks and their names are bound to the role.

$p_1 : \quad A \to \varepsilon$
$p_2 : \quad A \to BC$
$p_3 : \quad B \to \varepsilon$
$p_4 : \quad B \to D$

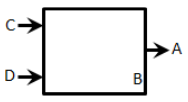

Figure 1. *A grammar for a role that provides a service A and uses the external services C and D. B is an internal task, bound to the role, and whose name can henceforth be changed.*

In order to solve a task A, that appears as a pending task in his workspace, the user may choose to apply production p_1 (which corresponds to a certain action or activity) and this decision ends the performance of task A (since the right-hand side is empty). Alternatively production p_2 may be chosen. In that case, two new (residual) tasks of respective sort B and C are created and A will terminate as soon as B and C have terminated.

The GAG model also attach (inherited and synthesized) information to a task as well as a guard (condition bearing on the inherited information) that specifies when the production is enabled. In this paper we restrict our attention to the dynamic evolution of tasks (the grammar) and forget about extra information and guards.

Our purpose is to define some abstraction of the grammar, called the *interface of the role*, whose aim is to specify what services are provided, which external services are required to carry them out and an over-approximation of the dependencies between required and provided services (the potential dependencies). In particular the interface disregards internal tasks. As a first attempt one considers that the provided service A potentially relies on external service B if a derivation $A \rightarrow^* u$ exists where word u contains an occurrence of B.

The interface of the role given in Figure 2 is relation $R = \{(C, A), (D, A)\}$. It is an over-approximation of the dependencies since it may happen that A uses none of the services C and D (using derivation $A \rightarrow^* \varepsilon$) or only C (using derivation $A \rightarrow^* C$). But an external user invoking service A does not know how the service will be carry out and therefore he must assume the availability of all external services that may potentially be used.

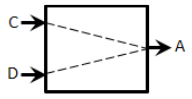

Figure 2. *An interface*

We assume that the grammars are *non-recursive* in the sense that no symbol can derive from itself. Namely we exclude the situation where a derivation $X \rightarrow^* u$ exists in which u is a word that contains an occurrence of X. This condition is very generally verified in the examples we have encountered in practice, for instance when modeling epidemiological surveillance [7]. [1]

Definition 2.1. *Let Ω denote a fixed set of services. An* interface $({}^\bullet R, R, R^\bullet)$ *consists of a finite binary relation $R \subseteq \Omega \times \Omega$ and disjoint subsets ${}^\bullet R$ and $R^\bullet$ of Ω, such that ${}^\bullet R = R^{-1}(\Omega) = \{A \in \Omega \mid \exists B \in \Omega\ (A, B) \in R\}$ and $R^\bullet \supseteq R(\Omega) = \{B \in \Omega \mid \exists A \in \Omega\ (A, B) \in R\}$. The set $R^\bullet$ stands for the services* provided *(or* defined*) by the interface and ${}^\bullet R$ for the* required *(or* used*) services. The relation $(A, B) \in R$ indicates that service B potentially depends upon service A. Thus $A \in R^\bullet \setminus R(\Omega)$ is a service provided by the interface that requires no external services. An interface is* closed *(or* autonomous*) if relation R (and thus also ${}^\bullet R$) is empty. Thus a closed interface is given by the set of services that it (autonomously) provides.*

Note that since ${}^\bullet R$ is the domain of relation R, the set of required services may be left implicit. The same is not true for the set of provided services since it can strictly encompass the codomain of the relation. Still, we shall by abuse of notation use the same symbol to denote an interface and its underlying relation. We extend the following notations from binary relations to interfaces:

1) The empty interface that renders no service at all is $\emptyset = (\emptyset, \emptyset, \emptyset)$.

1. However, it can sometimes be useful to model situations where a task A derives into an arbitrary number of tasks B. Such a situation can be presented by the recursive grammar with rules $A \rightarrow B\ A$ and $A \rightarrow \varepsilon$ which may equivalently be given by the (generalized) production: $A \rightarrow B^*$. Hence, one may be tempted to use non-recursive but generalized grammars (whose right-hand sides are given by regular expressions). However, as we are interested only in the dependencies between services, one can w.l.o.g. replace any regular expression on a right-hand side by the sequence of symbols (without repetition) that occur in it and therefore obtaining an ordinary non-recursive grammar with the same interface.

2) $R_1; R_2 = \{(A,C) \in \Omega^2 \mid \exists B \in \Omega\ (A,B) \in R_1 \wedge (B,C) \in R_2\}$ is the sequential composition with ${}^\bullet(R_1;R_2) = {}^\bullet R_1$ and $(R_1;R_2)^\bullet = R_2^\bullet$.

3) the restriction $R \restriction O$ of interface R to $O \subseteq \Omega$ is given by $R|O = \{(A,B) \in R \mid B \in O\}$ with $(R|O)^\bullet = O \cap R^\bullet$ and ${}^\bullet(R|O) = R^{-1}(O \cap R^\bullet)$.

3. The Composition of Interfaces

The union of interfaces is an interface if none of the services defined by an interface is used by another one. In the general case $R = (\cup_i {}^\bullet R_i, \cup_i R_i, \cup_i R_i^\bullet)$ satisfies the conditions in Definition 2.1 but ${}^\bullet R \cap R^\bullet = \emptyset$. If relation R^* is acyclic we say that it is a *quasi-interface* since it induces an interface given by the following definition.

Definition 3.1. *If $R = ({}^\bullet R, R, R^\bullet)$ is a quasi-interface, i.e. satisfies ${}^\bullet R = R^{-1}(\Omega)$, $R^\bullet \supseteq R(\Omega)$, and its transitive closure R^* is acyclic, then we let $\langle R\rangle = R^* \cap (I \times O)$, where $I = {}^\bullet R \setminus R^\bullet$ and $O = R^\bullet$. It is an interface with ${}^\bullet\langle R\rangle = I$ and $\langle R\rangle^\bullet = O$.*

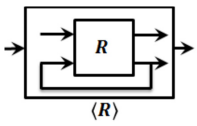

Figure 3. *Interface induced by a quasi-interface*

For instance if $R_1 = (\emptyset, \emptyset, \{A\})$ is the autonomous interface that provide service A and $R_2 = (\{A\}, \{(A,B\}, \{B\})$ uses A to define another service B, then they jointly provide an autonomous interface $\langle R_1 \cup R_2\rangle = (\emptyset, \emptyset, \{A,B\})$ that provides services A and B. Note that the information that B requires A is lost: the meaningful information is that the interface exports A and B and has no imports. If we assume that interface R_1 rather produces service B from A, namely $R_1 = (\{B\}, \{(B,A)\}, \{A\})$, then the computation of the composition would also give $\langle R_1 \cup R_2\rangle = (\emptyset, \emptyset, \{A,B\})$ even though these two interfaces when combined together cannot render any service. This is the rationale for assuming that a quasi-interface must be acyclic.

Definition 3.2. *Two interfaces R_1 and R_2 are said to be composable if $(R_1 \cup R_2)^*$ is acyclic and $R_1^\bullet \cap R_2^\bullet = \emptyset$. Then we let $R_1 \bowtie R_2 = \langle R_1 \cup R_2\rangle$ denote their composition.*

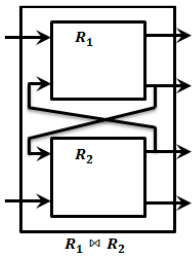

Figure 4. *The composition of two interfaces.*

Note that $(R_1 \bowtie R_2)^\bullet = R_1^\bullet \cup R_2^\bullet$.
Moreover, since ${}^\bullet R_i \cap R_i^\bullet = \emptyset$ for $i = 1,2$ one gets

$$ {}^\bullet(R_1 \bowtie R_2) = ({}^\bullet R_1 \setminus R_2^\bullet) \cup ({}^\bullet R_2 \setminus R_1^\bullet) $$

It follows also directly from the definition that the composition of interfaces is commutative and has the empty interface as neutral element. Note that we may have ${}^\bullet R_1 \cap {}^\bullet R_2 \neq \emptyset$, thus both interfaces may require some common external services. The following example shows that the composition is not associative if we do not require that composable interfaces have disjoints outputs.

Exemple 3.3. *Let R_1, R_2, and R_3 the three interfaces given in Figure 5. If $R_1 \bowtie (R_2 \bowtie R_3) = (R_1 \bowtie R_2) \bowtie R_3$ we would expect this interface to be given by $R = \langle R_1 \cup R_2 \cup R_3\rangle$ hence $R = \{(A,D),(C,D),(A,E),(B,E),(C,E)\}$. Note that service D may be produced*

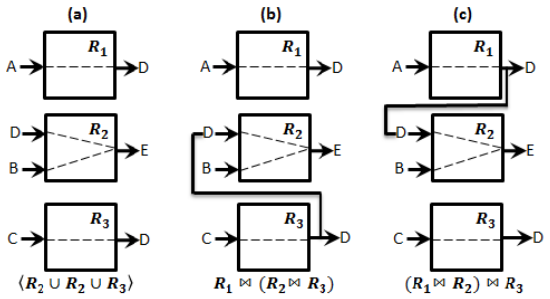

Figure 5. *A counter-example showing that associativity of composition does not hold if interfaces shared some provided services.*

by either R_1 or R_3 so that we find both (A, D) and (C, D) as dependencies in R. It follows that E potentially depends on both A, B, and C. However if we compute $R_1 \bowtie (R_2 \bowtie R_3)$ we get $R_r = \{(A, D), (C, D), (B, E), (C, E)\}$ because in R_2 the required service D is no longer an input in $R_2 \bowtie R_3$. Symmetrically $R_l = (R_1 \bowtie R_2) \bowtie R_3 = \{(A, D), (C, D), (A, E), (C, E)\}$.

Proposition 3.4. *The composition of interfaces is associative. More precisely, if $R_1 \cdots R_n$ are pairwise composable interfaces, then $\bowtie_{i=1}^{n} R_i = \langle R_1 \cup \cdots \cup R_n \rangle$.*

The following two cases of composition are noteworthy:

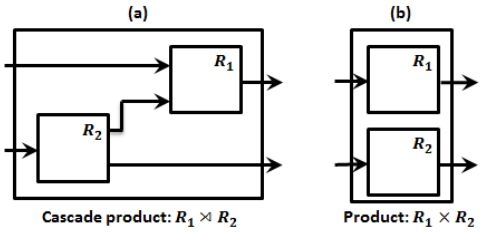

Figure 6. *Cascade product and (direct) product*

Cascade product If $R_1^\bullet \cap {}^\bullet R_2 = \emptyset$ we denote $R_1 \rtimes R_2$ their composition (or $R_2 \ltimes R_1$ since this operation as a particular case of $\bowtie$ is still commutative). Then ${}^\bullet(R_1 \rtimes R_2) = ({}^\bullet R_1 \setminus R_2^\bullet) \cup {}^\bullet R_2$, and $(R_1 \rtimes R_2)^\bullet = R_1^\bullet \cup R_2^\bullet$.

(Direct) product If both $R_1^\bullet \cap {}^\bullet R_2 = \emptyset$ and $R_2^\bullet \cap {}^\bullet R_1 = \emptyset$ hold we say that the composition is the product of R_1 and R_2, denoted as $R_1 \times R_2$. Note that $R_1 \times R_2 = R_1 \cup R_1$ and thus ${}^\bullet(R_1 \times R_2) = {}^\bullet R_1 \cup {}^\bullet R_2$ and $(R_1 \times R_2)^\bullet = R_1^\bullet \cup R_2^\bullet$.

Definition 3.5. *R_1 is a* component *of R, in notation $R_1 \sqsubseteq R$, if there exists an interface R_2 such that $R = R_1 \bowtie R_2$. R_1 is a* strict component *of R, in notation $R_1 \sqsubset R$, if there exists an interface R_2 such that $R = R_1 \times R_2$.*

4. Implementation Order

An environment for an interface is any component that provides all the services required by the interface and uses for that purpose only services that are provided by it.

Definition 4.1. *An interface E is an* admissible environment *for an interface R if the two interfaces are composable and the resulting composition is a closed interface, namely ${}^\bullet(R \bowtie E) = \emptyset$. We let $\mathbf{Env}(R)$ denote the set of admisible environments of interface R.*

Definition 4.2. *An interface R_2 is an* implementation *of interface R_1, in notation $R_1 \leq R_2$, when $R_2^\bullet = R_1^\bullet$ and $R_2 \subseteq R_1$.*

Thus R_2 is an implementation of R_1 if it renders the same services as R_1 using only services already used by R_1 and with less dependencies.[2] The following proposition shows that R_2 is an implementation of interface R_1 if and only if it can be substituted to R_1 in any admissible environment for R_1.

Proposition 4.3. *$R_1 \leq R_2$ if and only if $\mathbf{Env}(R_1) \subseteq \mathbf{Env}(R_2)$.*

5. Residual Specification

Proposition 5.1. *If $R_1 \sqsubseteq R$ then $R = R_1 \rtimes (R \swarrow R_1)$ where $R \swarrow R_1$, called the* strict residual *of R by R_1, is given as the restriction of R to $R^\bullet \setminus R_1^\bullet$. If $R = R_1 \rtimes R_2$ then $R{\restriction}R_2^\bullet = R \swarrow R_1 = R_2$ and $R = (R \swarrow R_2) \times (R \swarrow R_1)$.*

Corollary 5.2. *If $R^\bullet = O_1 \cup O_2$ with $O_1 \cap O_2 = \emptyset$ then $R = (R{\restriction}O_1) \times (R{\restriction}O_2)$ and $R{\restriction}O_i = R \swarrow (R{\restriction}O_j)$ for $\{i, j\} = \{1, 2\}$ and the following conditions are equivalent:*

1) R_1 is a strict component of R: $\exists R_2 \cdot R = R_1 \times R_2$,

2) R_1 is a left component in a cascade decomposition of R: $\exists R' \cdot R = R' \rtimes R_2$,

3) R_1 is a restriction of R: $R_1 = R{\restriction}(R_1^\bullet)$, and

4) R_1 is a strict residual of R: $\exists R_2 \cdot R_1 = R \swarrow R_2$.

Proposition 5.3. *If R_1 is a component of R and R' is an interface then*

$$R \swarrow R_1 \leq R' \iff R \leq R_1 \rtimes R'.$$

By Corollary 5.2 the above proposition implies that an implementation of a strict residual $R \swarrow R_1$ is a strict component of R and therefore it cannot capture all the components of an implementation of R, i.e. all interfaces R' such that $R \leq R_1 \bowtie R'$. For that purpose we need to add in the residual all the dependencies between the respective ouputs of the component and of the residual that do not contradict dependencies in R:

Definition 5.4. *If $R_1 \sqsubseteq R$ the* residual *R/R_1 of R by R_1 is given by $(R/R_1)^\bullet = R^\bullet \setminus R_1^\bullet$ and $R/R_1 = R \swarrow R_1 \cup R \nearrow R_1$ where*

$$R \nearrow R_1 = \left\{ (A, B) \in R_1^\bullet \times (R^\bullet \setminus R_1^\bullet) \;\middle|\; R^{-1}(\{A\}) \subseteq R^{-1}(\{B\}) \right\}.$$

2. In pratice an interface used as an implementation may define additional services: R_2 is a *weak implementation* of interface R_1, in notation $R_1 \leq_w R_2$, if $R_2^\bullet \supseteq R_1^\bullet$ and $R_1 \leq R_2{\restriction}(R_1^\bullet)$. However the additional services provided by R_2 should be hided so that they cannot conflict with services of any environment compatible with R_1.

Proposition 5.5. *If R_1 is a component of R and R' is an interface then*

$$R/R_1 \leq R' \iff R \leq R_1 \bowtie R'.$$

Hence the residual R/R_1 characterizes those interfaces that, when composed with R_1, produce an implementation of R.

6. Conclusion

This work is a first attempt to develop an interface theory for distributed collaborative systems in the context of service-oriented programming. We intend to use it to define and structure the activities of crowdsourcing system operators. The residual operation can be used to identify the skills to be sought in the context of existing services in order to achieve a desired overall behaviour. Such a system can be implemented by Guarded Attribute Grammars and interfaces can be used to type applications. However, the notion of interface presented in this paper is still a somewhat rudimentary abstraction of the roles described by a GAG specification. In particular, we would like to be able to take into account the non-determinism resulting from the choices of users in their ways to solve a given task. This could be done by replacing the relation $R \subseteq \Omega \times \Omega$ by a map $R : \Omega \to \wp(\wp(\Omega))$ that associates each service $A \in \Omega$ with a finite number of alternative ways to carry it out, and each of these with the set of external services that it requires. Then we would have $R^\bullet = \{A \in \Omega \mid R(\Omega) \neq \emptyset\}$ and ${}^\bullet R = \cup\{R(A) \mid A \in \Omega\}$. The composition, implementation (pre-)order, and residual would have to be adapted in this context. It would then be possible to define some new operations like the corestriction $R\lfloor I$ of an interface R to a set of services $I \subseteq \Omega$, where $(R\lfloor I)(A) = \{X \cap I \mid X \in R(A)\}$ states how a role can be used when the set of services actually provided by the environment is known (to be I). Now it might be possible that we have only a partial knowledge of the set of available services in the form of a believe function [10] or a possibilistic distribution [11]. Then we should enrich an interface with qualitative information and viewed it as a believe function transformer that updates the knowledge on the services rendered by the environment when a new role enters the system. Finally, one can also enrich the interface with information on time execution.

7. References

[1] MARTÍN ABADI, LESLI LAMPORT, "Composing Specifications", *ACM Transactions on Programming Languages and Systems*, vol. 15, 1993:73–132.

[2] MARTÍN ABADI, GORDON D. PLOTKIN, "A logical view of composition", *Theoretical Computer Science*, vol. 114, 1993:3–30.

[3] LUCA DE ALFARO, THOMAS A. HENZINGER, "Interface Automata", *Foundation of Software Engineering (ESEC/FES-9)*, 2001: 109–120.

[4] ERIC BADOUEL, LOÏC HÉLOUËT , GEORGES-EDOUARD KOUAMOU, CHRISTOPHE MORVAN , ROBERT FONDZE JR. NSAIBIRNI, "Active Workspaces; Distributed Collaborative Systems based on Guarded Attribute Grammars", *ACM SIGAPP Applied Computing Review*, vol. 15, num. 3, 2015:6–34.

[5] DAVID HAREL, AMIR PNUELI, "On the development of reactive systems", *Logics Models of Concurrent Systems*, NATO ASI Series, vol. F13, Springer Berlin, 1984: 477–498.

[6] PHILIP M. MERLIN, GREGOR VON BOCHMANN, "On the construction of submodule specifications and communication protocols", *ACM Transactions on Programming Languages and Systems*, vol. 5, 1983:1–25.

[7] ROBERT FONDZE JR NSAIBIRNI, ERIC BADOUEL, GAËTAN TEXIER , GEORGES-EDOUARD KOUAMOU, "Active Workspace: A Dynamic Collaborative Business Process Model for Disease Surveillance Systems", *Health Informatics and Medical Systems*, Las Vegas, USA, 2016: 58–64.

[8] VAUGHAN R. PRATT, "Origins of the Calculus of Binary Relations", *IEEE Logic in Computer Science (LICS'92)*, Santa Cruz, California, USA, 1992: 248–254.

[9] JEAN-BAPTISTE RACLET, "Residual for Component Specifications", *Electronic Notes in Theoretical Computer Science*, vol. 215, 2008:93–110.

[10] GLENN SHAFER, *A mathematical theory of evidence*, Princeton University Press, 1976.

[11] LOFTI ZADEH, "Fuzzy Sets as the Basis for a Theory of Possibility", *Fuzzy Sets and Systems*, vol. 1, 1978:3–28.

Appendix: Proofs of Results

The theory of interfaces that we consider is mainly a calculus of relations [8] even though we put stress on the (concurrent) composition rather than on the usual (sequential) composition of relations. As a result we have introduced a residuation operation for the composition in place of the left and right residuals for sequential composition. Similarly our implementation order is mostly given by the set-theoretical inclusion of relations. In order to ease computation we identify a set $X \subseteq \Omega$ with the interface $(X = (X, \{(A,A) \in \Omega^2 \mid A \in X\}, X)$. By doing so, one can for instance express the condition $B \in Y \wedge (\exists C \in X\ (A,C) \in R \wedge (C,B) \in Y)$ for $R; S \subseteq \Omega \times \Omega$ and $X, Y \subseteq \Omega$ as $(A,B) \in R; X; S; Y$. One can also express the cascade product as a sequential composition:

Remark 7.1. *$R_1 \rtimes R_2 = (I_1 \times R_2)\,;(R_1 \times O_2)$ where $I_1 = {}^\bullet R_1 \setminus R_2^\bullet$ and $O_2 = R_2^\bullet \setminus {}^\bullet R_1$.*

Note moreover that with this convention one has $R\restriction X = R; X$ and $X \cap Y = X; Y$ for R an interface and X and Y subsets of Ω.

1. Associativity of the Composition of Interfaces

Remark 7.2. *$\langle R\rangle = \{(A,B) \in R^* \mid \neg\,(\exists C \in \Omega.\ (C,A) \in R)\}$. Hence any $(A,B) \in \langle R\rangle$ is associated with a path in the graph of R that leads to $B \in R^\bullet$ and cannot be extended on the left. Note that such a path is of the form $A = A_0 \to A_1 \to \cdots \to A_n = B$, with $A \in {}^\bullet R \setminus R^\bullet$ and $\forall 1 \leq i \leq n\ (A_{i-1}, A_i) \in R$. Note that $\forall 1 \leq i \leq n\ A_i \in R^\bullet$, i.e. all elements in this path but the first one, namely A, belongs to $R^\bullet$.*

Proposition. *The composition of interfaces is associative. More precisely, if $R_1 \cdots R_n$ are pairwise composable interfaces, then $\bowtie_{i=1}^n R_i = \langle R_1 \cup \cdots \cup R_n\rangle$.*

Proof. Using the commutativity of composition, the proposition follows by induction on n as soon as it has been verified that $(R_1 \bowtie R_2) \bowtie R_3 = \langle R_1 \cup R_2 \cup R_3\rangle$ for pairwise composable interfaces R_1, R_2 and R_3. Hence we have to show $\langle\langle R_1 \cup R_2\rangle \cup R_3\rangle = \langle R_1 \cup R_2 \cup R_3\rangle$ or, more generally, that $\langle\langle R\rangle \cup R'\rangle = \langle R \cup R'\rangle$ where $R \subseteq \Omega \times \Omega$ is a finite binary relation with possibly ${}^\bullet R \cap R^\bullet \neq \emptyset$, and R' is an interface such that $(R \cup R')^*$ acyclic, and $R^\bullet \cap (R')^\bullet = \emptyset$. First, note that $\langle R\rangle^\bullet = R^\bullet$ and $(R \cup R')^\bullet = R^\bullet \cup (R')^\bullet$ and

thus $\langle\langle R\rangle \cup R'\rangle^\bullet = \langle R \cup R'\rangle^\bullet$. By condition $R^\bullet \cap (R')^\bullet = \emptyset$ we deduce $R \cap R' = \emptyset$. More precisely a transition $(A, B) \in R \cap R'$ belongs (exclusively) either to R or to R' depending respectively on $B \in R^\bullet$ or $B \in (R')^\bullet$. According to Remark 7.2, let $\pi = A_0 \to A_1 \to \cdots \to A_n$ be a path in $R \cup R'$ (i.e. $\forall 1 \leq i \leq n\ (A_{i-1}, A_i) \in R \cup R'$ and $A_0 \in {}^\bullet(R \cup R') \setminus (R \cup R')^\bullet$) witnessing that $(A_0, A_n) \in \langle R \cup R'\rangle$. Let $\pi' = A_i \to \cdots \to A_j$ be a maximal sub-path of π made of R transitions only (i.e., $\forall i \leq k \leq j\ A_k \in R^\bullet$). Then either $A_i = A_0$ or $(A_{i-1}, A_i) \in R'$. In both cases $A_i \in {}^\bullet R \setminus R^\bullet$ and thus π' is a path witnessing that $(A_i, A_j) \in \langle R\rangle$ from which it follows that π is a path witnessing that $(A_0, A_n) \in \langle\langle R\rangle \cup R'\rangle$, showing $\langle R \cup R'\rangle \subseteq \langle\langle R\rangle \cup R'\rangle$ and hence $\langle\langle R\rangle \cup R'\rangle = \langle R \cup R'\rangle$ since the converse inclusion is immediate. □

2. Implementation Order

Proposition. $R_1 \leq R_2$ *if and only if* $\mathbf{Env}(R_1) \subseteq \mathbf{Env}(R_2)$.

Proof. We first show that the condition is necessary. For that purpose let us assume $R_1 \leq R_2$ (which means that $R_2^\bullet = R_1^\bullet$ and $R_2 \subseteq R_1$) and prove that any admissible environment E for R_1 is an admissible environment for R_2. Since E is composable with R_1 we get $R_1^\bullet \cap E^\bullet = \emptyset$ and $(E \cup R_1)^*$ is acyclic. Then we also have $R_2^\bullet \cap E^\bullet = \emptyset$ and $(E \cup R_2)^*$ is acyclic since $R_2^\bullet = R_1^\bullet$ and $R_2 \subseteq R_1$. Hence E is composable with R_2. Moreover, for the same reasons, ${}^\bullet(E \bowtie R_2) = ({}^\bullet E \setminus R_2^\bullet) \cup ({}^\bullet R_2 \setminus E^\bullet) \subseteq ({}^\bullet E \setminus R_1^\bullet) \cup ({}^\bullet R_1 \setminus E^\bullet) = {}^\bullet(E \bowtie R_2) = \emptyset$. Henceforth $E \in \mathbf{Env}(R_2)$. We show that the condition is sufficient by contradiction. Since $R_1 \leq R_2$ implies $R_2^\bullet = R_1^\bullet$ one has to construct $\mathcal{E} \in \mathbf{Env}(R_1) \setminus \mathbf{Env}(R_2)$ under the assumption that $R_1 \not\supseteq R_2$. Let $(A, B) \in R_2 \setminus R_1$ then the interface we are looking for is E such that ${}^\bullet E = \{B\}$, $E^\bullet = {}^\bullet R_2$, and $E = \{(B, A)\}$. Indeed, E is composable with R_1 but not with R_2 because of the cycle $B \to A \to B$ in $(R_1 \cup \{(B, A)\})^*$. Moreover the composition of E with R_1 gives a closed interface. □

3. Residual specification

Proposition. *If* $R_1 \sqsubseteq R$ *then* $R = R_1 \rtimes (R \swarrow R_1)$ *where* $R \swarrow R_1$*, called the* strict residual *of* R *by* R_1*, is given as the restriction of* R *to* $R^\bullet \setminus R_1^\bullet$*. If* $R = R_1 \rtimes R_2$ *then* $R \restriction R_2^\bullet = R \swarrow R_1 = R_2$ *and* $R = (R \swarrow R_2) \times (R \swarrow R_1)$.

Proof. One has to show that if R_1 and R_2 are two composable relation with $R = R_1 \bowtie R_2$ then $R = R_1 \rtimes R \swarrow R_1$ and $R = (R \swarrow R_1) \times (R \swarrow R_1)$ where $R \swarrow R_i = R \restriction R_j^\bullet$ for $\{i, j\} = \{1, 2\}$. By remark 7.2 $R_1 \bowtie R_2$ is the (unique)[3] solution of the system of equations

$$
\begin{array}{rrl}
 & R_1 \bowtie R_2 = & (A \cup I_1); R_1 \cup (B \cup I_2); R_2 \\
\mathit{where} & I_1 = & {}^\bullet R_1 \setminus R_2^\bullet \\
 & I_2 = & {}^\bullet R_2 \setminus R_1^\bullet \\
 & O_1 = & R_1^\bullet \cap {}^\bullet R_2 \\
 & O_2 = & R_2^\bullet \cap {}^\bullet R_1 \\
 & A = & (B \cup I_2); R_2; O_2 \\
 & B = & A \cup I_1); R_1; O_1
\end{array}
$$

3. Unicity comes from the fact that one considers only finite paths due to acyclicity.

Then it is immediate (see Figure 7) that $R_1 \rtimes (R_1 \bowtie R_2) \lceil Out(R_2)$ is solution of the same system of equations and thus the two relations coincide. The same system of equations is associated with $(R_1 \swarrow R_2) \times (R_1 \swarrow R_1)$ as shown in Figure 8.

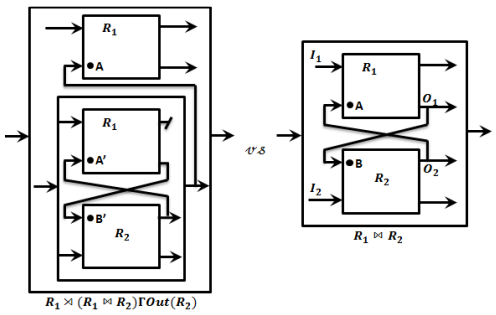

Figure 7. $R = R_1 \rtimes (R_1 \swarrow R_1)$ *when* $R = R_1 \bowtie R_2$

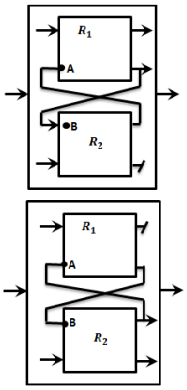

Figure 8. $(R_1 \swarrow R_2) \times (R_1 \swarrow R_1)$

It remains to show that if $R = R_1 \rtimes R_2$ then $R_1 \swarrow R_1 = R \restriction R_2^\bullet$ coincides with R_2, and indeed $R \restriction R_2^\bullet = ((^\bullet R_1 \setminus R_2^\bullet) \cup R_2)\,; (R_1 \cup R_2^\bullet) \restriction R_2^\bullet = ((^\bullet R_1 \setminus R_2^\bullet) \cup R_2) \restriction R_2^\bullet = R_2$ by Remark 7.1 and because $R_1^\bullet \cap R_2^\bullet = \emptyset$. □

Lemma 7.3. *$R_1 \leq R_2$ implies $R \bowtie R_1 \leq R \bowtie R_2$ whenever R_1 and R_2 are both components of R.*

Proof. By Remark 7.2 $(A, B) \in R \bowtie R_i$ if and only if there exists a finite sequence $A_0, \ldots, A_n$ such that $A = A_0 \in {}^\bullet R \setminus R_i^\bullet \cup {}^\bullet R_i \setminus R^\bullet$, $(A_{k-1}, A_k) \in R \cup R_i$ for all $1 \leq k \leq n$, and $B = A_n \in R^\bullet \cup R_i^\bullet$. Monotony of $R \bowtie -$ then follows from the fact that $R_1^\bullet = R_2^\bullet$. □

Lemma 7.4. *$R_1 \leq R_2$ implies $R_1 \swarrow R \leq R_2 \swarrow R$ whenever R is a component of both R_1 and R_2.*

Proof. $R_1 \leq R_2$ means that $R_1^\bullet = R_2^\bullet$ and $R_2 \subseteq R_1$. Then $R_1 \swarrow R = R_1 \restriction (R_1^\bullet \setminus R^\bullet) \leq R_2 \restriction (R_2^\bullet \setminus R^\bullet)$ because $R_1^\bullet \setminus R^\bullet = R_2^\bullet \setminus R^\bullet$ and $R_2 \subseteq R_1$. □

Proposition. *If R_1 is a component of R and R' is an interface then*

$$R \swarrow R_1 \leq R' \iff R \leq R_1 \rtimes R'.$$

Proof. By Proposition 5.1 and Lemma 7.3 we get $R \swarrow R_1 \leq R' \implies R = R_1 \rtimes (R \swarrow R_1) \leq R \rtimes R'$. The converse direction follows by Lemma 7.4 and Proposition 5.1: $R \leq R_1 \rtimes R' \implies R \swarrow R_1 \leq (R \rtimes R') \swarrow R_1 = R'$. □

Lemma 7.5. *If R_1 is a component of R then $R_1 \bowtie (R / R_1) = R$*

Proof. Since $(R/R_1)^\bullet = R^\bullet \setminus R_1^\bullet = (R \swarrow R_1)^\bullet$ and $R/R_1 \supseteq R \swarrow R_1$ one has $R/R_1 \leq R \swarrow R_1$ and by Lemma 7.3 $R_1 \bowtie (R/R_1) \leq R_1 \bowtie (R \swarrow R_1) = R_1 \rtimes (R \swarrow R_1) = R$. We are left to prove that $R_1 \bowtie (R/R_1) \subseteq R$. Let $(A, B) \in R_1 \bowtie (R/R_1)$ then by Remark 7.2 there exists a sequence $A_0, \ldots, A_n$ such that $A = A_0 \in {}^\bullet (R_1 \bowtie (R/R_1))$,

$B = A_n \in (R_1 \bowtie (R/R_1))^\bullet = R^\bullet$, and $(A_{i-1}, A_i) \in R_1 \cup (R/R_1)$ for all $1 \leq i \leq n$. One has ${}^\bullet(R_1 \bowtie (R/R_1)) = {}^\bullet R_1 \setminus (R^\bullet \setminus R_1^\bullet) \cup {}^\bullet(R/R_1) \setminus R_1^\bullet$. Thus $A \in {}^\bullet R$ because ${}^\bullet R_1$ and ${}^\bullet(R/R_1)$ are subsets of ${}^\bullet R$. There are three possibilities for each transition (A_{i-1}, A_i): *(i)* $(A_{i-1}, A_i) \in R_1$ if $A_i \in R_1^\bullet$, *(ii)* $(A_{i-1}, A_i) \in R_{\swarrow} R_1$ if $A_i \in R^\bullet \setminus R_1^\bullet$ and $A_{i-1} \in {}^\bullet R \setminus R_1^\bullet$, or *(iii)* $(A_{i-1}, A_i) \in R_{\nearrow} R_1$ if $A_i \in R^\bullet \setminus R_1^\bullet$ and $A_{i-1} \in R_1^\bullet$, Note that if the sequence contains no transition of the latter category then it witnesses that $(A, B) \in R$ due to the fact that $R_1 \bowtie (R_{\swarrow} R_1) = R_1 \rtimes (R_{\swarrow} R_1) = R$. We're going to gradually eliminate all transitions of type *(iii)*. For doing so let us consider the leftmost transition of this latter category if it exists. Thus i is the smallest index such that $(A_{i-1}, A_i) \in R_{\nearrow} R_1$. Since $R_1^\bullet$ is a subset of $R^\bullet$ and thus is disjoint of ${}^\bullet R$ we deduce that $A_{i-1} \neq A$ and thus $i - 1 \geq 1$. Now the sequence $\sigma : A = A_0 \to \cdots \to A_{i-1}$, which contains only transitions of types *(i)* or *(ii)*, witnesses that $A \in R^{-1}(\{A_{i-1}\})$. Since $(A_{i-1}, A_i) \in R_{\nearrow} R_1$ we deduce that $A \in R^{-1}(\{A_i\})$. Thus by replacing sequence σ by transition (A, A_i) we get a sequence with one less transition in $R_{\nearrow} R_1$ and thus we end up with a sequence with no transition in $R_{\nearrow} R_1$ witnessing that $(A, B) \in R$. □

Lemma 7.6. *If R_1 and R_2 are composable then $(R_1 \bowtie R_2)/R_1 \leq R_2$.*

Proof. Let R_1 and R_2 be composable interfaces, in particular $R_1^\bullet \cap R_2^\bullet = \emptyset$, and $R = R_1 \bowtie R_2$. Then $(R/R_1)^\bullet = (R_1^\bullet \cup R_2^\bullet) \setminus R_1^\bullet = R_2^\bullet$. $(A, B) \in R_2 \setminus (R_{\swarrow} R_1) = R_2 \setminus (R \restriction R_2)$ if and only if $(A, B) \in R_2$ (hence $B \in R_2^\bullet$, and $A \in {}^\bullet R_2 \cap R_1^\bullet$. Then necessarily $R^{-1}(\{A\}) \subseteq R^{-1}(\{B\})$ and therefore $(A, B) \in R_{\nearrow} R_1$. It follows that $R/R_1 = R_{\swarrow} R_1 \cup R_{\nearrow} R_1 \supseteq R_2$ and thus $R/R_1 \leq R_2$. □

Lemma 7.7. *$R_1 \leq R_2$ implies $R_1/R \leq R_2/R$ whenever R is a component of both R_1 and R_2.*

Proof. Recall that $R_i{}_{\nearrow} R = \left\{(A, B) \in R^\bullet \times (R_i^\bullet \setminus R^\bullet) \;\middle|\; R^{-1}(\{A\}) \subseteq R^{-1}(\{B\})\right\}$ and $R_i/R = R_{i\swarrow} R \cup R_i{}_{\nearrow} R$. $R_1 \leq R_2$ means that $R_1^\bullet = R_2^\bullet$ and $R_2 \subseteq R_1$ from which it follows that $R^\bullet \times (R_1^\bullet \setminus R^\bullet) = R^\bullet \times (R_2^\bullet \setminus R^\bullet)$ and thus $R_2{}_{\nearrow} R \subseteq R_1{}_{\nearrow} R$. The result then follows from Lemma 7.4 and $(R_1/R)^\bullet = R_1^\bullet \setminus R^\bullet = R_2^\bullet \setminus R^\bullet = (R_2/R)^\bullet$. □

Proposition. *If R_1 is a component of R and R' is an interface then*

$$R/R_1 \leq R' \iff R \leq R_1 \bowtie R'.$$

Proof. By Lemma 7.5 and Lemma 7.3 we get $R/R_1 \leq R' \implies R = R_1 \bowtie (R/R_1) \leq R \bowtie R'$. The converse direction follows by Lemma 7.7 and Lemma 7.6: $R \leq R_1 \bowtie R' \implies R/R_1 \leq (R \bowtie R')/R_1 \leq R'$. □