A formal approach for role-based modeling of business collaboration processes

Rodrigue Aimé Djeumen Djatcha

University of Douala
Postgraduate School for Pure and Applied Sciences
Postgraduate Training Unit for Mathematics, Applied Computer Science and Pure Physics
Applied Computer Science Laboratory

Thesis

A formal approach for role-based modeling of business collaboration processes

by
Rodrigue Aimé DJEUMEN DJATCHA

Directed by
Eric BADOUEL, Research Director, INRIA, France
Samuel BOWONG, Professor, University of Douala
Auguste NOUMSI, Senior Lecturer, University of Douala

Defended in fulfillment of the requirements for the degree of
Doctor of Philosophy (Ph.D) in Computer Science.

Date: December 15, 2022

Before the jury made up of:

President

Leandre NNEME NNEME, Professor, University of Douala

Referees

Alessandra AGOSTINI, Associate Professor, University of Milan-Bicocca

Georges E. KOUAMOU, Associate Professor, University of Yaoundé I

Jean Gaston TAMBA, Associate Professor, University of Douala

Members

Eric BADOUEL, Research Director, University of Rennes I (INRIA)

Samuel BOWONG, Professor, University of Douala

Auguste NOUMSI, Senior Lecturer, University of Douala

Academic Year: 2021 - 2022

ABSTRACT

Business collaboration defined as working together to achieve a common goal, is suitably specified in terms of where collaboration takes place (context), those who are involved (contributors), what they do both individually (skills) and collectively (production processes), and finally, what is the expected purpose of this collaboration (business goal). We assume a clear separation between skills and effective contribution to collaboration (i.e role). Technically, a context can the be defined by the set of roles involved, offering a formal basis for reasoning. Considering business collaborative processes, involved contributor's behaviors set is similar to a software components, possibly abstracted as role i.e services or group of services provided with their production rules. A collaborative context is then viewed as a user-centered service oriented architecture where every stakeholder could be assigned one or more roles. We address the problem of a formal design approach for such business collaboration processes in that context, facing components reuse and process management challenges as unambiguously describing business roles, role composition and dynamic collaborative process management.

Key words: Business Collaboration, Role-Based Design, Service Oriented Design, Software Reuse, Interface of Role, Guarded Attribute Grammar

RÉSUMÉ

La collaboration métier, définie comme le fait de travailler ensemble pour atteindre un objectif commun, se spécifie convenablement en termes du lieu de la collaboration (contexte), de ceux qui y sont impliqués (contributeurs), ce qu'ils font à la fois individuellement (compétences) et collectivement (processus de production) enfin, le but attendu de cette collaboration (objectif métier). On suppose une franche séparation entre la compétence et la contribution effective dans la collaboration. Techniquement, un contexte peut se décrire par l'ensemble des rôles en présence, offrant ainsi une base de raisonnement formelle. Considérant un processus de collaboration métier, l'ensemble des comportements de contributeurs, est similaire à un composant logiciel, pouvant s'abstraire comme un rôle i.e. des services ou des groupes de services fournis ainsi que les règles de production associées. Un contexte de collaboration s'apparente alors à une architecture orientée service centrée utilisateur où chaque contributeur pourrait se voir attribuer un ou plusieurs rôles. Nous abordons la problématique d'une approche formelle de conception, pour les processus de collaboration métier dans un tel contexte confrontés aux même défis que ceux de la réutilisation et le management de processus de composants, à savoir la description non ambiguë de rôles métiers, la composition de rôles et la gestion dynamique de processus.

Mots clés: Collaboration métier, Conception Rôle-Centrée, Conception Orientée Service, Réutilisation, Interface de rôle, Grammaire Attribuée Gardée.

ACKNOWLEDGMENTS

I would like with these words to express my deep gratitude and appreciation to Professor Eric Badouel, for his patience and his availability. A thanks also to Professor Claude Tangha of late memory for his advice; Finally, thanks to Professor Samuel Bowong and Doctor Auguste Noumsi for accepting the new co-direction of this work.

I would also like to say a special thank you to a number of natural and legal persons without whom the completion of this work would not have been possible:

- LIRIMA (International Laboratory for Research in Computer Science and Applied Mathematics),
- The FUCSHIA research team,
- The Faculty of Sciences of the University of Douala,
- My colleagues at the Department of Mathematics and Computer Science of the University of Douala.

Let all those who are not mentioned by name here, in particular brothers, friends and acquaintances, know that in some way they have supported me in the context of this work and receive all my gratitude.

DEDICATION

To the DJATCHA's

My late dad(Jean-Marc), mum(Louisette), all sisters and brothers;

AND TO the DJEUMEN's

Elisabeth Reine, Axel, Lucas, Noemie and Florin.

UNIVERSITE DE DOUALA

ECOLE DOCTORALE DES SCIENCES FONDAMENTALES ET APPLIQUEES

Unité de Formation Doctorale de Mathématiques, Informatique Appliquée et Physique Fondamentale

Laboratoire d'Informatique Appliquée
B. P. 24157 Douala
Tél : (+237) 697 91 65 57
Email : infos.ufd-miapf@univ-douala.com

THE UNIVERSITY OF DOUALA

POSTGRADUATE SCHOOL FOR PURE AND APPLIED SCIENCES

Postgraduate Training Unit for Mathematics, Applied Computer Science and Pure Physics

Applied Computer Sciences Laboratory
P.O. Box 24157 Douala
Tél : (+237) 697 91 65 57
Email : infos.ufd-miapf@univ-douala.com

DECLARATION SUR L'HONNEUR

Je soussigné, Monsieur DJEUMEN DJATCHA Rodrigue Aimé, Numéro d'inscription **210294524**, atteste que ce mémoire de thèse est le fruit de mon propre travail effectué au sein du Laboratoire d'Informatique Appliquée hébergé au sein de l'UFD en Mathématiques; Informatique Appliquée et Physique Fondamentale (MIP) de l'Ecole Doctorale des Sciences Fondamentales et Appliquées (EDOSFA) de l'Université de Douala sous la direction du Pr BADOUEL Eric, Pr BOWONG TSAKOU Samuel et Dr NOUMSI Auguste. Je soumets ce mémoire en accomplissement partiel en vue de l'obtention du doctorat /Ph.D, Option Informatique (Génie Logiciel).

Ce mémoire de thèse est authentique et n'a pas été déposé dans une autre Université.

L'AUTEUR

DJEUMEN DJATCHA Rodrigue Aimé

LES DIRECTEURS

NOUMSI WOGUIA Auguste V.
Chargé de Cours
Université de Douala

BOWONG TSAKOU Samuel
Professeur
Université de Douala

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Context

The dazzling development of ICTs in recent years has considerably contributed on the one hand, to develop and even create new types of behavior and, on the other hand, has broken down barriers geographically , economically, culturally, technologically, etc. Thus contributing not only to bringing together individuals, institutions and organizations in their daily functioning; but also to qualitatively improve their respective managerial approaches. Among the behaviors whose evolution is closely linked to ICT's evolution, we have business collaboration or simply collaboration. Etymologically, collaboration is defined as working together to achieve a common goal. Intuitively, it follows from this definition, that collaboration is suitably described in terms of (1) those who are involved (contributors) i.e. who they are and how to integrate them into the collaboration; (2) what they do both individually (skills) and collectively (production processes); (3) where collaboration takes place (context) and finally, (4) what is the expected purpose of this collaboration (business goal). In addition, depending on the purpose sought in the collaboration, there are two subcategories of collaborations [1, 2, 3]: collaboration for service delivery, in which functional rules are structured processes clearly defined a priori; and collaboration for co-creation, innovation and knowledge sharing, where the functional rules are unstructured processes defined as the collaboration is deployed.

Designing collaboration is a very complex exercise. The complexity here, is in-

herent to the heterogeneity of the four intrinsic elements of a collaboration, taken separately and collectively. Indeed, the context of a collaboration is strongly influenced among other things by contributors diversity, in terms of their individual skills; but also by the evolving nature of business goals, which in turn greatly influences production processes involved. In such a fairly dynamic context, two major almost recurring needs emerge; namely effectively taking into account the recurrent reorganization of work within collaborative structures on the one hand, and the possibility of incrementally integrating particularly heterogeneous sectors of activity in the collaboration, on the other hand. So a business collaboration design tool, must own dynamism, scalability and flexibility properties among others.

Motivations and Objectives

contributor's behaviors in a business collaborative process, are similar to software components, possibly abstracted roles i.e. services or group of services with associated production rules. A collaborative context is then similar to a user-centered service oriented architecture where every contributor could be assigned one or more roles. In this role-based design approach a role can be encapsulated by a module whose interface specifies provided services exported by the module and imported external services required. Usually, modules in service oriented design are hierarchically organized; rather the modules in role-based design approach often depend on one another (although cyclic dependencies between services should be avoided). In addition, the activated services can work as routines and a service call can activate new services in a way that may depend on user's choice on how the service is to be provided. Designing collaboration in the context above faces similar challenges as those of components reuse [4] and dynamic process management. A design activity, then requires the ability to discover services unambiguous choice a given service among potential services. Hence the motivation to have a precise description of tasks involved, a notion of a richer role interface and lastly means to dynamically manage and monitor collaboration evolution.

Contributions

Collaboration spectrum been wide and diverse choice has been made to focus on collaborative processes for service delivery. Thereby, our contribution has the form of three main extensions to GAG concept. Firstly the definition of a flexible tasking model. Secondly, the definition of composition and reuse mechanisms for role interfaces, inferred from tasking models specifications in GAG formalism. Thirdly, the introduction of a role-based design approach and its mechanisms, suitable to business collaboration design. More specifically, our contribution can be summed up with the three salient points below:

- A definition of a tasking model, for service delivery business collaboration: A business process model formally defining tasks, services, crowd tasks, and contributor's intrinsic skills involved and their orchestration as business skills needed to complete a collaborative job.
- A definition of an interface of role theory for collaboration: Here business skills are encapsulated component behaviors, described by interfaces, termed interface of role. In a contract-based service-oriented and user-centric context we design components interfaces and provides three main operations: interfaces composition specifying how components behaves to the environment, an implementation order for been able to state when a component satisfies an interface, finally a residual operation checking systems needed for realizing a global specification, when composed with a given component.
- A definition of a role-based design approach for collaboration: In this approach a clear separation is made between contributor's intrinsic skills and his explicit contribution (role) in a given collaboration. A role is then viewed as a particular concern of a domain. In a dynamic context roles involved offers a formal abstract basis for reasoning; but traditional role-based systems lack flexibility, because only role static description mechanisms are available. We introduce improvements by providing various mechanisms for business skills dynamic choreography, role switching, workflow monitoring and checking.

Publications

(1) A Stable and Consistent Document Model Suitable for Asynchronous Cooperative Edition [5, 6]: Complex structured documents can be intentionally represented as a tree structure decorated with attributes. Ignoring attributes in the context of a cooperative edition (these are related to semantic aspects that can be treated separately from purely structural aspects on interest here). Legal structures are characterized by a document model (an abstract grammar) and each intentional representation can be manipulated independently and possibly asynchronously by several co-authors through various editing tools that operate on "partial replicas ". For unsynchronized edition of a partial replica concerned co-author must have a syntactic document local model that constraints him to ensure minimum consistency of local representation with respect to the global model. This consistency is synonymous with the existence of one or more (global) intentional representations towards the global model, assuming the current local representation as her/their partial replica. The purpose of this work is to present grammatical structures, which are grammars permitting not only to specify a (global) model for documents published in a cooperative manner, but also to derive automatically via a so call projection operation, consistent (local) models for each co-authors involved in the cooperative edition. Some properties that meet these grammatical structures are also shown.

(2) Modular Design of Domain-Specific Languages using Splittings of Catamorphisms [7]: Language oriented programming is an approach to software composition based on domain specific languages (DSL) dedicated to specific aspects of an application domain. In order to combine such languages we embed them into a host language (namely Haskell, a strongly typed higher-order lazy functional language). A DSL is then given by an algebraic type, whose operators are the constructors of abstract syntax trees to which one can associate a polynomial functor. Algebras (respectively co-algebras) express how to evaluate/interpret/execute programs (resp. generate/construct programs) where

programs are viewed as abstract syntax trees. Using Bekîc theorem we define a modular decomposition of algebras that leads to a class of parametric abstract context-free grammars, associated with regular functors, allowing for the modular design of domain-specific embedded languages.

(3) A Calculus of Interfaces for Distributed Collaborative Systems: The Guarded Attribute Grammar Approach [8, 9] 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 modeled by Guarded Attribute Grammars. Following the contract-based specification of components we develop an approach to an interface theory for the components of a collaborative system in three stages: we define a composition of interfaces that specifies how the component behaves with respect to its environment, 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.

(4) A role-based collaborative process design on crowdsourcing systems [10]: Crowdsourcing is a collaborative business process model, in which tasks are carried out by a crowd. In crowdsourcing systems, there are two types of stakeholders namely, requesters who outsources tasks and the crowd, or contributors, performing those tasks. We consider a stakeholder as an actor, or a standalone software component, evolving on a platform and having both mechanisms of interaction with its environment and business skills. A set of stakeholders interacting in a dynamic context for solving a problem is a distributed collaborative system and we term it crowdsourcing system. In such a system the role concept is central, because each stakeholder must have a specific framework within which he collaborate. Traditionally, collaborative systems lose flexibility if their design is role-based, because only static role description mechanisms based on intuitive concepts are available. We propose in this paper an improvement consisting of four things: (1) defining clearly what an outsourceable task or crowd task is, (2) specifying roles clearly and rigorously, while ensuring flexibility for collaboration, (3) providing role switching mech-

anisms, and (4) providing an abstract basis for crowdsourcing system design and workflow monitoring and checking mechanisms, for potential activities, dynamically carried out by a system.

Document organization

The remainder of this document, is organized as follow:

- Chapter 1: devoted to the presentation of fundamental theoretical concepts necessary and useful in this work, as role and contributor concepts, business process modeling (BPM), system design approaches and principles, guarded attributes grammars (GAG).
- Chapter 2: presents principles, classifications, and mechanisms of business collaboration. A general tasking model for service delivery collaboration, described in GAG formalism, is also provided.
- Chapter 3: presents an extension made to GAGs, by defining the role interface notion and associated mechanisms, useful for components composition.
- Chapter 4: introduces a role-based approach, the design of service delivery business collaboration processes.
- Conclusion: to this work, with future perspectives and challenges.
- Appendix A: provides implementations of main concepts presented in this work, in the purely functional programming language Haskell[1].
- Appendix B: lists all publications made in the context of this work.

[1] https://www.haskell.org

CHAPTER 1

RELATED WORKS AND USEFUL BACKGROUND THEORIES

1.1 Introduction

This chapter introduces some founding basic key concepts, useful for various ideas developed in the next part. Specifically, we will introduce concepts of role and contributors of a dynamic system, as the desired approach in this work is role oriented. Then business process modeling will be presented with an emphasis on it definition, taxonomy, characterization and its usual mechanisms, since business processes design in a dynamic environment is also a major concern. Next, focus will be made on system design principles. Finally, we introduce guarded attribute grammar, the formal tool capturing characteristic principles of a user-centric collaboration.

1.2 Roles, contributors and mechanisms

1.2.1 The role approach

The concept of role is crucial in any collaborative or cooperative work system. Importance here is related to the fact that, each stakeholder or contributor in this type of system, must have a clear framework within which he collaborates with others. In this context, the role specifies both what the system expects from the contributor,

but also what the contributor expects from the system; preventing a contributor from being overwhelmed by information (or tasks) that is not necessary. In practice, the role-based approach is variously used, such as in UML design (use case diagram, class diagram, etc.), access controls modeling [11, 12], and collaborative or cooperative systems modeling such as a company, multi-agent systems [13, 14], business processes [15], social media, etc.

1.2.2 Common definitions and terminologies

Role and role type

A role type is the perception that one contributor has of another contributor [16]. A role type is specified uniquely and a contributor plays at a given time a role specified by a role type. A role is therefore defined as one of the instances of a given role type, played by a contributor. This separation between contributors (i.e. the one that intervenes in the system), and a role (i.e. what an actor does in the system), brings dynamism to the system described as a dynamic role [17].

Foundation and semantic rigidity

Depending on the case, it may not always be easy to differentiate between a role and a contributor. For instance, consider the use case below :

Use case 0. According to laws in Cameroon, civil-status certificates are issued by a civil status officer (csOfficer). Depending on the context, a user could apply to a Mayor, if he is locally based in the country; or rather to the embassy if he is abroad.

Mayor and csOfficer entities, are part of the business collaboration within an organization. Being able to distinguish between these entities, which is role type (or simply role) and which is contributor, can be rather complex. The sharp distinction between a contributor and a role type is based on both concepts of foundation and semantic rigidity [18].

Definition 1.2.1 (Foundation & Rigidity). An entity is considered to be founded, if its specification implies a dependency or relationship with another entity. While an

entity is semantically rigid, if its identity depends on certain characteristics, and can not exist without them.

Thus, it will be said that Mayor is unfounded and semantically rigid (and therefore Mayor is a contributor), whereas csOfficer is founded and semantically non-rigid (csOfficer is a role). In other words, a Mayor plays the csOfficer role. Hence a contributor is unfounded and semantically rigid, while a role is founded and semantically non-rigid. From the above we consider in this work that concepts of role type and role are equivalent, and will simply be called role.

1.2.3 Constraints on roles

Considering a contributor and his potential roles, constraints can be defined over those roles, as predicates which if necessary, return the value "acceptable" or "not acceptable", applicable to the play relationship between contributor's roles. Let r_0 and r_1 be two roles, four constraints can be defined over those roles [16, 19]:

1. Dcr (Don't care) for no constraints on contributors playing each of the roles.
2. $Impl(r_0, r_1)$ (Prerequisite or Implied) a contributor playing a role r_0 should always be able to play a role r_1 (the reverse is not always necessary).
3. $Eqv(r_0, r_1)$ (Equivalence) the implied constraint is checked for both the pair (r_0, r_1) and the pair (r_1, r_0).
4. $Phb(r_0, r_1)$ (Prohibited or mutual exclusion) a contributor playing a role r_0 never plays a role r_1 and vice versa.

1.3 Business processes Modeling

1.3.1 Processes and workflows

A process is a set of repeatable activities, that need to be carried out by contributors in order to accomplish some sort of organizational goal. Depending on the context a process can be structured or unstructured [20]. Structured processes (or static processes) are those processes unchangeable in form, as well as those which change

over a long period of time. Their structure is known beforehand and are completely automatized. Conversely, unstructured processes (or dynamic processes) are processes the course of which is dependent on individual condition of execution or which contain such a large amount of variables that it is impossible to model them. They requires to factor in at design the possibility of process contributors making individual decisions that are not able to foresee before hand and integrating contributor's knowledge in design and improvement of processes.

A workflow [21] is a series of repeatable activities needed to be carried out to perform a task. A workflow is developed in support to a process and describes how to get a process accomplished. A workflow is equivalent to a process if, and only if, the task carried out by that workflow is an organizational goal. A workflow gather several elements as [22, 23, 24]:

(1) conditions: which decides when a given action should be performed;

(2) activities: which determines the functionality of the workflow. Each activity performs a different task, can succeed or fail;

(3) activity content: being the different information units as data, object, documents, artifacts, etc. Content help articulating workflow progression;

(4) sequence activity: that is, part of multiple activities of a workflow, useful for organizing several successive activities into one major step, and implemented as a function;

(5) decision nodes: which enables a workflow to make a selection on the execution path to follow, it behavior can be seen as a switch-case statement.

1.3.2 Business processes

A business process is a set of organized activities or tasks potentially performed in an environment or organization. Those activity jointly realize a business goal as delivering values for internal or external contributors. Each business process is enacted by a single contributor but it may interact with other business processes performed by other contributors [25, 26, 27]. A business process is modeled by explicitly defining it's two components, in particular a related workflow describing

the targeted process and associated contributors with respect to understandability and maintainability properties [28].

1.3.3 Dynamism in Business Process

The dynamism of a business process is the ability to react to changing conditions (internal or external) of operation, according to needs, in a appropriate timely manner, at process instance runtime without having a negative impact on the process existence or its expected completion [23, 29]. A dynamic business process can then be defined as a business process whose related workflows components and contributors may vary and, if necessary, change with low latency at runtime due to changes of the context [24].

Technically there are two ways perception of process dynamism [29]. On one hand, horizontal perception or implementation perspective where the process is viewed at different stage of it life cycle (execution, simulation, automation). On the other hand, vertical perception or layered dynamism as the ability of a business process to adapt to changes in the environment. From the last perception, there are three dynamism implementation levels:

(1) Using decision points: human or automated systems decides what to do next, according to predefined rules (a user-centric approach).

(2) Automatic configuration of business process: construct a process from a number of reusable fragments, depending on conditions change.

(3) Goal-driven business process: constructing a business process from a number of possible and reusable tasks at runtime.

1.4 System design principles

1.4.1 Separation of concerns

1.4.1.1 Definitions and goal

Separation of concerns (SoC) [30] is an abstract concept of system decomposition, difficult to define. The difficulty here is mainly due to the complexity of the targeted

system, but also due to the exact definition of what is a concern in that system. Craftsman [31] defines separation of concerns as a delineation and correlation of system elements, in order to establish a well organized system where each part fulfills a meaningful and intuitive role while maximizing its ability to adapt to change. It follows from this definition that a concern as part of the target system, is a set of clearly defined responsibilities making it possible to achieve a single purpose; but in practice, not all concerns are easily separable and they are termed cross-cutting concerns. SoC is an important design principle in many areas such as urban planing, architecture and information design. Well achieved, it promotes a better understanding, design and management of complex interdependent systems, so that parts can be reused, optimized independently of other parts, and insulated from the potential failure of other parts.

1.4.1.2 SoC mechanisms, principles and properties

According to SoC definition above, three interrelated and complementary mechanisms emerges: (i) delineation i.e. the action of indicating the exact position of a border or boundary. In practice, boundaries might be methods (functions), objects, components, services, etc. Delineation mechanism ensures exclusivity and singularity of purpose and boundaries between elements. (ii) Correlation i.e. mutual relationship or connection (interface) between concerns and (iii) adaptation to change (evolutivity). Those mechanisms are captured by the single responsibility and interface segregation principles in Martin's SOLID principles [32, 33, 34, 35, 36] as:

(1) Single responsibility (Cohesion and Coupling) [37]: A concern should have one and only one reason to change, and should be isolated from complexities of the system as a whole. Changes can only originate from a single concern, or rather a single tightly coupled group of concerns, representing a single narrowly defined business goal. So each concern or group of concerns is responsible of one business goal. The single responsibility principle, details the cohesion and coupling mechanism as the goal is to gather together concerns that change for the same reasons, and separates those changing for different reasons.

(2) Interface segregation (Scalability)[38]: Systems evolve by dynamically redefining interfaces involved. For that purpose, interface segregation principle, recommends the implementation of several specific interfaces, instead of one general purpose interface. However, if such an approach considerably reduces the frequency of necessary changes to the system, it could cause side effects if, while designing interfaces, not only useful methods are expressed each time.

A concern gathers several properties [39] as: canonicity, composability, reusability, adequation and closure, wich in turn guarantee maintainability, robustness, adaptability and evolutivity properties to the system.

1.4.1.3 Soc design approaches

Systems must usually satisfy multiple properties, perform multiple functions simultaneously, and satisfy multiple purposes [40]. With that high level complexity, the main difficulty in designing those systems, is achieving separation of concerns; Various separation techniques exists [41, 42, 31]: (i) horizontal separation or layered separation, i.e grouping processing concerns based on their role within application. The process is to divide an application into logical layers of functionality that fulfill the same role within the system. This is an organization of concerns which minimizes the level of dependencies with application. (ii) Vertical separation i.e dividing an application into modules of functionalities that relate to the same feature or subsystem within an application. This clarifies the responsibility and dependencies of each features. Boundaries may be defined logically to aid in organization, or physically to enable independent development and maintenance. (iii) Aspect separation i.e. segregating an application's cross-cutting concerns [43] or aspects, from its core concerns. This approach is known as aspect-oriented programming [44].

1.4.2 Service Oriented Design

1.4.2.1 Definitions, principles and properties

Service oriented design (SOD), is a particular client/server design approach where architecturally, design units are services. Each meaningful service is implemented as an autonomous component, sufficiently equipped to work alone for a desired goal.

Components publish services availability in a repository and make service calls in a peer-to-peer manner [45, 46, 47, 48]. This could be abstracted as an integrated business application where services represent steps of a business process and one the main application is the composite business application, also viewed as a distributed computing environment. Properties gained from this approach are: (i) ability to use or combine services in ways not conceived by their originators (conjonctivity), (ii) ability to deploy or reuse a component in any compatible environment (portability, deployability), (iii) possibly redundant networked resources (availability, failure handling), (iv) ability of components from different sources to use each others services (interoperability) [49].

1.4.2.2 SOD architecture

Service Oriented Architecture (SOA) is a distributed computing environment, organized around five main elements [50, 49, 51, 52]:

(1) Context: i.e. an environment for deploying plug and play components, prescribing details of installation, security, discovery and lookup;

(2) Component: reusable element providing services independently to platforms, protocols and context;

(3) Container: an environment executing components that manages availability and code security;

(4) Contract: an interface, that contractually defines the syntax and semantics of single behavior;

(5) Connector: autonomous element encapsulating transport specific details for a specified contracts.

1.4.2.3 SOD interactions

Interactions describes how, through the connector, components make their services available or request those of others and how, collaboration takes place between components. Concretely there are two interaction pattern forms: communication

pattern and services composition pattern elements of Service Oriented Computing (SOC) [53].

Communication pattern These patterns describes mechanisms used to publish and discover services. Service subscribers should possibly express their interest to a service event or may be a service event pattern and be notified when they occur, i.e published [54]. Technically three main implementations exist presented below, the least developed to the most elaborate:

- Messaging: subscribers send asynchronous messages through a common canal to publishers who processes the service synchronously.
- Remote Procedure Call (RPC): subscribers make asynchronous blocking service call which are synchronously processed by publishers.
- Publish/Subscribe: it is an event oriented style with total de coupling in time i.e. stakeholders are not supposed to take part actively in the interaction at the same time, space i.e. stakeholders does not know each other, and synchronization i.e publishing and subscribing are not locked parallel events. There could be many pub/sub categories: topic-based, content-based, type-based [55].

Services composition pattern Describes those patterns used to select and bind services, for a particular goal. In a layered architecture context, vertical service composition takes place at all layers of the architecture, it is a component deployment; rather horizontal service composition take place at the same layer. Composition can be static when occurring at design step (i.e service orchestration), or dynamic when occurring at execution, then services are substituted on demand, without rebuilding the whole system, providing more flexibility to the system (service choreography) [47, 56].

- Service orchestration: is a behavior that a service provider performs internally to realize a service that it provides. It describes tasks that a service provider performs internally.

- Service choreography: it is a goal focus approach, describing common goal interactions and tasks (flow relations) among service providers and between service providers and service requester. It does not describe sub-tasks performed internally, because those tasks are not essential to collaboration. Finally, it covers the perspective of a stakeholder that wishes to have an overview of a collaboration.

1.4.3 User-Centered design

A User-Centered Design is an iterative design process framework that incorporate validation from the user every step of the way [57]. Thereby, it is an optimistic approach to invent new solutions [58] by designing from user's perspective. This closeness and frequent interaction, helps understand user's needs.

1.5 Guarded attribute grammar

1.5.1 Concepts and principles

Guarded attribute grammar (GAG)[59] is a formal framework, inspired from Knuth's attribute grammar concepts [60, 61, 62], also viewed as a process grammar [63], and designed to model open user-centric architectures where data and rules requires flexibility. GAG help designing business processes by allowing dynamic task creation and data handling. Tasks competitive execution is implemented by a declarative approach where task dependencies are specified without any order.

The GAG system offers a process model with three levels of stratification [64]:

(i) the business services (or tasks) level which identifies all of the existing tasks (known or inferred) in the target domain; this level does not evolve much, it is these tasks that act on the artifacts, according to the rules;

(ii) the business rules level, used to aggregate services together in a declarative language;

(iii) the business artifact level which offers the ultimate stage of abstraction, by combining data and processes.

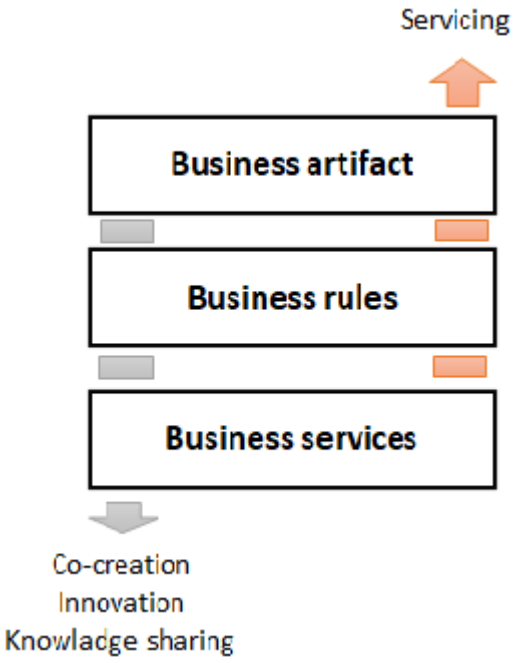


Figure 1.1: GAG layered architecture with associated goals

By their layered and flexible architecture (figure 1.1), GAGs helps modeling the two main forms of business collaboration, namely collaboration in order to bring out new knowledge and innovate (section 2.3), and collaboration for service supplying (section 2.4).

1.5.2 Formal definitions

1.5.2.1 Grammar and derivation relation

Definition 1.5.1. (Grammar): A Grammar $G = (S, P)$ is given by a set of grammatical symbols S and a set of productions $P \subseteq S \times S^*$ expressed as $rule_i : s_0 \to s_1 \cdots s_n$ ($i \in \mathbb{N}$), $rule_i$ is production label, symbol s_0 is the left-hand side (lhs) and $s_1 \cdots s_n$ is the right-hand side (rhs).

A symbol is said to be used, (respectively defined) when it appears in the right-hand side (resp. left-hand side) of some production. We let ${}^\bullet G$ and $G^\bullet$ denote respectively the set of used but not defined symbols, and the set of defined but not used symbols respectively.

Definition 1.5.2. (Derivation relation): We let the derivation relation $\to \subseteq S^* \times S^*$ given by $w \to w'$ iff exist $u_1, u_2 \in S^*$ and $(X, u) \in P$ such that $w = u_1 \cdot X \cdot u_2$ and

$w' = u_1 \cdot u \cdot u_2$; and we let $\rightarrow^*$, be its reflexive and transitive closure.

1.5.2.2 Reduced grammar

Let $\sharp_B(u)$ denotes the number of occurrences of symbol B in word u.

Definition 1.5.3. (Reduced grammar): A grammar is reduced if

(i) every symbol is accessible:

$$(\forall B \in S)\ (\exists A \rightarrow^* u) \quad A \in G^\bullet \ \wedge \ \sharp_B(u) \neq 0$$

and,

(ii) for every symbol A exists a derivation $A \rightarrow^* u$ leading to a word u all of whose symbols are in ${}^\bullet G$.

1.5.3 Data, Variables, Attributes and Guards

Data are arbitrarily typed piece of information needed to complete the execution of a task or produced as the result of the execution of a task. The GAG model associates data to pending task, using attributes. Each pending task is associated with two types of attributes, namely inherited attributes (inh) as incoming data to pending task context and synthesized attributes (syn) produced during task resolution and used by the context.

Each attribute a_i is associated with a term or pattern and given by an equation of the form $a_i = t_i$, where t_i is a data record, simply presented in a functional [65] manner and can be recursively defined by equations 1.1 below

$$t_i \ = \ \underline{c}(t_1, \ldots, t_n) \mid v \mid \text{cte} \tag{1.1}$$

where $\underline{c}$ is a data constructor or record, $t_1, \ldots, t_n$ are subterms or field of the record, v is a variable; variables in terms represent holes or place-holders which identify parts of the term that remain to be filled. Finally cte is a constant value of a given basic type which can be identified with zero arity data constructor. A term is said to be closed, if it contains only data constructors, filled with constant values

carrying some meaning in the context in which they are used.

We let $Var(t)$ be the set of variables occurring in term t, then t is closed iff $Var(t) = \emptyset$. We also let $p : s_0 \to s_1 \cdots s_n$ be a business rule, with $inh(s)$ and $syn(s)$ being respectively inherited and synthesized attributes associated to symbol s; then we define

$$\begin{aligned} Var_{inh}(s) &= \{x \in Var(t_a) \mid a \in inh(s)\} \\ Var_{syn}(s) &= syn(s) \\ Var(s) &= Var_{inh}(s) \cup Var_{syn}(s) \\ Var_{inh}(p) &= Var_{inh}(s_0) \\ Var_{syn}(p) &= Var_{syn}(s_0) \\ Var(p) &= \bigcup_{\forall s \in lhs(p) \cup rhs(p)} Var(s) \end{aligned} \tag{1.2}$$

as well as a guard (condition affecting inherited information) which specifies when production is activated.

1.5.4 Artifacts

An artifact is an abstract structure obtained by combining data and operations (processes), so as to form a monolithic block. Offering as advantages, several levels of abstraction, modularity and flexibility in the composition of business operations. The description of an artifact first requires a data model that describes the internal structure of business objects; and secondly a model life cycle which describes the possible paths and timing for the execution of these objects.

GAGs are artifact-centric as they offers a high level of abstraction, while promoting varied and rich communication between the stakeholders of the system in the performance of an activity. The models obtained by this approach is said to be actionable [64, 66], i.e. can be used to automatically generate executable.

Formally, an artifact is seen as a tree with typed nodes termed as $X :: s$ where X is the node of type s. Each node associated with the left hand side of a grammar

rule. An artifact is given by a set of equations of the form $X = P(X_1 \cdots X_n)$ indicating that the node $X :: s$ is labeled by the production $P : \ s \to s_1 \cdots s_n$ and having as successors $X1 :: s1 \cdots Xn :: sn$ respectively; and two equations cannot have the same left hand side. There are two types of nodes, namely open nodes or tasks being resolved and closed nodes or resolved tasks.

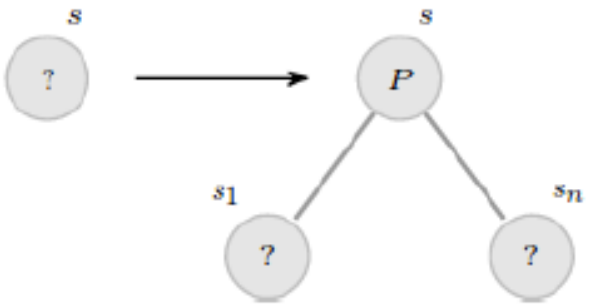


Figure 1.2: Artifact construction by rewriting rules

A closed node is a node labeled with the name of the rule that made it possible to carry out the associated task. It is described by an equation of the form $X = R(X_1 \cdots X_n)$ where $X :: s$ and $X_i :: s_i$ with the underlying rule $\mathcal{U}(\mathcal{R}) = s \to s_1 \cdots s_n$. A closed node state that task s is completed using rule P, as shown on figure 1.2.

An opened node is a $X :: s$ node associated to a pending task for which, performing rules are not yet known, and given by an equation on the form $X = s(t_1, \ldots t_n)\langle u_1 \ldots u_m \rangle$ where s is the pending task, $t_1, \ldots t_n$ are inputs and $u_1 \ldots u_m$ are outputs; we say an opened node is a form to fill out later. On figure 1.2, $X1 :: s1 \cdots Xn :: sn$ are opened nodes.

The life cycle of an artifact is implicitly given by the set of all productions. Initially, an artifact is reduced to an open node. This open node can then be refined by the choice of an appropriate rule or rewrite rule. An artifact will be said to be closed when it no longer contains open nodes. Task invocation, is done by filling the inherited positions (entries) of the form and, by indicating variables which will receive the results after task execution, we talk of variable subscription as shown on figure 1.3 above.

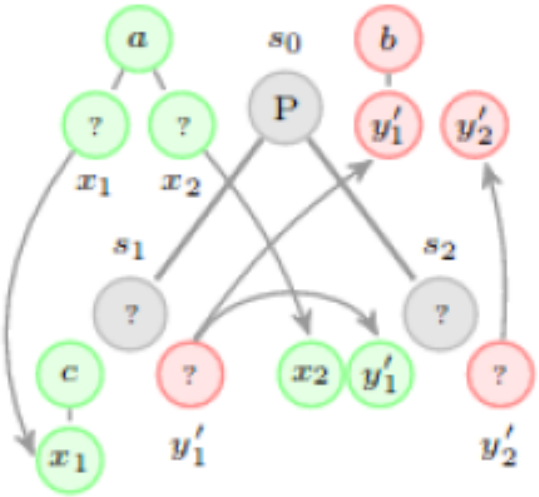


Figure 1.3: Data handling by an artifact

1.6 Conclusion

This chapter presented the founding elements of this work, that is the role approach and its mechanisms, business process modeling, systems design principles and approaches, with the separation of concerns and service-oriented design. But also, the formal modeling framework that are the GAGs, used in this work was introduced. The next chapter will complete the basic concepts, by describing the two various forms of business collaboration.

CHAPTER 2

BUSINESS COLLABORATION: PILLARS, TAXONOMY, USE CASES AND TASKING MODELS

2.1 Introduction

A business collaboration or more simply collaboration is the process of two or more people or organizations working together to complete a task, or to achieve a shared goal. Collaboration allows diverse skill sets both internally and externally, to be harnessed together, to set comprehensive content strategy designed to achieve business goals. As a process, a collaboration is completely defined by four elements: (1) its process organization, i.e the way work will be carried out; (2) those involved namely contributors; (3) the reason for the existence of collaboration or business goals; (4) the context of collaboration.

Collaboration and its deployment environment is a distributed collaborative system (or simply collaborative system) as it is a set of independent entities or contributors either moral, human, machine or software components, providing precise services each individually. Every contributor in the system has personal intrinsic skills, i.e he is equipped with various mechanisms, namely for data storage, for communication (by message, Restful like, SOAP like,...) with his environment, for

service discovery, sensing, etc. A contributor, by using a contextual assigned business skills, also plays a specific role in the accomplishment of some system business goal. In collaborative systems contributors can enter or go out from the system as they pleased. Same services occurrences can be provided by several contributors in the system. This chapter is devoted to business collaboration, for characterization and classification purpose, and to deduce both a formal definition,properties and describe associated tasking models for collaboration.

2.2 Pillars, criteria and strategies

2.2.1 Pillars of collaboration

A collaboration deployment process relationies on five pillars, namely: requester, contributors, tasks, interactions and outsourcing [47, 67].

2.2.1.1 Requester

The requester is a natural or legal person, who requests the power and wisdom of a contributor or a group of contributors, for the performance of a given service. To this end, the requester, among other things, can encourage contributors by gratification or social motivation (public recognition), outsource tasks, check the compliance of results with predefined standards and finally ensure project contributor's data confidentiality.

2.2.1.2 Contributors

A group of contributors is a community of actors or persons, taking part in a collaborative activity. A group of contributors must gather these four characteristic properties:

(1) Diversity: expressed here in terms of expertise (expertise and variety of skills), spatiality (geographic and disparate profiles), gender and age of contributors, etc.

(2) Anonymity: because contributors of a project, do not necessarily know each other, and do not necessarily know the requester.

(3) Importance: in terms of contributors numbers.

(4) Completeness or adequacy: i.e. contributors gather skills necessary to complete a task. However, an abundance in this context leads to overload, confusion and management difficulties.

2.2.1.3 Tasks

Within the framework of collaboration, three types of tasks can be distinguished, in particular primitives, services and outsourced tasks.

- Primitives are basic skills or intrinsic skills (or basics know how), acquired (or developed) by (for) contributors. They are atomic tasks. These skills cover various types of activities namely, communication, geolocation, object assessment, voting, etc. By convention, all primitives names are written in bold capital letters and, table 2.1 below summarizes some common primitives.

Primitives	Purposes
SEND	for sending data
RECEIVE	receiving data
INPUT	input data or fill a form
SNAP	get pictures
LOCATION	geo-localize a place
CONTEXT	describing an object context
TRANSCRIPT	record data in a register
SIGNUP	put a signature
CORRECT	update previous data
...	...

Table 2.1: Some usual primitives

- Services are (de)-composable pieces of work, possibly described in terms of primitives, outsourced tasks and services. A service is a goal or a targeted functionality offered by a role (by convention, a service names starts with a lower case letter).

- Outsourced task or crowd task is a decomposable service described only in terms of primitives. They are those types of services, requiring only contributor's intrinsic skills. When contributors are massively used to perform the same task within an activity they are called a crowd, and that task is a crowd

task. Crowd task is an activity in which the crowd participates. This activity can be a large-scale data collection, a co-creation task, or an innovation task. By convention, their names are overlined. The association mechanism between a crowd and a crowd task is called outsourcing. A crowd task can be characterized by the five major properties defined below:

(1) Modularity i.e. a crowd task can be either atomic or decomposable.

(2) Complexity i.e. level of difficulty in carrying out this task. The complexity here is different from modularity in that a task can be complex, while remaining atomic.

(3) Solvability i.e. a crowd task is simple enough to be solved by humans, but complex for computers.

(4) Automaticity i.e. a crowd task is either difficult to automate, or expensive in automation.

(5) Contributor-centric i.e it is controlled and carried out by contributors.

2.2.1.4 Interaction scheme

An interaction represents the existing pairwise relation between requester, contributors and task in a collaborative system in order to perform a given goal. It is an explicitness of collaboration between those entities. Different forms of interaction are distinguished: between requester and contributors termed tasking relation, between tasks and requester or requesting relation, between tasks and contributors or supplying relation and finally between tasks or dependency relation.

An interaction scheme is a description of all interactions existing in a given collaborative system. To this end, there are two levels of interaction description: first level, describing interactions between required services (requested by requesters) and provided services (supplied by contributors) termed service choreography. Second level, describing how contributors render their services and term service orchestration.

2.2.2 Skills orchestration strategies

Skills orchestration consists in tasks distribution to contributors, aggregating results and resuming the process based on the partial result obtained. According to task complexity, results acceptance and gratification granted criteria; there are mainly 3 strategies for orchestrating collaborative activities namely market, contest and auction strategies [68, 69]:

2.2.2.1 Market strategy

Contributors freely participate to tasks, according to their skills and gratification is not very important. This strategy is suitable for limited complexity micro-tasks (photo marking, text translation, etc.) for which a large number of responses are expected. Here acceptance criterion for results is simple and is mainly based on statistical methods.

2.2.2.2 Contest strategy

The requester has a task to perform, funds to allocate, but does not have acceptance criteria for the result. Contributors are invited to participate in a competition, proposing their solutions to this problem. According to a well-established criterion (number of contributions, deadline, ...), requester chooses the best solution and the winner; this strategy is suitable for creative tasks.

2.2.2.3 Auction strategy

Requester publishes his needs, contributors express the respective amounts of the gratuity expected for this work, and the best bidder is chosen to do the work. This strategy is suitable when the acceptance criteria for the solutions are known, but we cannot estimate the gratification to be associated with them.

2.2.3 Classification criteria for collaboration

2.2.3.1 Process organization

In collaboration organization or governance is the internal mechanism by which the production process is structured; it is also seeing as the distribution of decision-

making power, on the evaluation of work carried out, choice between proposals for solutions, ideas, work orientation. In practice, there are two governance modes: hierarchical governance mode and flat governance mode:

- Top-down mode or hierarchical governance: in this mode, to supply a given service, related tasks are assigned in an hierarchical manner progressively down to the base, and results are also assembled until the complete delivery of the desired service. Such approach is said to be a top-down hierarchical process and control rules are pre-established, control is operated at different stages of the system.
- Bottom-up mode or flat governance: by this mode, no control rule is established in advance. It consists in being able to provide a required service without prior organization. So it is according to needs that a task is carried out and results obtained are assembled progressively. Such an approach is said to be basic bottom-up process.

2.2.3.2 Participation

Participation defines properly the way contributors enter the collaborative activity; it can possibly be closed or opened.

- Closed participation: In this case, according to some system needs, contributors are explicitly invited, based on their skills. This casting approach provides a form of homogeneity of skills in the system, since contributors are chosen such grouping their skills, may cover the needs of entire targeted collaboration. This form of collaboration is also said to be internal collaboration [70, 71].
- Opened participation: contributors can freely access the collaborative process according to their feelings, skills and knowledge; in that context, skills can be hardly heterogeneous in the sense that possibly, there may be several occurrences of the same skills; but also, the set of skills involved may not cover the collaboration needs. This form of collaboration is also said to be external collaboration [71].

2.2.3.3 Business goal categories

As stated earlier, a collaboration is orchestrated for a given goal. Technically, there are three main goal categories for collaboration. Namely service supplying, knowledge sharing and co-creation (innovation). However, it should be noted that a choice made on a goal type for a collaboration also influences the choices in organization and participation criteria.

- Service supplying: process is clearly defined and structured, so organization is top-down; but participation could either be closed or opened.
- Knowledge sharing: organization could be top-down, as in learning case, where some contributors teach others; rather could also be flat as in social media. participation could either be closed or opened.
- Co-creation and innovation: organization is flat, contributors jointly define rules as things progress. participation could either be closed or opened.

2.2.3.4 Types of collaboration

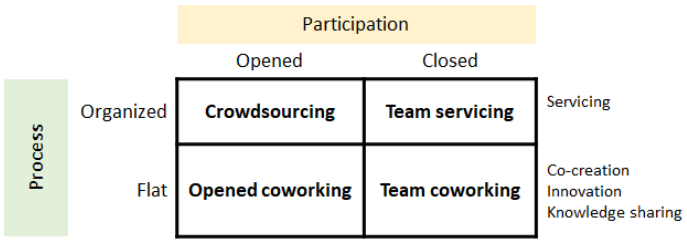


Figure 2.1: Business collaboration taxonomy

Depending on above organization criteria (section 2.2.3.1), two business collaboration families emerge [1, 2]. When there is no pre-established organization, and rules are defined by needs, it is a coworking. Rather, when production is hierarchically organized i.e. the business process is well defined, we talk of service supplying collaboration (see figure 2.1 above).

2.3 Coworking

Coworking is an arrangement in which several contributors from different area, share a common space [72, 73]. There is no pre-established rules, and contributors are business partners as they freely contribute to an activity. When participation is closed, we talk of team coworking, rather when participation is opened, it is a opened coworking.

2.3.1 Team coworking

Team coworking is a form of collaboration, where members are chosen related to their skills. It is a form of collaboration in which contributors jointly choose a problem, determine the organization of work and select results. This type of collaboration is suite for cases where an organization has a case, but doesn't know how to manage it, as in innovation and co-creation cases.

2.3.2 Opened coworking

Opened coworking or communitarianism is a case of a crowdsourcing for innovation, as an opened community where members freely join. It is a form of collaboration in which everyone proposes problems and solutions as in a knowledge sharing case; governance is flat and the participation of contributors is open.

2.4 Service supplying collaboration

Participation criteria applied to service supplying collaboration, leads to two subtypes of collaboration. Namely, when participation is closed, we talk of team servicing; while when participation is open, it is crowdsourcing.

2.4.1 Team servicing

It is a form collaboration where, contributors have complementary skills and are chosen by an organization to carry out a project. Everyone knows everybody else, their skill sets and their contribution to work. Governance here is hierarchical, with business processes well defined; while participation remains closed. Deadlines

are set and achievements are equally recognized. This type of collaboration is the most common, due to service supplying e.g health management cases [65], software development, administrative managements, etc.

As an illustration case, consider an E-administration application for issuing civil status certificates, with the process described on figure 2.2, showing collaborations between roles Declarer, Secretary, csOfficer and Judiciary_Authority; roles played respectively by the actors Citizen, Agent, (Mayor, Diplomat) and Prosecutor as contributors in an issuance civil status certificate (birth, marriage, death) activity according to laws in Cameroon. this example is a cross-functional collaboration case, as Mayor and Diplomat can issue certificate deliverance, but in different contexts.

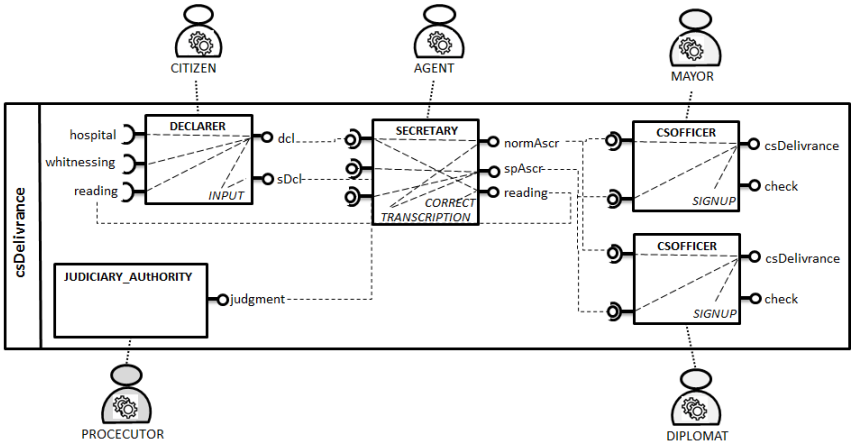


Figure 2.2: Administrative collaboration in civil status certificate issuance

2.4.2 Crowdsourcing

The crowdsourcing concept, introduced by Jeff Howe [74], is a collaborative business process model, in which tasks are carried out by a crowd. Simply, an organization wishing to carry out a task can requests an online community for a voluntary accomplishment of the task, according to what both parties have a mutual benefit. In crowdsourcing, organizational objectives are top-down, while creative activity is bottom-up. We therefore say that crowdsourcing is a mixed mode organized process [1, 75, 76, 3, 77], as it is a combination of top-down and bottom-up processes.

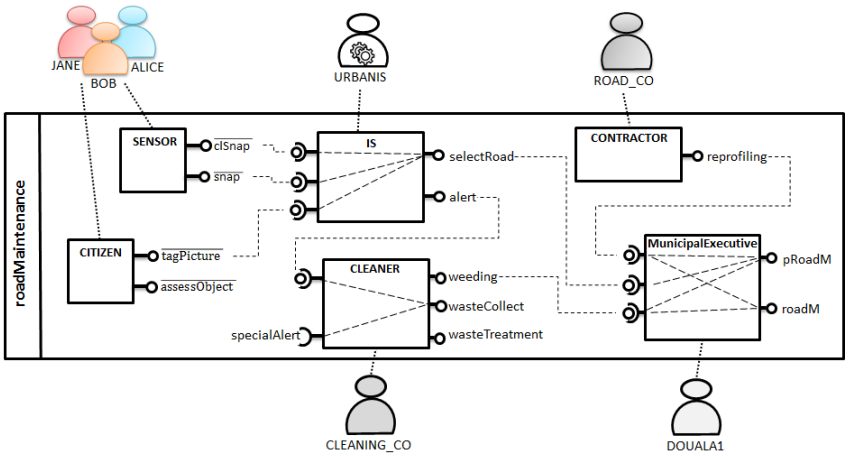


Figure 2.3: A crowdsourced road maintenance activity

As crowdsourcing use case, consider a city participatory management case [78, 79], with processes depict on figure 2.3, where citizens (Bob, Jane, Alice,etc.) via the urban information system (UrbanIS), provide information on the state of city roads and determine which ones to maintain as a priority. Cleaning is done (by Cleaning_CO) on the targeted roads, while municipal executive (MunicipalExecutive) contracts with a company (Road_CO) in order to carry out the work.

Crowdsourcing can also be defined as an activity (either costly in time or in resources) outsourcing process, towards a large number of anonymous actors termed the crowd. A crowdsourcing system is then an environment, on which a crowdsourcing process is deployed. It integrates the practices of business process modeling, and establishes the connection between the crowd, different individual actors and machines.

2.5 Formalizing business collaboration

2.5.1 contributor in a business collaboration

Definition 2.5.1. (contributor): A contributor of a business collaborative process, is an active entity gathering four things:

(1) an interface describing required services (or data) and provided services (or data),

(2) a storage for persistence within the contributor,

(3) business rules or functionalities targeted by the contributor, lastly

(4) communication resources and mechanisms by which a contributor exchanges with others.

Building collaboration, requires to identify capabilities that must be added to individual contributors so that they can work with others [80]. So when considering a contributor, a clear separation is made between its infrastructure (storage, communication mechanisms, sensing,etc.) and its functional goal in the system (interface, business rules). The contributor infrastructure is thus perceived as an actor part, while its functional goal is a role part. In fact, within a collaboration, each entity can be assigned a single role, or rather play several roles at a moment. In addition, due to potential unavailability of entities (maybe caused by failure, saturation, etc.) at a given time, a complete process reconfiguration may occurred, which strongly impacts the nature of potential services carried out. Contributors may not have the same skills, and several occurrences of a role could exist at the same time; for those reasons, collaboration is said to be a dynamic process.

2.5.2 Business collaboration tasking model

2.5.2.1 Preliminaries and properties

At this step we focus on service supplying collaboration types. Recall that, as the goal is to describe processes, we are just interested with grammar structure in GAG concept, mainly business service and business rules levels. A production is interpreted as an action that is performed in order to (partially) solve the task A in its left-hand side, and the symbols in its right-hand side represent the residual tasks that should in turn be solved in order to get a full completion of A. Thus a derivation $A \rightarrow^* u$ is interpreted as a process (a partially ordered set of actions) directed towards the resolution of task A, and this process is complete if u is the empty word.

The process is conditionally complete if u contains only external services (symbols in ${}^\bullet G$) since its completion is then conditioned by the complete execution of

these services by its environment. If ${}^{\bullet}G = \emptyset$, i.e. the grammar represent a standalone application that relies on no external services, and if the grammar is reduced then every partial resolution of a task can be extended in order to reach completion. This property is called soundness. One can define a relative notion of conditional soundness stating that any provided service can indeed be rendered using external services irrespective of the way the computation started.

The soundness property that can easily be checked by a fixed point computation is nonetheless undecidable for guarded attribute grammars due to fact that data attached to services can dynamically restrict the set of potentially applicable productions [59]. Note moreover that the soundness of its underlying grammar is neither a sufficient nor a necessary condition for the soundness of a guarded attribute grammar due to the non-monotony of this property arising from the combination of a universal and an existential quantifier in its definition:

$$\forall A \rightarrow^* u \quad \exists u \rightarrow^* \varepsilon$$

2.5.2.2 Intrinsic skills and services

Consider a production or business rule of a given grammar $G = (S, P)$ below

$$s_0 \rightarrow \boldsymbol{T_0 \cdots T_m} s_1 \cdots s_n \; with \; n, p \in \mathbb{N} \tag{2.1}$$

- $s_0 \in S$ is a defined service, i.e service carried out by processing the rule 2.1 above.
- $s_1 \cdots s_n$ are used services in that rule, and they must be completed before completing s_0.
- $T_0 \cdots T_m$ are intrinsic skills, implemented as primitives (see section 2.2.1.3).

Within a role and considering business rules, services defined but not used, are called provided services, those used but not defined are required services, lastly services defined and used are internal services.

2.5.2.3 Crowd tasks

Definition 2.5.2 (Crowd task). A crowd task $\overline{s}_i$ with $i \in \mathbb{N}$ is defined by a business rule like $\overline{s}_i \rightarrow \boldsymbol{T_0 \cdots T_m}$ i.e they are type of services requiring only contributors intrinsic skills, to be carried out and thus are autonomous services.

Consider a business rule $s_0 \rightarrow \boldsymbol{T_0 \cdots T_m}$ one may ask the difference between services s_0 and $\overline{s}_0$. In fact s_0 is a potentially outsourceable task, it can be defined and used in the same role (or used by another role); rather $\overline{s}_0$ is an outsourced task, so it is only defined in an autonomous role, termed crowd role, and only used by a requester role. By convention, a crowd role only define outsourced tasks, and we say a crowd role supply only crowd services.

2.5.2.4 Business skills

Definition 2.5.3 (Business skills). A business skill is a set of business rules, describing job decomposition in the form

$$s_0 \rightarrow \boldsymbol{T_0 \cdots T_m}\ s_1 \cdots s_n\ \overline{s}_0 \cdots \overline{s}_p \tag{2.2}$$

where:

- s_0 is a defined service,
- $s_1 \cdots s_n$ are used services
- $\boldsymbol{T_0 \cdots T_m}$ are contributor's intrinsic skills, finally
- $\overline{s}_0 \cdots \overline{s}_p$ are crowd tasks.

According to rule 2.2, to supply service s_0, services $s_1 \cdots s_n$ and $\overline{s}_0 \cdots \overline{s}_p$ must be achieved and skills $\boldsymbol{T_0 \cdots T_m}$ are needed by contributor playing that role.

2.5.3 Tasking model use cases

2.5.3.1 Issuing civil status certificates

Reconsider the e-administration application for issuing civil status certificates, described on figure 2.3,

- Any Declarer, can trigger the process of establishing a civil status certificate, by a declaration which can be either normal (dcl) or special ($sDcl$), following business skills expressed by rules (2.3) below.

$$\begin{aligned} br_0: \quad dcl \quad &\rightarrow \quad \textbf{\textit{INPUT}}\ withnessing \\ &\mid \quad \textbf{\textit{INPUT}}\ hospital \\ &\mid \quad \textbf{\textit{INPUT}}\ reading \end{aligned} \tag{2.3}$$

$$br_1: \quad sDcl \quad \rightarrow \quad \textbf{\textit{INPUT}}$$

- A Secretary according to business skills in rule (2.4), is responsible of declarations correction and transcription in related registers, whether these declarations are normal ($normAscr$) or special ($spAscr$).

$$\begin{aligned} br_2: \quad normAscr \quad &\rightarrow \quad \textbf{\textit{TRANSCRIPTION}}\ dcl \\ br_3: \quad spAscr \quad &\rightarrow \quad \textbf{\textit{TRANSCRIPTION}}\ sDcl\ judgment \\ br_4: \quad reading \quad &\rightarrow \quad \textbf{\textit{CORRECT}}\ dcl \end{aligned} \tag{2.4}$$

- Lastly according to rule (2.5), a CsOfficer is responsible of issuing civil status certificates and certain checks. Checks rules are not yet expressed and may be done later.

$$\begin{aligned} br_5: \quad csDelivrance \quad &\rightarrow \quad \textbf{\textit{SIGNUP}}\ normAscr \\ &\mid \quad \textbf{\textit{SIGNUP}}\ spAscr \\ br_6: \quad check \quad &\rightarrow \quad \varepsilon \end{aligned} \tag{2.5}$$

2.5.3.2 Crowdsourced road maintenance activity

If we reconsider the crowdsourcing activity described on figure 2.3,

- Any Sensor, can snap an object, in a predefined context (state of the object) and (geographic) location ($\overline{clSnap}$) on the one hand or quite simply according to the circumstances of the moment ($\overline{snap}$), following business skills expressed by business rules in (2.6).

$$\begin{aligned} \overline{clSnap} &\rightarrow \boldsymbol{CONTEXT\ LOCATION\ SNAP} \\ \overline{snap} &\rightarrow \boldsymbol{SNAP} \end{aligned} \tag{2.6}$$

- A Citizen according to business skills in rule (2.7), can tag a picture and assess object, in both cases by inputting some data.

$$\begin{aligned} \overline{tagPicture} &\rightarrow \boldsymbol{INPUT} \\ \overline{assessObject} &\rightarrow \boldsymbol{INPUT} \end{aligned} \tag{2.7}$$

- an IS i.e information system is responsible of data collection, selecting targeted road for maintenance, and sending some alerts (rules are not yet defined).

$$\begin{aligned} selectRoad &\rightarrow \overline{clSnap}\ \ \overline{snap}\ \ \overline{tagPicture} \\ alert &\rightarrow \varepsilon \end{aligned} \tag{2.8}$$

2.6 Conclusion

This chapter was devoted to business collaboration concepts, particularly on definition, characterization, taxonomy, as well as use cases illustrating some real world situations. Ultimately, a collaboration is implemented as a set of independent tenants, qualified to offer specific services, in a given context. Among the four types of collaboration presented, we focused for the family of collaborations with an hierarchical organization (team servicing and crowdsourcing) for which a business tasking model has been described, in guarded attribute grammars formalism. The next chapter will define role interface, i.e. an extension made to GAGs, with the aim of providing them with (de) composition mechanisms, conducive to flexible description of collaborations.

CHAPTER 3

AN INTERFACE OF ROLE THEORY FOR COLLABORATION

3.1 Introduction

This chapter focuses on service-oriented design, for collaboration modeled by GAG, with the purpose of providing a reasoning base. 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 works that have been carried out on behavioral interfaces of communicating processes. Three main ingredients have been put forward here, which will serve as our guideline. First, an interface is mainly used to formalize a contract-based reasoning for components. Second, an interface is viewed as an abstraction of a component, a so-called behavioral type. Third, 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 C, satisfies the global property (residual specification concept).

3.2 Modeling interface of role

Considering a very simple extension of interface concept, obtained by adding a binary relation on a set of services, indicating for each of provided services, the list of services that are potentially required to carry it out. This relation gives only potential dependencies because a service can be provided in various ways and relying on a variety of external services.

Recall that as the goal is to describe processes, we are just interested with grammar structure in GAG concept. Business skills are expressed as grammar production rules (or business rules), describing job decomposition, in the form

$$s_0 \rightarrow \boldsymbol{T_0 \cdots T_m}\ s_1 \cdots s_n\ \overline{s}_0 \cdots \overline{s}_p \tag{3.1}$$

where s_0 is a defined service, $s_1 \cdots s_n$ are used services and $\boldsymbol{T_0 \cdots T_m}$ are contributor's intrinsic skills, finally $\overline{s}_0 \cdots \overline{s}_p$ are crowd tasks (see section 2.5.2).

3.2.1 Potential dependencies

In a business collaboration, a service is potentially provided in different ways, described by business rules. However, the manner of rendering this service is dictated both, on the one hand, by choices made by the contributor of this service and, on the other hand, by the availability of required services.

$$\begin{aligned} declaration_0 :\quad dcl \quad &\rightarrow \quad \boldsymbol{INPUT}\ withnessing \\ &\mid \quad \boldsymbol{INPUT}\ hospital \\ &\mid \quad \boldsymbol{INPUT}\ reading \end{aligned} \tag{3.2}$$

$$declaration_1 :\quad sDcl \quad \rightarrow \quad \boldsymbol{INPUT}$$

Reconsidering rule $declaration_0$ above for example, we can observe that service *dcl* could be rendered in three different ways. It will be said that *dcl* potentially depends on services *withnessing*, *hospital* and *reading* respectively. At a given moment, the effective way of rendering this service *dcl*, will depend on the choice made by contributor Citizen, playing role Declarer.

Definition 3.2.1. Potential dependencies are over-approximations of relations between provided services and required services.

3.2.2 Grammar interface

The goal is to define an abstraction of grammar, called an interface for grammar, specifying which services are provided, which external services are required to perform them, and potential dependencies; in particular, the interface ignores internal tasks.

$$
\begin{array}{ll}
br_1: & A \to \varepsilon \\
br_2: & A \to B\ C \\
br_3: & B \to \varepsilon \\
br_4: & B \to D
\end{array}
$$

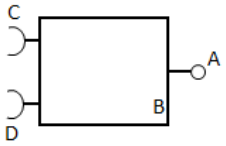


Figure 3.1: Grammar of a role providing service A

Figure 3.1 above shows a grammar of a role, providing service A by using external services C and D. B is an internal service, related to the role, and can be renamed. In order to process service A that is a pending task in his workspace, rule br_1 can be chosen and applied (which corresponds to a certain action or activity) and this decision ends execution of service A (since the right part is empty). Alternatively, rule br_2 can also be chosen. In that case, two new (residual) services, namely B and C are created and A will end as soon as B and C are completed.

It is considered that provided service s_0, potentially depend on required service s_1, if there is a derivation $s_0 \to^* u$ where the word u contains an occurrence of s_1.

Definition 3.2.2. An interface for grammar $\mathcal{G} = (S, P)$ is $I(\mathcal{G}) = ({}^\bullet\mathcal{G}, R(\mathcal{G}), \mathcal{G}^\bullet)$ where $R(\mathcal{G}) = \{(s_1, s_0) \mid \exists u \in S^* \quad s_0 \to^* u \ \wedge \ \sharp_{s_1}(u) \neq 0\}$ and $\sharp_{s_1}(u)$ denotes occurrences number of symbol s_1 in the word u.

3.2.3 Interface of role

An interface of role [8, 9] is some abstraction of the guarded attribute grammar $\mathcal{G}$ associated to a 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 termed potential dependencies. Interface disregards internal services.

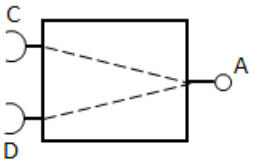


Figure 3.2: An interface of role associated with the role on figure 3.1

The interface of role given on figure 3.2, is the relation $R = \{(C, A), (D, A)\}$, an over-approximation of dependencies, because according to business rules on figure 3.1, it may happen that A uses none of the services C and D (in using the $A \rightarrow^* \varepsilon$ derivation) or only C (using the $A \rightarrow^* C$ derivation). But calling service A does not require knowing how the service will be performed, and must therefore assume availability of all potentially required services used. Nevertheless, We will assume that the grammar is reduced.

Definition 3.2.3. Let Ω a fixed set of services. An interface $({}^\bullet R, R, R^\bullet)$ consist of a finite binary relation $R \subseteq \Omega \times \Omega$ of disjointed sub-sets ${}^\bullet R$ and $R^\bullet$ from Ω, such that ${}^\bullet R = R^{-1}(\Omega) = \{s_0 \in \Omega \mid \exists s_1 \in \Omega,\ (s_0, s_1) \in R\}$ and $R^\bullet \supseteq R(\Omega) = \{s_1 \in \Omega \mid \exists\, s_0 \in \Omega,\ (s_0, s_1) \in R\}$.

Set $R^\bullet$ represents services provided (or defined) by the interface and ${}^\bullet R$ is the set of required (or used) services. 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.

Note that, since ${}^\bullet R$ is the domain of the R relation, the set of required services may remain implicit. This is not the case for all the services provided because it can strictly encompass the co-domain of the relation. Nevertheless, we will use the same symbol to designate an interface and its underlying relation.

The Declarer interface of rules 3.2 above, is then defined by figure 3.3 and equation 3.3 below.

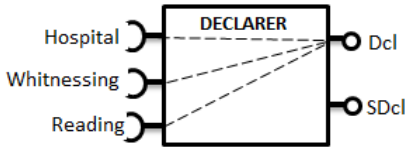


Figure 3.3: Interface of a Declarer role

$$
\begin{aligned}
&{}^{\bullet}DECLARER = \{hospital, whitnessing, reading\}\\
&DECLARER^{\bullet} = \{dcl, sDcl\}\\
&DECALER = \{(hospital, dcl), (whitnessing, dcl), (reading, dcl), (\_, sDcl)\}
\end{aligned}
\tag{3.3}
$$

3.3 Properties, conventions, basic operations

3.3.1 Property and conventions

Autonomy or closure: An interface is closed (or autonomous) if relation R (and therefore also ${}^{\bullet}R$) is empty. Thus, a closed interface is given by all services it (autonomously) provides.

In order to facilitate ratings and calculations, we will use the following conventions:

Emptiness: The empty interface that does not provide any service is noted $\emptyset = (\emptyset, \emptyset, \emptyset)$.

Transitivity: We will first identify a relation $R \subseteq \Omega \times \Omega$ such that $R(\Omega) \cap R^{-1}(\Omega) = \emptyset$ with interface $({}^{\bullet}R, R, R^{\bullet})$ such that ${}^{\bullet}R = R^{-1}(\Omega)$ and $R^{\bullet} = R(\Omega)$. We also identify a set of $X \subseteq \Omega$ with the restriction of the identity relation to the whole X, i.e., the diagonal $\{(A, A) \in \Omega^2 \mid A \in X\}$. In doing so, it is possible, for example, to express the condition $B \in Y \wedge (\exists C \in X\ (A, C) \in R \wedge (C, B) \in Y)$ for $R, S \subseteq \Omega \times \Omega$ et $X, Y \subseteq \Omega$ since $(A, B) \in R; X; S; Y$ where operator $(;)$ denotes sequential composition defined below in section 3.3.2 Note also that with this convention we have $X \cap Y = X; Y$ for X and Y sub-sets of Ω.

3.3.2 Sequential composition

Let R_1 and R_2 two interfaces, a sequential composition of those interfaces, denoted $R_1; R_2$ is defined when ${}^\bullet R_1 \cap R_2^\bullet = \emptyset$ and given by

$$R_1; R_2 = \left\{(s_0, s_2) \in \Omega^2 \,|\, \exists s_1 \in \Omega \; (s_0, s_1) \in R_1 \wedge (s_1, s_2) \in R_2\right\}$$

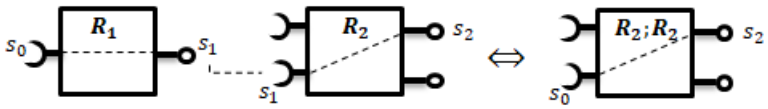


Figure 3.4: Sequential composition of two interfaces

the composition of their underlying relations with ${}^\bullet(R_1; R_2) = R_1^{-1}({}^\bullet R_2) \subseteq^\bullet R_1$ and $(R_1; R_2)^\bullet = R_2(R_1^\bullet) \subseteq R_2^\bullet$. Sequential composition, simply solves transitivity in potential dependencies between services.

3.3.3 Restriction and Co-restriction

Let $O \subseteq \Omega$ a subset of services. A restriction of interface R to O denoted by $R \restriction O$ is given as

$$R \restriction O = \{(s_1, s_0) \in R \,|\, s_0 \in O\}$$

With $(R \restriction O)^\bullet = O \cap R^\bullet$ and ${}^\bullet(R \restriction O) = R^{-1}(O \cap R^\bullet)$ respectively. In fact, considering any system having a role r interfaced by R, and a subset of services O, the restriction operation reconfigures role r so that, only its provided services elements of O are enabled; other provided services are inhibited. In practice, this operator is useful when you are interested just by some skills of a multi-skilled role.

3.3.4 Union of interfaces, quasi-interface and acyclicity

The union of interfaces is an interface if none of the services defined by one interface is used by another. In the general case, $R = (\cup_i^\bullet R_i, \cup_i R_i, \cup_i R_i^\bullet)$ meets the conditions of the definition 3.2.3 of interface, but ${}^\bullet R \cap R^\bullet = \emptyset$. If the R relation is acyclic, it is said to be a quasi-interface since it induces an interface given by the following definition.

Definition 3.3.1. If $R = ({}^\bullet R, R, R^\bullet)$ is a quasi-interface, i. e. R is an acyclic relation, ${}^\bullet R = R^{-1}(\Omega)$ and $R^\bullet \supseteq R(\Omega)$, let's put $\langle R \rangle = R^* \cap (I \times O)$, where $I = {}^\bullet R \setminus R^\bullet$ and $O = R^\bullet$. $\langle R \rangle$ is an interface with ${}^\bullet\langle R \rangle = I$ and $\langle R \rangle^\bullet = O$.

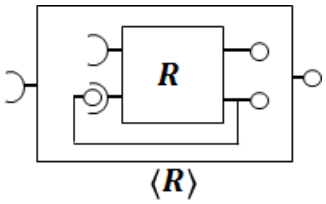


Figure 3.5: Interface induced by a quasi-interface

Example 1. For instance if $R_1 = (\emptyset, \emptyset, \{A\})$ is an 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 A from B, 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. More specifically, what example 1 above, shows is the (simplest) illustration of two grammars that are reduced but whose union is not. This is due to the cycle created when we put them together. It is immediate that the union of two reduced grammars whose union of interfaces is acyclic is also reduced. Our objective is to be able to present an autonomous grammar as the gathering of subgrammars. The global grammar will thus be reduced (and therefore sound) if each of the subgrammars is reduced and if the operation of union (whose associativity we will seen below) preserves this property. Hence the importance of this acyclicity hypothesis, even though it is somewhat pessimistic. In a way, this assumption imposes a constraint on how to break down a system into sub-modules, i.e. how to structure our specification. In practice, as we have experienced in our previous studies [81], these

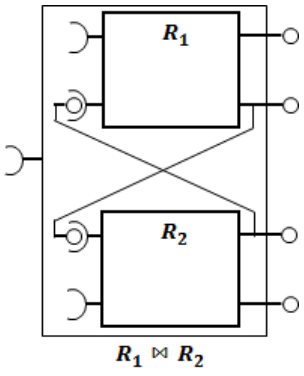
constraints are reasonable. Nevertheless, if we give ourselves finer abstractions of grammars, which will be done further by introducing non-deterministic interfaces, we can end up with less constrained forms of composition.

3.4 Interface of role composition

3.4.1 Concept and principles

An interface is mainly used to formalize a contract-based reasoning for components. The idea is that a component of a reactive system [82] 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 [83]. 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 [84] to I/O automata and games [85]. These interface theories have also been extended to take some qualitative aspects into account (time and/or probability).

3.4.2 Composition operation



Note that $(R_1 \bowtie R_2)^\bullet = R_1^\bullet \cup R_2^\bullet$. In addition, since $R_i^\bullet \cap R_i^\bullet = \emptyset$ for $i = 1, 2$ we get

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

Figure 3.6: Composition of two interfaces.

Definition 3.4.1. Two interfaces R_1 and R_2 are said to be composable (figure ??) if their union $R_1 \cup R_2$ is an acyclic relation and $R_1^\bullet \cap R_2^\bullet = \emptyset$. Then we'll denote

$R_1 \bowtie R_2 = \langle R_1 \cup R_2 \rangle$ their composition.

It follows directly from the definition that the composition of interfaces is commutative and has the empty interface as a neutral element. Note that we can have ${}^\bullet R_1 \cap {}^\bullet R_2 \neq \emptyset$, so that both interfaces may require common external services.

3.4.3 Associativity of the composition

The following example shows that composition is not associative, if we do not require that composable interfaces have disjoint outputs. Of course, the hypothesis of having an associative interface composition operation is a matter of choice. One could alternatively have chosen to drop the assumption that composable interfaces have disjoint output sets and to deal with a non-associative composition operation.

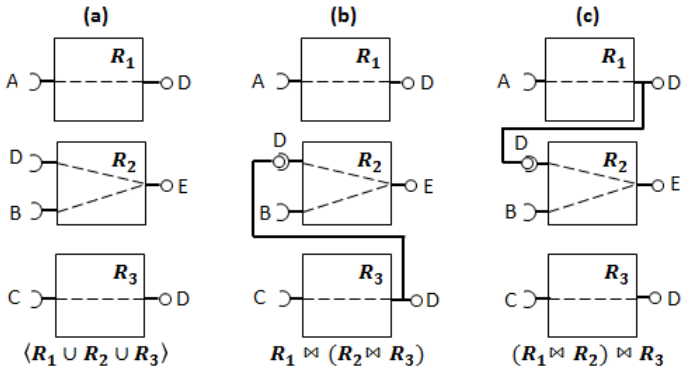


Figure 3.7: A counter-example showing that, associativity of composition does not hold, if interfaces shared some provided services.

Example 2. Let R_1, R_2, and R_3 three interfaces given by figure 3.7. 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 $R = \langle R_1 \cup R_2 \cup R_3 \rangle$, so $R = \{(A, D), (C, D), (A, E), (B, E), (C, E)\}$. It should be noted that service D could be produced either by R_1 or R_3 so that we have 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 required service D is no longer an entry in $R_2 \bowtie R_3$. Symmetrically $R_l = (R_1 \bowtie R_2) \bowtie R_3 = \{(A, D), (C, D), (A, E), (C, E)\}$.

Remark 2. $\langle R\rangle = \{(A,B) \in R^* \mid \neg\,(\exists C \in \Omega.\ (C,A) \in R)\}$. Thus $(A,B) \in \langle R\rangle$ is associated to path in the graph R leading to $B \in R^\bullet$ and cannot be extended to 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 the elements of this path, except the first one, A, belong to $R^\bullet$.

Proposition 3.4.1. The composition of the interfaces is associative. More precisely, if $R_1 \cdots R_n$ are interfaces that can be pairwise composed, then $\bowtie_{i=1}^{n} R_i = \langle R_1 \cup \cdots \cup R_n\rangle$.

Proof. Using the commutativity of composition, the proposal follows by induction on n as soon as it was verified that $(R_1 \bowtie R_2) \bowtie R_3 = \langle R_1 \cup R_2 \cup R_3\rangle$ for interfaces that can be composed in pairs R_1, R_2 and R_3. So 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 than $\langle\langle R\rangle \cup R'\rangle = \langle R \cup R'\rangle$ 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 $(R \cup R')^*$ is acyclic, and ${}^\bullet R \cap (R')^\bullet = \emptyset$. First of all, note that $\langle R\rangle^\bullet = R^\bullet$ and $(R \cup R')^\bullet = R^\bullet \cup (R')^\bullet$ And so $\langle\langle \cup R'\rangle^\bullet = \langle R \cup R'\rangle^\bullet$. By condition $R^\bullet \cap (R')^\bullet = \emptyset$ we deduce $R \cap R' = \emptyset$. More specifically, a transition from $(A,B) \in R \cap R'$ belongs (exclusively) either to R or R' depending respectively of $B \in R^\bullet$ or $B \in (R')^\bullet$. According to the remark 1, i. e. $\pi = A_0 \to A_1 \to \cdots \to A_n$ a path in $R \cup R'$ (i.e. $\forall 1 \leq i \leq n\ (A_{i-1}, A_i) \in R \cup R'$ and $A_0 \in {}^\bullet(R \cup R') \setminus (R \cup R')^\bullet$ considering that $(A_0, A_n) \in\in \langle R \cup R \cup R'\rangle$. Let's say $\pi' = A_i \to \cdots \to A_j$ be a maximum subpath of π made of R transitions only (i.e., $\forall i \leq k \leq j \leq j\ A_k \in R^\bullet$). So 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 so π' is a path that shows that $(A_i, A_j) \in \langle R\rangle$ which means that π is a path that shows 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 so $\langle\langle R\rangle \cup R'\rangle = \langle R \cup R'\rangle$ since the reverse inclusion is immediate. □

The following two cases of composition are to be distinguished: the cascaded product and the direct product.

3.4.4 Cascade product

Let R_1 and R_2 two composable role interfaces, as stated in definition 3.4.1. If $R_1^\bullet \cap {}^\bullet R_2 = \emptyset$ we denote $R_1 \rtimes R_2$ their cascaded composition or semi-direct product

(or $R_2 \ltimes R_1$ since this operation, as a special case of $\bowtie$ remains commutative). So ${}^\bullet(R_1 \rtimes R_2) = ({}^\bullet R_1 \setminus R_2^\bullet) \cup {}^\bullet R_2$, and $(R_1 \rtimes R_2)^\bullet = R_1^\bullet \cup R_2^\bullet$.

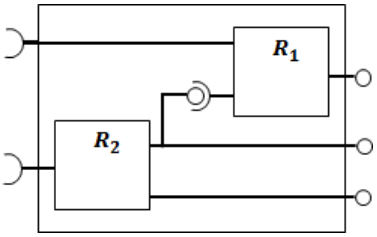


Figure 3.8: Cascaded product of two roles R_1 and R_2

The cascaded product of interfaces R_1 and R_2 is a form of role interfaces composition, in which R_2 requires no service from R_1. The composite interface obtained has as provided services, the union of the various provided services of R_1 and R_2. The composite interface required services is then defined as the union of required services of R_1 and R_2, where those provided by R_2 and required by R_1 are removed.

Remark 2. The underlying relation of the cascaded product is given by

$$R_1 \rtimes R_2 = (I_1 \times R_2)\,;(R_1 \times O_2)$$

where $I_1 = {}^\bullet R_1 \setminus R_2^\bullet$ et $O_2 = R_2^\bullet \setminus {}^\bullet R_1$.

3.4.5 Direct product

Let R_1 and R_2 two composable role interfaces, as stated in definition 3.4.1. If $R_1^\bullet \cap {}^\bullet R_2 = \emptyset$ and $R_2^\bullet \cap {}^\bullet R_1 = \emptyset$ are verified, we say that the composition is the product (direct) of R_1 and R_2, denoted by $R_1 \times R_2$. Note that $R_1 \times R_2 = R_1 \cup R_1$ and so ${}^\bullet(R_1 \times R_2) = {}^\bullet R_1 \cup {}^\bullet R_2$ and $(R_1 \times R_2)^\bullet = R_1^\bullet \cup R_2^\bullet$.

Direct product of interfaces R_1 and R_2, is a special case of composition, where both associated roles requires no services from each other. Composition operation, is then assumed as a pooling skills operation, since required services of the composition, is the union of R_1 and R_2 required services, and the same for the composition provided services.

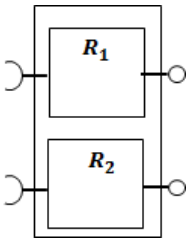


Figure 3.9: Direct product of two roles R_1 and R_2

3.4.6 Componentization test

Componentization which is the process of breaking into separate components, is an approach to software development that involves breaking software down into identifiable pieces, independently implements and deploy, these components are then stitched together with network connections and workflows. We gain flexibility in design and ease of component reuse. Precisely, in a context of role-based approach, having two interfaces R and R_1, it may be useful to know if their skills encompass, or may be if R_1 can be used in a context where R is expected; then R_1 is said to be a component of R.

Definition 3.4.2. R_1 is a component of R, noted $R_1 \sqsubseteq R$, if there is an interface R_2 such as $R = R_1 \bowtie R_2$. R_1 is a strict component of R, noted $R_1 \sqsubset R$, if there is an interface R_2 such as $R = R_1 \times R_2$.

3.5 Implementation order

An interface is viewed as an abstraction of a component, a so-called behavioral type. Thus we must be able to state when a component satisfies an interface, viewed as an abstract specification of its behavior. 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.

An interface environment is a component that provides all the services required by the interface and uses only the services provided by the interface for this purpose.

Definition 3.5.1. An E interface is an eligible environment for an R interface if both interfaces are composable and the composition is a closed interface, namely ${}^{\bullet}(R \bowtie E) = \emptyset$. We let **Env** (R) refers to all eligible environments of the R interface.

Definition 3.5.2. An interface R_2 is a implementation of the interface R_1, noted $R_1 \leq R_2$, when $R_2^\bullet = R_1^\bullet$ and $R_2 \subseteq R_1$.

Thus, R_2 is an implementation of R_1 if it provides the same services as R_1 using only those services already available to R_1 and with fewer dependencies [1]. The following proposition shows that R_2 is an implementation of the R_1 interface if and only if it can be substituted to R_1 in any eligible environment for R_1.

Proposition 3.5.1. $R_1 \leq R_2$ if and only if $\mathbf{Env}(R_1) \subseteq \mathbf{Env}(R_2)$.

Proof. We first show that the condition is necessary. For this reason, let's 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 eligible environment E for R_1 is a qualifying environment for R_2. Since E is composable with R_1 we have $R_1^\bullet \cap E^\bullet = \emptyset$ 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 $R_2 \subseteq R_1$. Thus E is composable with R_2. Moreover, for the same reasons, ${}^\bullet(E \bowtie R_2) = ({}^\bullet E \setminus R_2^\bullet) \cup ({}^\bullet R_2 \setminus E^\bullet) \subseteq ({}^\bullet E \setminus R_1^\bullet) \cup ({}^\bullet R_1 \setminus E^\bullet) = {}^\bullet(E \bowtie R_2) = \emptyset$. Hence $E \in \mathbf{Env}(R_2)$. We show that the condition is sufficient by contradiction. Since $R_1 \leq R_2$ implies $R_2^\bullet = R_1^\bullet$ you have to build $\mathcal{E} \in \mathbf{Env}(R_1) \setminus \mathbf{Env}(R_2)$ assuming that $R_1 \not\supseteq R_2$. Let's say $(A, B) \in R_2 \setminus R_1$ then the interface we're looking for is E such as ${}^\bullet E = \{B\}$, $E^\bullet = {}^\bullet R_2$, and $E = \{(B, A)\}$. Indeed, E can be composed with R_1 but not with R_2 because of the $B \to A \to B$ cycle in $(R_1 \cup \{(B, A)\}^*$. In addition, the composition of E with R_1 gives a closed interface. □

3.6 Residual specification

Third, a notion of residual specification has also proved to be useful. The problem was first stated in [86] 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 $C \bowtie X \approx G$ where $\bowtie$ stands for the composition of specifications and $\approx$ is some

[1] In practice, an interface used as an implementation can define additional services: R_2 is a implementation low of the R_1 interface, in notation $R_1 \leq_w R_2$, if $R_2^\bullet \supseteq R_1^\bullet$ and $R_1 \leq R_2 \upharpoonright (R_1^\bullet)$. However, the additional services provided by R_2 must be masked so that they cannot conflict with the services in any environment compatible withR_1.

equivalence relation. If $\approx$ is the equivalence induced by the refinement relation the above problem can better be formulated as a Galois connection [87] $G/C \leq X \iff G \leq C \bowtie X$ stating that the residual specification G/C 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 $C \bowtie -$ 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 .

Proposition 3.6.1. If $R_1 \sqsubseteq R$ then $R = R_1 \rtimes (R \swarrow R_1)$ or $R \swarrow R_1$, called the strict residue of R per R_1, is given as the restriction from R to $R^\bullet \setminus R_1^\bullet$. If $R = R_1 \rtimes R_2$ then $R \upharpoonright R_2^\bullet = R \swarrow R_1 = R_2$ and $R = (R \swarrow R) \times (R \swarrow R_1)$.

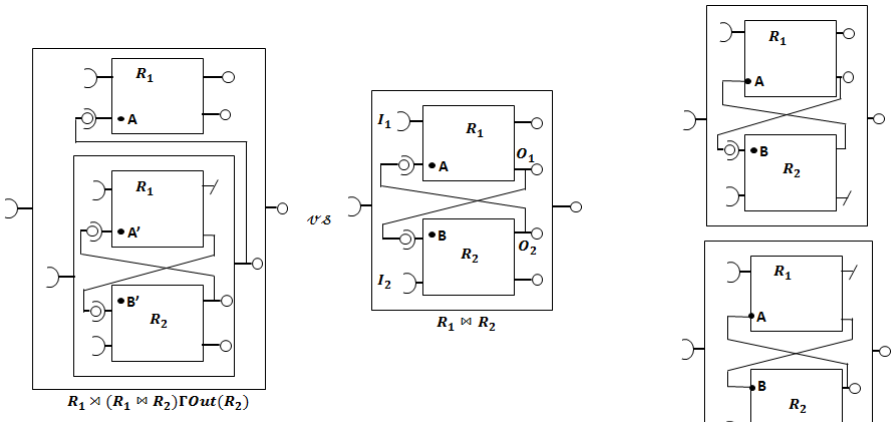
Proof. It must be shown that if R_1 and R_2 are two relations that can be composed with $R = R_1 \bowtie R_2$ then $R = R_1 \rtimes R \swarrow R_1$ and $R = (R \swarrow R_1) \times (R \swarrow R_1)$ or $R \swarrow R_i = R \upharpoonright R_j^\bullet$ for $\{i, j\} = \{1, 2\}$. By the remark 1 $R_1 \bowtie R_2$ is (the only) [2] solution of the equation system.

$$
\begin{aligned}
R_1 \bowtie R_2 = \;& (A \cup I_1); R_1 \cup (B \cup I_2); R_2 \\
\text{où} \qquad I_1 = \;& {}^\bullet R_1 \setminus R_2^\bullet \\
I_2 = \;& {}^\bullet R_2 \setminus R_1^\bullet \\
O_1 = \;& R_1^\bullet \cap {}^\bullet R_2 \\
O_2 = \;& R_2^\bullet \cap {}^\bullet R_1 \\
A = \;& (B \cup I_2); R_2; O_2 \\
B = \;& A \cup I_1); R_1; O_1
\end{aligned}
$$

So it is immediate (see Figure 3.10*a*) that $R_1 \rtimes (R_1 \bowtie R_2) \upharpoonright R_2^\bullet$ is the solution of the same 'equations system and thus the two relations coincide. The same system of equations is associated to $(R \swarrow R_2) \times (R \swarrow R_1)$ as shown in figure 3.10*b*. □

It remains to be shown that if $R = R_1 \rtimes R_2$ then $R \swarrow R_1 = R \upharpoonright R_2^\bullet$ coincides with R_2, and indeed $R \upharpoonright R_2^\bullet = (({}^\bullet R_1 \setminus R_2^\bullet) \cup R_2)\,; (R_1 \cup R_2^\bullet) \upharpoonright R_2^\bullet = (({}^\bullet R_1 \setminus R_2^\bullet) \cup R_2) \upharpoonright R_2^\bullet = R_2$ by the remark 2 and because $R_1^\bullet \cap R_2^\bullet = \emptyset$.

[2] The uniqueness comes from the fact that we only consider finite paths due to acyclicity.



(a) $R = R_1 \rtimes (R \swarrow R_1)$ when $R = R_1 \bowtie R_2$

(b) $(R \swarrow R_2) \times (R \swarrow R_1)$

Figure 3.10: Residual composition

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

- R_1 is a strict component of R: $\exists R_2 \cdot R = R_1 \times R_2$,
- R_1 is a left component in the cascaded decomposition of R: $\exists R' \cdot R = R' \rtimes R_2$,
- R_1 is a restriction of R: $R_1 = \upharpoonright (R_1^\bullet)$, and
- R_1 is a strict residue of R: $\exists R_2 \cdot R_1 = R \swarrow R_2$.

Lemma 3.6.2. $R_1 \leq R_2$ implies that $R \bowtie R_1 \leq R \bowtie R_2$ each time R_1 and R_2 are both components of R.

Proof. By the remark 1 $(A, B) \in R \bowtie R_i$ if and only if there is 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 $B = A_n \in R^\bullet \cup R_i^\bullet$. The monotony of $R \bowtie -$ then results from the fact that $R_1^\bullet = R_2^\bullet$. □

Lemma 3.6.3. $R_1 \leq R_2$ implies that $R_1 \swarrow R \leq R_2 \swarrow R$ each time R is a respective component of R_1 and R_2.

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

Proposition 3.6.4. 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 the proposal 3.6.1 and the lemma 3.6.2 we have $R \swarrow R_1 \leq R' \implies R = R_1 \rtimes (R \swarrow R_1) \leq R \rtimes R'$. The opposite direction follows by the lemma 3.6.3 and the proposal 3.6.1: $R \leq R_1 \rtimes R' \implies R \swarrow R_1 \leq (R \rtimes R') \swarrow R_1 = R'$. □

As a corollary 3.6 the above proposal implies that an implementation of a strict residue $R \swarrow R_1$ is a strict component of R and therefore cannot capture all the components of an implementation of R, i.e. all interfaces R' such as $R \leq R_1 \bowtie R'$. To do this, we must add to the residue all dependencies between the respective outputs of the component and the residue that do not contradict the dependencies in R:

Definition 3.6.1. if $R_1 \sqsubseteq R$ the residual R/R_1 of R by R_1 is given by $(R/R_1)^\bullet = R^\bullet \setminus R_1^\bullet$ et $R/R_1 = R \swarrow R_1 \cup R \nearrow R_1$ where

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

Lemma 3.6.5. if R_1 is a component ofe 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 $R/R_1 \supseteq R \swarrow R_1$ we have $R/R_1 \leq R \swarrow R_1$ and by the lemma 3.6.2 $R_1 \bowtie (R/R_1) \leq R_1 \bowtie (R \swarrow R_1) = R_1 \rtimes (R \swarrow R_1) = R$. We still have to prove that $R_1 \bowtie (R/R_1) \subseteq R$. Let $(A, B) \in R_1 \bowtie (R/R_1)$ then by the remark 1 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 $1 \leq i \leq n$. On a ${}^\bullet(R_1 \bowtie (R/R_1)) = {}^\bullet R_1 \setminus (R^\bullet \setminus R_1^\bullet) \cup {}^\bullet(R/R_1) \setminus R_1^\bullet$. 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): □

1. $(A_{i-1}, A_i) \in R_1$ if $A_i \in R_1^\bullet$,
   (a) $(A_{i-1}, A_i) \in R \swarrow R_1$ if $A_i \in R^\bullet \setminus R_1^\bullet$ and $A_{i-1} \in {}^\bullet R \setminus R_1^\bullet$, or
   (b) $(A_{i-1}, A_i) \in R \nearrow R_1$ if $A_i \in R^\bullet \setminus R_1^\bullet$ and $A_{i-1} \in R_1^\bullet$,

Proof. Note that if the sequence does not contain any transition of the latter category, it then witnesses the fact that $(A, B) \in R$ because $R_1 \bowtie (R \swarrow R_1) = R_1 \rtimes (R \swarrow R_1) = R$. We're going to phase out all transitions of type (3). To do this, let us consider the leftmost transition that belongs to this type if it exists. Thus, i is the smallest index such as $(A_{i-1}, A_i) \in R \nearrow R_1$. Since $R_1^\bullet$ is a subset of $R^\bullet$ and thus disjoint from ${}^\bullet R$ we deduce that $A_{i-1} \neq A$ and $i - 1 \geq 1$. Now the sequence $\sigma : A = A_0 \to \cdots \to A_{i-1}$, which contains only type transitions (1) or (2), shows that $A \in R^{-1}(\{A_{i-1}\})$. Since $(A_{i-1}, A_i) \in R \nearrow R_1$ we deduct $A \in R^{-1}(\{A_i\})$. And so by replacing the sequence σ by the transition (A, A_i) we get a sequence with a transition less $R \nearrow R_1$ and thus we end up with a sequence without transition $R \nearrow R_1$ showing that $(A, B) \in R$. □

Lemma 3.6.6. 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 $(R/R_1)^\bullet = (R_1^\bullet \cup R_2^\bullet) \setminus R_1^\bullet = R_2^\bullet$. $(A, B) \in R_2 \setminus (R \swarrow R) = R_2 \setminus (R \restriction R_2)$ 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 3.6.7. $R_1 \leq R_2$ implies $R_1/R \leq R_2/R$ whenever R is a component of both R_1 and R_2.

Proof. Recall that $R_i \nearrow R = \{(A, B) \in R^\bullet \times (R_i^\bullet \setminus R^\bullet)\, R^{-1} \mid (\{A\}) \subseteq R^{-1}(\{B\})\}$ et $R_i/R = R_i \swarrow R \cup R_i \nearrow R$. $R_1 \leq R_2$ means that $R_1^\bullet = R_2^\bullet$ et $R_2 \subseteq R_1$ 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 $R \nearrow R \subseteq R_1 \nearrow R$. The result then follows lemma 3.6.3 and $(R_1/R)^\bullet = R_1^\bullet \setminus R^\bullet = R_2^\bullet \setminus R^\bullet = (R_2/R)^\bullet$. □

Proposition 3.6.8. If R_1 is a component R and R' is an interface, then

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

Proof. By lemma 3.6.5 and lemma 3.6.2 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 3.6.7 and lemma 3.6.6: $R \leq R_1 \bowtie R' \implies R/R_1 \leq (R \bowtie R')/R_1 \leq R'$. □

Hence residual R/R_1 characterize the interfaces of the components that, when composed with R_1, provide an implementation of R.

3.7 Non-deterministic Interfaces

The notion of interface presented so far is still a somewhat rough abstraction of the roles described by a GAG specification. In particular, we would like to be able to take into account the non-determinism that results from the choices offered to the user on how to solve a task. For that purpose we replace 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.

Definition 3.7.1. A non-deterministic quasi interface on a set Ω of services is a map $R : \Omega \to \wp(\wp(\Omega))$. We let $R^\bullet = \{A \in \Omega \mid R(A) \neq \emptyset\}$ and ${}^\bullet R = \cup\{R(A) \mid A \in \Omega\}$. It is a non-deterministic interface when ${}^\bullet R \cap R^\bullet = \emptyset$.

Definition 3.7.2. The non-deterministic interface of a grammar $G = (S, P)$ is the function $R = NI(G) : \Omega \to \wp(\wp(\Omega))$ where $\Omega = {}^\bullet G \cup G^\bullet$ (i.e. grammatical symbols that correspond to internal tasks are abstracted) and $R(A) = \{\pi(u) \subseteq {}^\bullet G \mid A \to^* u\}$ where $\pi(u) = \{A \in S \mid \sharp_A(u) \neq 0\}$ is the set of grammatical symbols that occur in word u. Thus a set of symbols is in $R(A)$ if and only if it is the set of symbols of a word in $({}^\bullet G)^*$ that derives from A. Note that if the grammar is reduced one has ${}^\bullet R = {}^\bullet G$ and $R^\bullet = G^\bullet$.

This new representation of interfaces is richer than its deterministic counterpart. It is more precise and might indeed been considered too much precise in some situations since it requires to handle more information. Nonetheless non-deterministic interfaces are in many respect more easy to handle than deterministic interfaces and they lead to simplified definitions and easier proofs as shown next. In particular, and by contrast with deterministic interfaces, both sets ${}^\bullet R$ and $R^\bullet$ can be left implicit. This is due to the fact that services A provided by a non-deterministic interface that requires no external services are now explicitly given as those such that $R(A) = \{\emptyset\}$ (which should not be confused with $R(A) = \emptyset$ which corresponds

to $A \notin R^\bullet$). Thus the following operations can straightforwardly be defined for non-deterministic quasi-interfaces.

- Sequential composition: $R_1; R_2(A) = \{\cup_i X_i \mid \{A_1, \ldots, A_n\} \in R_2(A) \text{ and } X_i \in A_i\}$.
- Restriction: $R{\restriction}O(A) = R(A)$ if $A \in O$ and $R{\restriction}O(A) = \emptyset$ otherwise.
- Union: $(\cup_i R_i)(A) = \cup_i R_i(A)$.
- Transitive closure: $R^*(A) = \cup_{i=1}^{\infty} R^i(A)$ where $R^{n+1} = R^n; R$. Note that $R^* = R$ if R is an interface.
- Composition: $R_1 \bowtie R_2 = \langle R_1 \cup R_2 \rangle$ where $\langle R \rangle(A) = R^*(A) \cap \wp(\wp(\Omega \setminus R^\bullet))$ is the non-deterministic interface induced by the non-deterministic quasi-interface R. Note that $(R_1 \bowtie R_2)^\bullet \subseteq R_1^\bullet \cup R_2^\bullet$ and ${}^\bullet(R_1 \bowtie R_2) \subseteq ({}^\bullet R_1 \setminus R_2^\bullet) \cup ({}^\bullet R_2 \setminus R_1^\bullet)$. A quasi-interface R is said to be reduced when ${}^\bullet\langle R \rangle = {}^\bullet R$ and ${}^\bullet\langle R \rangle = {}^\bullet R$. We say that two interfaces R_1 and R_2 are safely composable [3] when their union (the quasi-interface $R_1 \cup R_2$) is reduced. Two interfaces R_1 and R_2 are safely composable if and only if $(R_1 \bowtie R_2)^\bullet = R_1^\bullet \cup R_2^\bullet$ and ${}^\bullet(R_1 \bowtie R_2) = ({}^\bullet R_1 \setminus R_2^\bullet) \cup ({}^\bullet R_2 \setminus R_1^\bullet)$.

It is readily shown that the composition of non-deterministic quasi-interfaces is associative and one does not need to require that the individual components have disjoint output sets to ensure that property. Thus non-deterministic quasi-interfaces equipped with this composition operation is a commutative monoid (whose neutral element is the empty interface). Moreover $R \bowtie R = \langle R \rangle$ and since $\langle R \rangle = R$ when R is an interface, we deduce that the composition of interfaces is idempotent.

The natural extension of the implementation order in the non-deterministic case is to let

$$R_1 \leq R_2 \iff (\forall A \in \Omega)\ (\forall X \in R_1(A))\ (\exists Y \in R_2(A)) \cdot Y \subseteq X$$

expressing that anything that R_1 can do R_2 can do better (namely, it can do the same using fewer external services). This is a pre-order (reflexive and transitive

[3] Safely composability is the natural extension to the non-determinstic case of the composability of deterministic interfaces. However since we first defined the operation of composition of non-deterministic (quasi-) interfaces we felt obliged to add this qualifying term.

relation) where two non-deterministic interfaces are equivalent when their images for each A have the same sets of minimal elements (for set inclusion) or equivalently have identical upward closures. Say that an interface is saturated if $R(A)$ is upward-closed for every $A \in \Omega$. And it is reduced is for every $A \in \Omega$ any two elements of $R(A)$ are incomparable. Thus any non-deterministic interface is equivalent to its upward-closure (a saturated interface) and to its restriction to its set of minimal elements (a reduced interface). The order relation on saturated interfaces is simply given by the pointwise set-theoretic inclusion:

$$R_1 \leq R_2 \iff (\forall A \in \Omega) \quad R_1(A) \subseteq R_2(A)$$

Thus their least upper bounds are given by pointwise set-theoretic union: $(\bigvee_i R_i)(A) = \bigcup_i R_i(A)$ and their greatest lower bounds by pointwise set-theoretic intersection: $(\bigwedge_i R_i)(A) = \bigcap_i R_i(A)$. The following properties then immediately follow:

- distributivity: $\bigvee_i \bigwedge_j R_{i,j} = \bigwedge_j \bigvee_i R_{i,j}$.
- $(\bigvee_i R_i)^\bullet = \bigcup_i R_i^\bullet$ but also $(\bigwedge_i R_i)^\bullet = \bigcap_i R_i^\bullet$ because the sets $R_i(A)$ are upper closed sets and thus all those which are not empty contain Ω and thus have non empty intersection.

Note that the greatest lower bound of an arbitrary set of (saturated and non-deterministic) interfaces is also given by $(\bigwedge_i R_i)(A) = \{\cup_i X_i \mid \forall i \;\; X_i \in R_i(A)\}$.

Lemma 3.7.1. The composition commutes with joins:

$$R \bowtie (\bigwedge_i R_i) = \bigwedge_i (R \bowtie R_i).$$

Proof. The operation $\bowtie$ is monotonic in each of its argument and thus $R \bowtie (\wedge_i R_i) \leq \wedge_i (R \bowtie R_i)$. We are left to prove the converse relation. By definition of the sequential composition of relations and the greatest lower bound of interfaces it follows that $R; (\wedge_i R_i) = \wedge_i (R; R_i)$, and thus $\wedge_i R_i^n = (\wedge_i R_i)^n$ by induction on n. Hence $\wedge_i R_i^* = \wedge_i (\vee_n R_i^n) = \vee_n (\wedge_i R_i^n) = \vee_n (\wedge_i R_i)^n = (\wedge_i R_i)^*$. It follows that $\langle \wedge_i R_i \rangle (A) = (\wedge_i R_i)^*(A) \cap \wp(\wp(\Omega \setminus (\wedge_i R_i)^\bullet)) = (\cap_i R_i^*(A)) \cap \wp(\wp(\Omega \setminus \cap_i R_i^\bullet)) =$

$(\cap_i R_i^*(A)) \cap \wp(\wp(\cup_i(\Omega \setminus R_i^\bullet))) \supseteq (\cap_i R_i^*(A)) \cap (\cap_i \wp(\wp(\Omega \setminus R^\bullet{}_i))) = \cap_i \langle R_i \rangle(A)$ i.e. $\langle \wedge_i R_i \rangle \geq \wedge_i \langle R_i \rangle$. Thus $R \bowtie (\wedge_i R_i) = \langle R \vee (\wedge_i R_i) \rangle = \langle \wedge_i (R \vee R_i) \rangle \geq \wedge_i \langle R \vee R_i \rangle = \wedge_i (R \bowtie R_i)$. □

Since we can compute the least upper bound of an arbitrary family of (saturated and non-deterministic) interfaces, one can directly define the residual operation as:

Definition 3.7.3. The residual of two interfaces is given by:

$$R/R_1 = \bigwedge \{R' \mid R \leq R_1 \bowtie R'\}$$

which satisfies (almost by definition!) the required property of residuals:

Proposition 3.7.2. $R/R_1 \leq R' \iff R \leq R_1 \bowtie R'$.

Proof. $R \leq R_1 \bowtie R' \Rightarrow R/R_1 \leq R'$ by definition of the residual. Conversely let us assume that $R/R_1 \leq R'$ then by monotony $R_1 \bowtie R/R_1 \leq R_1 \bowtie R'$. By lemma 3.7.1 $R_1 \bowtie R/R_1 = R_1 \bowtie \wedge \{R'' \mid R \leq R_1 \bowtie R''\} = \wedge \{R_1 \bowtie R'' \mid R \leq R_1 \bowtie R''\} \geq R$ (since R is a lower bound of the given set). Hence $R \leq R_1 \bowtie R'$. □

3.8 Conclusion

This work is a first attempt to develop an interface theory for distributed collaborative systems in the context of service-oriented programming. We defined a notion of interface in order to explicit how a module can be used in a given environment using an assume/guarantee approach: we describe the set of services that can be provided by the module under the assumption that some other services are available in its environment. We have then defined 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. We intend to use residuation to define and structure the activities of crowdsourcing system actors. 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 behavior. Such a system can be implemented by Guarded Attribute Grammars and interfaces can be used to type applications.

We have mainly worked with basic (deterministic) interfaces. We have nonetheless shown how to extend the approach to non-deterministic interfaces. In the non-deterministic case it becomes possible to define an additional operation, namely an

operation that gives the co-restriction $R\lfloor I$ of an interface R to a set of services $I \subseteq \Omega$ by letting $(R\lfloor I)(A) = \{X \subseteq I \mid X \in R(A)\}$. That operation states how a role can be used when the set of services actually provided by the environment is known (to be I). This operation can be used to identify the usefulness of a component given by its interface knowing which services are actually available in the environment. Note that one can define the corresponding operation on grammars with $NI(G\lfloor I) = NI(G)\lfloor I$: $G\lfloor I$ by deleting all productions whose right-hand side contains a grammatical symbol in ${}^{\bullet}G \setminus I$.

On that basis one can enrich the information to take qualitative information into account to cope with uncertainty or time constraints. Actually it might be possible that we have only a partial knowledge of the set of available services in the environment in the form of a believe function [88] or a possibilistic distribution [89]. The interface should then allow us to quantify the possibility of realizing a service given this information on the environment. Similarly information can also be added on time execution, for instance by limiting behavior to ensure that services are delivered within certain time constraints..

CHAPTER 4

A ROLE-BASED BUSINESS COLLABORATION DESIGN APPROACH

4.1 Introduction

The concept of role is central in any collaborative or cooperative system. This importance is motivated by the fact that each contributor must have a clear framework within which he collaborates with other contributors. In this context, a role specifies both what the system expects from contributors, but also what contributors expects from the system; thus avoiding that a contributor be overwhelmed by information (or tasks) that he doesn't need. Traditionally, collaborative systems lose flexibility as soon as their design makes use of roles; because only static role description mechanisms, based on intuitive concepts, are available [90, 19]. Indeed, a dynamic context of collaboration further complicates a design of such systems, since involved entities evolve over time in number and in skills. An improvement would be to provide two things: firstly being able to specify roles clearly and rigorously, while ensuring flexibility for collaboration, and secondly provide an abstraction basis, for dynamic collaborative system design with tools for dynamic workflow management; in order to make collaboration more productive and effective. This design approach

implies a prior rigorous definition, of the set of roles and the relations between them, necessary to describe the functioning of a target domain. The role is seen here as a particular concern in the system to be modeled. In fact, a role-based design approach is similar to a separation of concern technique [91, 15, 92], applied to business process design, and enables inter-organizational flexible process design [93, 94, 95]. It is implemented by specifying activities defining the process steps, as well as flows describing coordination of these activities, as it may be done with BPMN [96, 97] orchestration or UML collaboration or activities diagrams [98]. Applying a role-based approach requires a clear separation of infrastructure mechanisms (communication, storage, service discovery, etc.) seen as primitives, necessary for its collaboration within its environment on the one hand, and functional goal i.e its effective contribution to collaboration on the other hand. By considering all functional goals present in a given system context, we get a formal basis for business objectives of the whole dynamic system.

4.2 Context, role and collaboration

4.2.1 Context of collaboration

In a targeted domain, a context of collaboration is the set of roles readily available for the realization of some activity. A context of collaboration is dynamic, i.e roles involved vary over time. For instance, we will consider the context on Figure 4.1 as a running collaborative context for the next parts of this work.

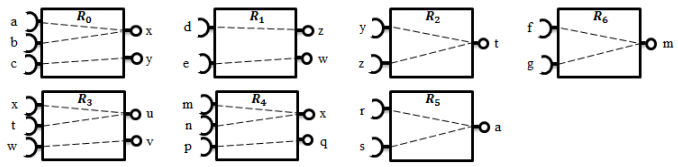


Figure 4.1: Running collaboration context of a system

4.2.2 A grammatical modeling of role concept

Each role is associated with a grammar describing its business skills, i.e. it's contribution in a given collaborative process. We recall that a production of the grammar is given by a left part, indicating the non-terminal to be expanded and a right part, describing how to expand this non-terminal. We interpret a production as a means of decomposing a task, the symbol on the left-hand side, as sub-tasks associated with the symbols on the right-hand side. A role specifies the set of actions that a given contributor can take in the system. For simplification purpose, consider role type and role as equivalent, simply called role (see section 1.2.2 for more details). A role specifies the set of actions or business skills devolved to a given actor in the system. We also recall that, as the goal is to describe processes, we are not yet interested with attributes in guarded attribute grammars.

Definition 4.2.1. A role r is given by a couple $(\mathcal{G}, R)$ where $\mathcal{G}$ is a guarded attribute grammar (GAG) [59] specifying role r business skills, and R it's associated interface of role.

4.2.3 Role collaboration

We talk about collaboration between two roles, when there is a service dependency between these two roles. This dependency can be direct, in which case it is a direct collaboration, likewise, the dependency can be indirect and in that case it is an indirect collaboration.

Definition 4.2.2. Two roles $r_1 = (\mathcal{G}_1, R_1)$ and $r_2 = (\mathcal{G}_2, R_2)$ are in a direct collaboration iff $R_1 \bowtie R_2$ holds. We denote $({}^{\bullet}R_1 \cap R_2^{\bullet}, r_2, r_1)$ that collaboration, labeled by ${}^{\bullet}R_1 \cap R_2^{\bullet}$, the set of services for which r_1 and r_2 collaborate; r_1 being the services requester, while r_2 is the provider of those services.

Thus r_1 and r_2 will be in an indirect collaboration iff $\exists r_k = (\mathcal{G}_k, R_k)$ such that $R_1 \bowtie R_k \bowtie R_2$ holds.

4.2.4 Potential direct collaborations of a role

A potential direct collaborations of a role, is a graph showing all potential services providers for that role in a collaborative context. Let r_0 be a given role; algorithm

1, determines potential direct collaborations (or potential dependencies) of r_0, in a context $\mathbf{R}$, in which r is member and such that $\forall r_i \in \mathbf{R},\ r_i = (\mathcal{G}_i, R_i)$.

```
1 input:   r_0 = (G_0, R_0) , R
2 output: C //set of potential collaborations of role r_0.
3
4    C ← ∅
5    rPDG(r_0, R) =
6            forall r_i in R
7               if R_0 ⋊ R_i then C ∪ {(•R_0 ∩ R_i^•, r_i, r_0)}
```

Algorithm 1: Role potential dependencies graph (rPDG) calculus

In our running context of figure 4.1, applying algorithm 1 to determine potential dependencies of role r_3, will result to graph below figure 4.2, given in extension by $\mathcal{C}$ below.

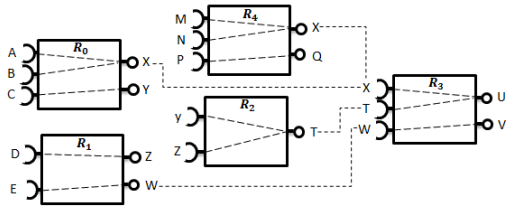


Figure 4.2: Direct potential collaborations of role r_3, in figure 4.1 context

$$\mathcal{C} = \{(\{x\}, r_0, r_3), (\{x\}, r_4, r_3), (\{t\}, r_2, r_3), (\{w\}, r_1, r_3)\}$$

4.3 Collaboration schemes, service workflow

4.3.1 Induced potential dependencies graph (iPDG)

A role collaboration scheme is a potential dependencies graph, induces by a context. From any context $\mathbf{R}$, an induced potential dependencies graph (iPDG) is obtained by grouping, step by step, all the potential dependencies of the various roles in the context, as implemented by the algorithm 2. Applying that algorithm on the previous context $\mathbf{R}$, we obtain collaboration scheme given by equation 4.1.

```
1 input: R
2          R' //roles whose rPDG have already been determined, initially empty.
3 output: C
4
5   R' ← ∅
6   iPDG(R, R') =
7          if (r_i in R) and (R ≠ ∅) then
8                 rPDG(r_i, R ∪ {R' ∪ {r_i}}) ∪ iPDG(R \ {r_i}, R')
9          else iPDG(R \ {r_i}, R')
```

Algorithm 2: Context $\mathbf{R}$ induced potential dependencies graph (iPDG)

$$\begin{aligned}\mathcal{C} = \{ \quad & (\{a\}, r_5, r_0), (\{m\}, r_6, r_4) \\ & (\{y\}, r_0, r_2), (\{z\}, r_1, r_2) \\ & (\{w\}, r_1, r_3), (\{x\}, r_4, r_3) \\ & (\{x\}, r_0, r_3), (\{t\}, r_2, r_3) \\ \} &\end{aligned} \tag{4.1}$$

4.3.2 Potential workflow of a service

The induced workflow of a service s_0, describes how this service will be issue; and is implemented as a dependency subgraph, derived from the induced potential dependencies graph ($iPDG$) of the role providing service s_0.

Consider a predicate $depend(l, s_0)$ with $l, s_0 \in \Omega$, which returns $true$ if service s_0 depends on the service l and $false$ otherwise. We define function $dependOn(s_0, \mathbf{R}) = \{(L, r) \mid L \subset \Omega^*, r \in \mathbf{R}\ and\ \forall l \in L\ depend(l, s_0)\}$ which seeks in context $\mathbf{R}$, all the roles involved in the process of providing service s_0, as well as the associated required services, and returns a list of couples with on one hand a set of required services L and the role r requesting these services on the other hand.

We also let

$$iPDG(\mathbf{R}, \emptyset) \restriction L_k = \{(l_0, r_i, r_j) \mid \forall (L_k, r_k) \in dependOn(s_0, \mathbf{R}),\ l_0 \in L_k\ and\ r_j = r_k\}$$

be a filtering made on the induced potential dependencies graph, concerning

collaborations (l_0, r_i, r_j) such as for any couple $(L_k, r_k) \in dependOn(s_0, \mathbf{R})$, r_k being requester of service l_0 with $(l_0 \in L_k \, et \, r_j = r_k)$.

Inputs: $Serv = \emptyset \cup \{s_0\}, \boldsymbol{R}$
output: $\mathcal{C}$ //a set of potential collaborations needed to provide the service s_0.

workflow$(Serv, \boldsymbol{R}) =$
  if s_i in $Serv$ then $-\ i \in \{1, \cdots, |Serv|\}$
    $ns = Serv \setminus \{s_i\} \cup \{s \mid (s, r) \in dependOn(s_i, \boldsymbol{R})\}$
    $\{\boldsymbol{iPDG}(\boldsymbol{R}, \emptyset) \restriction L_k\} \cup$ workflow$(ns, \mathbf{R}, \boldsymbol{iPDG}(\boldsymbol{R}, \emptyset))$
   where $(L_k, r_k) \in dependOn(s_0, \mathbf{R})$

Algorithm 3: Determining a potential workflow for a service

Algorithm 3, generates the potential workflow of a given service s_0, from a context $(\mathbf{R})$, having the associated induced potential dependencies graph $(\boldsymbol{iPDG}(\boldsymbol{R}, \emptyset))$. For instance, the potential workflow of the service u, obtained by applying that algorithm in the context of previous figure 4.1, is presented in the figure 4.3 below, in which unnecessary services and dependencies are watermarked.

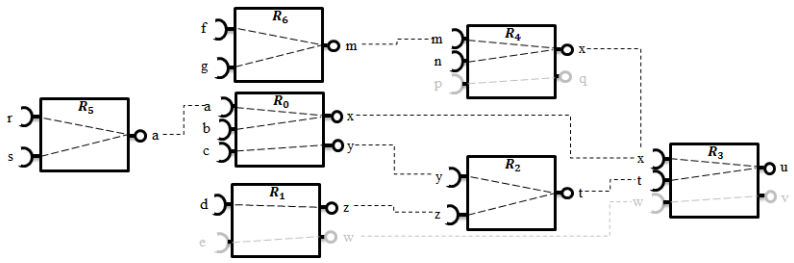


Figure 4.3: Service u potential workflow

4.3.3 Factorizing a workflow

A collaboration scheme can include several alternatives in supplying the same service. Such a case corresponds to the one of figure 4.4(a), where service x is provided by roles r_0 and r_4 respectively.

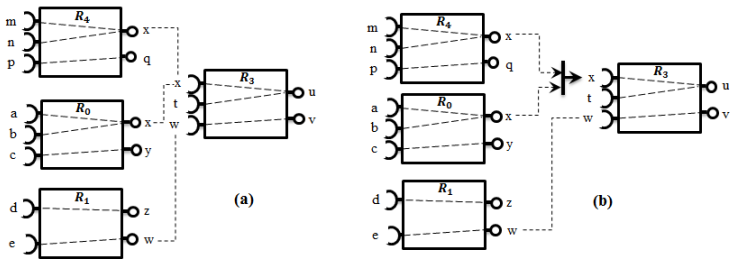


Figure 4.4: Factorizing a workflow - (a) a collaboration scheme. (b) the factorized collaboration scheme equivalent to the one on (a).

In a given context factorizing, is to be able to transform cases such as for a role r_3, requesting service x, so that we have service x potential suppliers list; as it is shown on figure 4.4(b).

Definition 4.3.1. An $\mathcal{F}-$*collaboration* is a triplet $({}^\bullet R_0 \cap R_i^\bullet, \{r_i, \cdots r_k\}, r_0)$, where $r_1, \cdots r_k$ are potential providers of services elements of set ${}^\bullet R_0 \cap R_i^\bullet$ and r_0 is the requester for those services (with ${}^\bullet R_0 \cap R_i^\bullet = \cdots = {}^\bullet R_0 \cap R_k^\bullet$), for some i and j.

A factorized collaboration scheme is then a potential dependency graph, possibly consisting of *collaborations* (i.e $({}^\bullet R_0 \cap R_1^\bullet, r_1, r_0)$), and $\mathcal{F}-$*collaboration* $(({}^\bullet R_0 \cap R_i^\bullet, \{r_i, \cdots r_k\}, r_0))$ if necessary.

```
1  input: C
2  output: C'
3    C' ← ∅
4    factorize(C) =
5      forall c in P(C)
6        if |c| ==1 then //c is like {(•R0 ∩ R1•, r1, r0)}
7            C' ∪ c
8        else if |c| > 1 then //c is like {(•R0 ∩ R1•, r1, r0), ··· , (•R0 ∩ R|c|•, r|c|, r0)}
9            C' ∪ {(•R0 ∩ R1•, {r1, ··· , r|c|}, r0) |
10                r1≤i≤|c| ∈ provider (•R1 ∩ R|c|•, c) and r0 == requester(c)}
```

Algorithm 4: Factorizing a service

Let $P(\mathcal{C}) \subseteq \mathcal{P}(\mathcal{C})$ a subset of parts of set $\mathcal{C}$, where elements are grouped subsets of all identically labeled collaborations. Algorithm 4 transforms a $\mathcal{C}$ collaborations scheme to a $\mathcal{C}'$ factorized collaborations scheme. Where function *provider* $\left({}^\bullet R_1 \cap R_{|c|}^\bullet, c\right)$ returns the list of service providers, labeled by elements of set ${}^\bullet R_1 \cap R_{|c|}^\bullet$ in a c

collaboration set, while $requester(c)$ checks if r_0 is the requester in each case of collaboration.

4.4 Activity in collaborative context

4.4.1 Formal definition

An activity is a process description in term of a role collaboration scheme, by which a given service s_0 can be rendered. Activity is then given by the pair $(s_0, workflow(\{s_0\}, \mathbf{R}))$, where $workflow(\{s_0\}, \mathbf{R})$ is service s_0 potential workflow, derived from the context $\mathbf{R}$.

Definition 4.4.1. An activity for a given service s_0, in a context $\mathbf{R}$, is a couple denoted $activity_{s_0} = (s_0, workflow(\{s_0\}, \mathbf{R}))$, and is the process of supplying service s_0, described by $workflow(\{s_0\}, \mathbf{R})$ his potential workflow.

Let us consider the process of delivering service u, given by equation (4.2) and described graphically in the figure 4.5 below.

$$\begin{aligned} activity_u \quad = \quad & \Big(u, \big[(x, r_4, r_3), \\ & (x, r_0, r_3), (t, r_2, r_3), \\ & (m, r_6, r_4), (a, r_5, r_0), \\ & (y, r_0, r_2), (z, r_1, r_2)\big]\Big) \end{aligned} \tag{4.2}$$

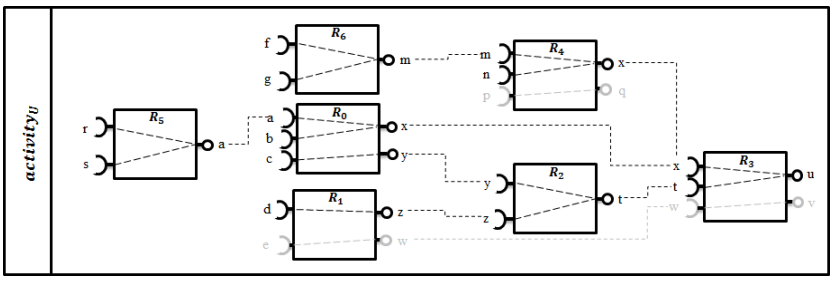


Figure 4.5: Activity $activity_u$ workflow

An activity can have several occurrences of the same role (indifferently supplier or requester). If two roles r_0 and r_1 respectively, provide the same service s_0 within an activity, then they do not necessarily use the same required services i.e. $rPDG(r_0, \mathbf{R}) \neq rPDG(r_1, \mathbf{R})$.

Proposition 4.4.1. Two activities $activity_{s_0}$ and $activity_{s_1}$ are equivalent, denoted by $activity_{s_0} \equiv activity_{s_1}$, iff $s_0 = s_1$ and $workflow(\{s_0\}, \mathbf{R}) \cong workflow(\{s_1\}, \mathbf{R})$ i.e they deliver the same service in context $\mathbf{R}$, with $s_0, s_1 \in \Omega$.

Proof. Consider $R_1 \dots R_m$ as pairwise composable role interfaces involved in a given $workflow(\{s_0\}, \mathbf{R})$ and $R_1^{'} \dots R_n^{'}$ those of $workflow(\{s_1\}, \mathbf{R})$ respectively, with $m \neq n$. Let $R = \rtimes_{i=1}^{m} R_i$ and $R' = \rtimes_{i=1}^{n} R_i$ their respective cascade compositions. By proposition 4.5 in [8, 9], those compositions are associative. As by hypothesis those workflows render same services, we have $s_0 = s_1$ and $s_0 \in R^{\bullet} \cap R'^{\bullet}$, two cases can be distinguished: whether $m > n$ and then $R' \subseteq R$, we say R' realizes service s_0 with less business rules than R; or $m < n$ so $R \subseteq R'$ and as R', R realizes service s_0 with less business rules. □

4.4.2 Atomicity of an activity

An activity for a given service s_0, is said to be atomic [15], if it has only one occurrence of role supplier for each required service in that activity. For example, the atomic forms of activity $activity_u$ in the previous equation 4.2, are respectively:

$$\begin{aligned} activity_{u_0} \quad = \quad & \Big(u, \big[(x, r_0, r_3), \\ & (t, r_2, r_3), (a, r_5, r_0), \\ & (y, r_0, r_2), (z, r_1, r_2)\big]\Big) \end{aligned} \tag{4.3}$$

$$\begin{aligned} activity_{u_1} \quad = \quad & \Big(u, \big[(x, r_4, r_3), \\ & (t, r_2, r_3), (m, r_6, r_4), \\ & (y, r_0, r_2), (z, r_1, r_2)\big]\Big) \end{aligned} \tag{4.4}$$

Definition 4.4.2. An activity $activity_{s_0} = (s_0, workflow(s_0, \mathbf{R}))$ is atomic, iff for all (s_0, r_i, r_j) and (s_0, r_k, r_j) in $workflow(s_0, \mathbf{R})$, $r_i = r_k$.

4.4.3 Activities functional decomposition

An activity can be progressively fragmented into a set of atomic activities. The principle of decomposition, is based on roles (concerns), and states that, as long as there are several occurrences of the same role r in an activity, this activity is broken down into new activities containing a single role occurrence r. This principle

is repeated until all the activities obtained are atomic [15]. For this, the associated workflow must be factorized; if at the end of this, $\mathcal{F} - collaborations$ exist, that means the activity is decomposable, according to the principle described by the algorithm 5.

Proposition 4.4.2. Consider $activity_{s_0} = (s_0, workflow_0(s_0, \mathbf{R}))$ and $activity'_{s_0} = (s_0, workflow_1(s_0, \mathbf{R}))$ two activities, where $activity_{s_0} \equiv activity'_{s_0}$. $activity_{s_0}$ is decomposable to $activity'_{s_0}$ iff $workflow_0(s_0, \mathbf{R}) \neq workflow_1(s_0, \mathbf{R})$
and $factorize(workflow_0(s_0, \mathbf{R})) = workflow_1(s_0, \mathbf{R})$

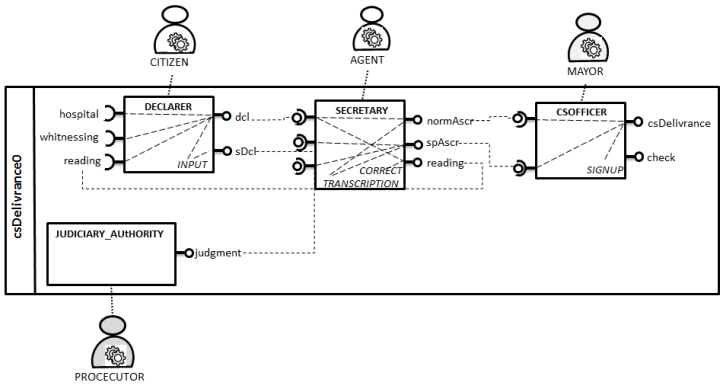
Proof. As the two activities are equivalents by hypothesis, the demonstration is equivalent to show that a in a workflow, several collaborations for given service, is equivalent to an $\mathcal{F}-$collaboration on the same service; and this is done by definition 4.3.1. □

```
1 input: C' //factorized workflow of the activity .
2 output: C //set of potential atomic workflows.
3   C ← ∅
4   decomp(C', C) =
5         forall c in C'
6           if |provider(c)| == 1 then //c is like {(•R0 ∩ R1•, {r1}, r0)}
7                 decomp(C' \ {c}, insert (c, C))}
8           else if |provider(c)| > 1 then //c is like (•R0 ∩ R1•, {r1, ··· , r|c|}, r0)
9                 decomp(C' \ {c}, mdup (c, C))}
```

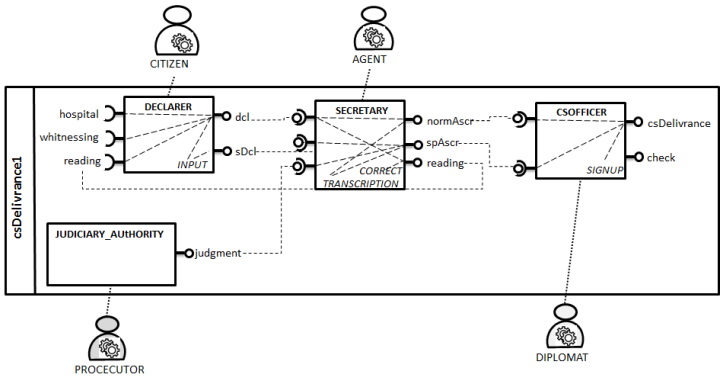
Algorithm 5: Atomic decomposition of an activity's workflow

Function **insert** $(c, \mathcal{C})$, insert collaboration c, in the different atomic workflows of $\mathcal{C}$, for which c is necessary. **mdup** $\left(\left\{(^\bullet R_0 \cap R_1^\bullet, \{r_1, \cdots, r_{|c|}\}, r_0)\right\}, \mathcal{C}\right)$ add in each atomic workflow of $\mathcal{C}$, collaborations $(^\bullet R_0 \cap R_1^\bullet, r, r_0)$ with $r \in \{r_1, \cdots, r_{|c|}\}$, as long as these collaborations are useful, for the realization of the service associated with that workflow.

Application of algorithm 5 on collaboration scheme of previous figure 2.3, result to two distinct way of carrying out activity *csDelivrance* as shown on figure 4.6



(a) Process of issuing a civil status certificate by the mayor



(b) Process for the issuance of a civil status certificate by a diplomat

Figure 4.6: Decomposition of the previous activity on figure 2.3, into two sub-activities, $csDelivrance0$ and $csDelivrance1$

below. Mainly, $csDelivrance0$ and $csDelivrance1$ are atomic activities describing the issuing of civil status certificate process, either by a Mayor or by a Diplomat respectively.

4.4.4 Activity realizability

The question of feasibility here, refers to the possibility of carrying out an activity, in a finite number of stages, and rendering provided service. This assumes that all the roles necessary for carrying out that activity must be available; we say it is a favorable context.

Termination of an activity, is conditioned by the fact that its workflow must contain autonomous roles at the start of the chain. These autonomous roles, playing activity trigger role, we also say they are service trigger roles. Algorithm 6 below, describes principle of test of feasibility for a workflow, by applying a filtering pattern

mechanism. A workflow is realizable if that algorithm returns $True$ and its required services queue($Serv$) is empty. In case that $False$ is returned. Required service queue contains a list of services, still to be provided, for completing activity.

```
1  input: Serv //set of required services
2         C //activity workflow
3  output: (Bool, Serv)
4    realizable(∅, _) = (True, ∅) //the workflow is realizable
5    realizable(Serv, ∅) = (False, Serv) // not realizable, Serv are required services
6    realizable(Serv, c ∈ C)
7          | Serv ∩ label(c) == ∅ =realizable(Serv ∪ req, C \ {c})
8          | Serv ∩ label(c) ≠ ∅ =realizable(Serv' ∪ req, C \ {c})
9            where
10                Serv' = Serv ↾ (x ∈ Serv ∧ x ∉ label(c))
11                req = dependOn(label(c), provider(c))
```

Algorithm 6: Checking realizability of $activity_{s_0}$

For a given collaboration $c = ({}^{\bullet}R_i \cap R_k^{\bullet}, r_k, r_i)$, $label(c)$ returns set ${}^{\bullet}R_i \cap R_k^{\bullet}$ of services labeling that collaboration, and $provider(c)$ returns r_k providing those services.

Remark 0. An activity $activity_{s_0} = (s_0, workflow(s_0, \mathbf{R}))$ is said to be almost realizable, if at least one of its atomic forms obtained by decomposition, ends; i.e there is a $C \in \mathbf{decomp}(workflow(s_0, \mathbf{R}), \emptyset)$ such that $\mathbf{realisable}(\{s_0\}, C) = (True, \emptyset)$. Similarly the activity $activity_{s_0}$ is said realizable, if all atomic forms end; i.e. whatever $C \in \mathbf{decomp}(workflow(s_0, \mathbf{R}), \emptyset)$, $\mathbf{realisable}(\{s_0\}, C) = (True, \emptyset)$.

4.5 Contributor of a business collaborative process

4.5.1 Concept and definitions

A contributor is any entity in a system capable of acquiring information, communicating with its environment and processing. For this, a contributor has several primitive. He can also play one or more roles in the system. Since a contributor knows how to instrument, communicate and process information he is considered an intelligent active space [59].

In this work, since a contributor is an entity which intervenes in a process by playing

a predetermined role, we will assume that it has an internal mechanism of communication with its environment. Similarly we will assume that it also has persistence mechanisms for its data, and finally, a publication and service discovery mechanism.

Definition 4.5.1. A contributor a_τ is given by a couple $(\mathbf{R}_\tau, \mathcal{C}_\tau)$, where $\mathbf{R}_\tau$ is the set of potential roles the contributor can play, and $\mathcal{C}_\tau$ is the set of constraints on those potential roles.

4.5.2 Constraints on contributor's potential roles

Let $a_\tau = (\mathbf{R}_\tau, \mathcal{C}_\tau)$ an actor, r_i and r_j two roles; an association between contributor a_τ and roles r_i and r_j is materialized by the fact that $r_i, r_j \in \mathbf{R}_\tau$. We will say for instance that r_i,r_j are contributor's a_τ potential roles.

It is possible to define constraints, on the potential roles of a contributor [16]. Consider role pairs (r_i, r_j) such that $r_i, r_j \in \mathbf{R}_\tau$ with $1 \leq i, j \leq |\mathbf{R}_\tau|$; four constraint values can be associated with these pairs roles (see section 1.2.3), namely:

Dcr or nothing, for no constraint;

$\boldsymbol{Imp\,(r_i, r_j)}$ indicating that if actor a_τ plays role r_i, then he must also play role r_j;

$\boldsymbol{Eqv\,(r_i, r_j)}$ in case both $\boldsymbol{Imp\,(r_i, r_j)}$ and $\boldsymbol{Imp\,(r_j, r_i)}$ holds;

$\boldsymbol{Phb\,(r_i, r_j)}$ in this case, actor a_τ playing role r_i, cannot play role r_j.

4.5.3 Relation "play a role"

A contributor can play one or more roles, within an activity, or in several parallel activities. So one can distinguishes several cases of implementation of "play" relation.

4.5.3.1 Case 1: Playing several roles in an activity

A contributor actor can play more than one role within an activity; provided that those roles do not provide the same services.

(a) Contributor a_τ playing two roles

(b) Workflow simplified by the direct product of R_0 and R_1

Figure 4.7: A contributor playing several roles in an activity

A contributor a_τ playing several roles r_0 and r_1 respectively, in $activity_{s_0} = (s_0, [\cdots, (B, r_1, r_2), (C, r_2, r_0), \cdots])$, see figure 4.7(a), is multi-skilled in that activity. Therefore, the different roles of a_τ can be pooled into a single macro-role $r_0^{'}$ whose interface is $R_0 \times R_1$. So activity $activity_{s_0}$ can undergo a transformation to become $activity_{s_0}^{'} = \left(s_0, \left[\cdots, (B, r_0^{'}, r_2), (C, r_2, r_0^{'}), \cdots\right]\right)$, as illustrated in the figure 4.7(b).

4.5.3.2 Case 2: crowdsourced role played by several contributors

Several contributors can play the same role within an activity.

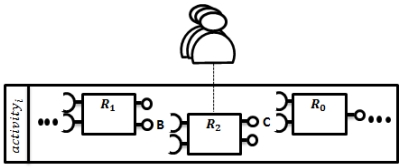


Figure 4.8: Several contributors playing same role r_2 in an activity

Within an activity, several contributors can play the same role; which assumes that several instances of this same role can be created. In a workflow, if role r is played by several actors (figure 4.8), then any collaboration with r will be on a crowded task.

4.5.3.3 Case 3: Competing activities

A contributor can play roles (identical or different) concomitantly in several parallel activities.

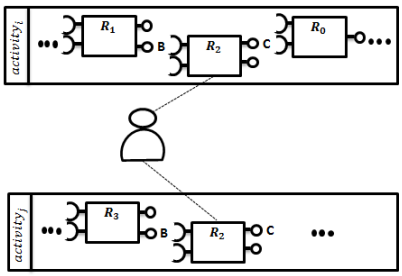


Figure 4.9: A contributor involved in two parallel activities

A contributor can participate in several activities at the same time, either by playing the same role each time (see figure 4.9), or by having different roles. In all these cases, each of these activities is implemented individually as in case 1 above.

4.5.4 Implementation of the "play" relation

Consider three primitives $\overline{\textbf{play}}\left(a_\tau, \{r_i\}_{1\leq i\leq|\mathbf{R}_\tau|}, activity_{s_0}\right)_{\{cstr_k\}_{1\leq k\leq N}}$ indicating that a_τ potentially can play roles $\{r_i\}_{1\leq i\leq|\mathbf{R}_\tau|}$ in activity $activity_{s_0}$, with constraints $\{cstr_k\}_{1\leq k\leq N}$; $\overline{play}\,(a_\tau, r_i, activity_{s_0})_{cstr_k}$ to express that a_τ potentially can play the roles r_i in activity $activity_{s_0}$, according to the constraint $cstr_k$, with

$$\overline{\textbf{play}}\left(a_\tau, \{r_i\}_{1\leq i\leq|\mathbf{R}_\tau|}, activity_{s_0}\right)_{\{cstr_k\}_{1\leq k\leq N}} = \bigcup_{1\leq i\leq|\mathbf{R}_\tau|} \overline{play}\,(a_\tau, r_i, activity_{s_0})_{cstr_k}$$

and finally $play\,(a_\tau, R_i, activity_{s_0})$ expressing that a_τ actually plays the role r_i whose interface is R_i, in activity $activity_{s_0}$.

Possible implementations of the "play" relation while taking account of constraints on roles, are described by equations 4.5 given below:

$$\begin{aligned}
\overline{play}\,(a_\tau, r_i, activity_{s_0})_{Dcr} &= play\,(a_\tau, R_i, activity_{s_0}) \\
\overline{play}\,(a_\tau, r_i, activity_{s_0})_{Imp(r_i,r_j)} &= play\,(a_\tau, R_i \times R_j, activity_{s_0}) \\
\overline{play}\,(a_\tau, r_i, activity_{s_0})_{Eqv(r_i,r_j)} &= play\,(a_\tau, R_i \times R_j, activity_{s_0}) \\
&\qquad \textbf{or } play\,(a_\tau, R_j \times R_i, activity_{s_0}) \\
\overline{play}\,(a_\tau, r_i, activity_{s_0})_{Phb(r_i,r_j)} &= play\,(a_\tau, R_i, activity_{s_0}) \\
&\qquad \textbf{and not } play\,(a_\tau, R_j, activity_{s_0})
\end{aligned} \tag{4.5}$$

4.6 Conclusion

The main interest in this chapter, was to outline a role-based design approach, for business collaboration processes. The role concept has been formally defined, as well as associated readjustment and switching mechanisms, to provide a certain flexibility to design. Collaboration scheme was formally defined, in the form of a given service workflow, thus making it possible to describe what is an activity in a given context. Finally, the notion of contributor and it's interaction mechanisms was presented, thus offering the whole range of tools necessary for this design approach. Architecturally, role-based design approach is similar to agent design approach, but differ on how job is carried out. In agent systems, a contributor will do the job according to its knowledge to the whole system [99, 90, 14], while here, a contributor will ordinarily accomplish his tasks according to his own decisions.

CONCLUSION AND PERSPECTIVES

Summary of thesis achievements

At this point of our dissertation, let us recall that the initial problem was to formalize a role-based approach for the design of business collaboration processes in a dynamic context where recurrent reorganization of work within collaborative structures occur, but also the need of incrementally integrating particularly heterogeneous sectors of activity in the collaboration. Our approach made a clear separation between contributor capabilities (intrinsic skills) and what he does in a collaboration (business role). Considering all roles involved in a particular context of collaboration offers a formal basis for reasoning. As there are mainly two goal-oriented collaborative approaches, we focused on servicing collaboration types and we benefited from the GAG formal framework, the collaboration processes description mechanisms. We improved GAG by adding three extension elements, namely: (1) a tasking model making it possible to describe a design for services provision without ambiguity and for any form of business collaboration process. (2) a notion of role interface, as an integration of single responsibility and interface segregation principles in GAGs, thus offering flexible mechanisms for services composition. (3) A role-based design approach, which is a form of user-centered approach, with roles assigned to each contributor in the process. Thus a contributor is perceived both in terms of its intrinsic skills and the role he plays in the collaboration. This approach allows us to monitor and manage all the roles present in a given context, to substitute roles, to compose them, in order to choreograph a given activity.

To conclude, we discuss the key features of our design approach and draw some future research directions.

Assessment of the design approach

Guarded attribute grammars provide a modular, declarative, user-centric, data-driven, distributed and reconfigurable formal design tool for business collaboration. It favors flexible design and execution of business process since it possesses concurrency, modularity, reconfiguration, distribution, interoperability properties [65]. These properties are directly inherited by the role-based approach, since it has GAGs as a basis. In addition, this approach guarantees many of the design principles and properties set out in section 1.4; also as a process modeling, those of understandability and maintainability [28, 100].

Cohesion and coupling: our approach design unit is a role. A role is described in terms of services provided, services required, internal services and relationship between them; that is, a role wrap together semantically close elements. Thus, within a role, cohesion is strong. Also, interface of role and its mechanisms, provides flexibility in composition, making coupling weak. These requirements of cohesion and coupling, being necessary for a good design, guarantee the single responsibility principle; and then one could say that the role-based approach encapsulate the separation of concerns principle.

Scalability: as setting operators like restriction, helps to hide not needed functionalities of an interface, together with composition mechanisms associated with role interfaces, this approach facilitates the dynamic redefinition of roles available in a context, thus respecting the principle of interface segregation.

Reusability: this approach is concerned with building collaboration, based on use of existing roles in the collaborative process. These roles are designs and byproducts of the design life cycle, viewed as stand alone units.

Maintainability: as a role is single responsible, possible failures in processes may be easily detected, identified, isolated and corrected.

Future works

Defining a data model for business collaboration: An immediate extension of this work is to include data handling in the design approach. For this, it is necessary to define a graphical syntax for the business artifacts, as well as to explain the interaction mechanisms between them, which can highlight the data manipulated in the business collaboration, and define data aggregation and splitting operators; [6] could be a basis to this work.

A dynamic tasking model for a coworking context: the coworking approach to collaboration is goal oriented; it assumes an unstructured production process, and it is through contributors interactions that production rules are defined. From the GAG architecture (see section 1.5.1) describing such a business collaboration approach supposes starting from business artifacts, in order to infer the rules that have favored their evolution. Then the business rules life cycle is therefore closely linked to artifacts life cycle. It would be interesting to propose a design approach for such cases.

Modular DSL for roles and collaborations: We will be dealing here with the serialization of collaboration schemes, in a role-based design context. As we have seen previously, this approach favors the use of modules or libraries considered as a toolbox, for the assembly line development, of products conforming to a specific field of application. Since the role implementation (library) is seen as a domain specific language (DSL), capturing the semantics of an application domain, a collaboration scheme can reasonably be seen as a composition of DSLs. This is not achieved without certain difficulties, in particular the difficulty of conceiving and implementing new languages on the one hand, the little flexibility that a language offers in order to facilitate its evolution, and finally the collaboration between languages. This work can be carried out on the basis of the precedent work done on modular language design [7].

BIBLIOGRAPHY

[1] Thierry Burger-Helmchen and Julien Pénin. Crowdsourcing: Définition, enjeux, typologie. Management & avenir, Management Prospective Ed., 41:254–269, 2011.

[2] Gary P. Pisano and Roberto Verganti. Which kind of collaboration is right for you? Harvard Business Review, December 2008.

[3] Jacki O'Neill and David Martin. Relationship-based business process crowdsourcing? In Paula Kotzé, Gary Marsden, Gitte Lindgaard, Janet Wesson, and Marco Winckler, editors, Human-Computer Interaction – INTERACT 2013, pages 429–446, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[4] J.A. Goguen. Reusing and interconnecting software components. Computer; (United States), 19:2, 2 1986.

[5] Maurice Tchoupé Tchendji, Marcellin Atemkeng T., and Djeumen D. Rodrigue. Un modèle de documents stable par projections pour l'édition coopérative asynchrone. In Actes du Colloque Africain pour la Recherche en Informatique, pages 325–332, octobre 2014.

[6] Tchoupe Tchendji M., Djeumen Djatcha R., and Atemkeng M. A stable and consistent document model suitable for asynchronous cooperative edition. Journal of Computer and Communications, 5:69–82, 2017.

[7] Eric Badouel and Rodrigue Djeumen Djatcha. Modular Design of Domain-Specific Languages using Splittings of Catamorphisms. In Bernd Fischer and Tarmo Uustalu, editors, Theoretical Aspects of Computing – ICTAC 2018., volume 11187 of Lecture Notes in Computer Science, pages 62–79. Springer, Cham., October 2018.

[8] Eric Badouel and Rodrigue Djeumen Djatcha. Interfaces of Roles in Distributed Collaborative Systems. In CARI 2018 - Colloque Africain sur la Recherche en Informatique et Mathématiques Appliquées, pages 182–193, Stellenbosch, South Africa, October 2018.

[9] Eric Badouel and Rodrigue Aimé Djeumen Djatcha. A Calculus of Interfaces for Distributed Collaborative Systems: The Guarded Attribute Grammar Approach. Revue Africaine de la

Recherche en Informatique et Mathématiques Appliquées, Volume 31 - 2019 - CARI 2018, 2020.

[10] Rodrigue Aimé Djeumen Djatcha. A role-based collaborative process design on crowdsourcing systems. In Africain Conference on Research in Computer Science and Applied Mathematics, Thiès, Senegal, October 2020. Polytech School of Thiès.

[11] David Ferraiolo and Richard Kuhn. Role-based access control. In In 15th NIST-NCSC National Computer Security Conference, pages 554–563, 1992.

[12] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. Computer, 29(2):38–47, February 1996.

[13] Haiping Xu, Xiaoqin Zhang, and Rinkesh J. Patel. Developing role-based open multi-agent software systems. Technical report, INTERNATIONAL JOURNAL OF COMPUTATIONAL INTELLIGENCE THEORY AND PRACTICE (IJCITP, 2007.

[14] O Kazik. Role-based approaches to development of multi-agent systems: A survey. In WDS'10 Proceedings of Contributed Papers, pages 19–24, 01 2010.

[15] Artur Caetano, Antonio Rito Silva, and José Tribolet. Business process decomposition - an approach based on the principle of separation of concerns. Enterprise Modelling and Information Systems Architectures, 5(1):44–57, July 2010.

[16] Dirk Riehle and Thomas Gross. Role model based framework design and integration. SIGPLAN Not., 33(10):117–133, October 1998.

[17] W. Keith Edwards. Policies and roles in collaborative applications. In Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work, CSCW '96, pages 11–20, New York, NY, USA, 1996. ACM.

[18] Nicola Guarino. Concepts, attributes and arbitrary relations: Some linguistic and ontological criteria for structuring knowledge bases. Data & Knowledge Engineering, 8(3):249 – 261, 1992.

[19] Stephan Bögel, Stefan Stieglitz, and Christian Meske. A role model-based approach for modelling collaborative processes. Business Process Management Journal, 20(4):598–614, 2014.

[20] Marek Szelagowski. Static and dynamic processes. bpmleader, August 2014.

[21] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. Distributed and Parallel Databases, 14(1):5–51, 2003.

[22] Jörg Becker, Michael Rosemann, and Christoph von Uthmann. Guidelines of Business Process Modeling, pages 30–49. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[23] Dongsoo Kim, Minsoo Kim, and Hoontae Kim. Dynamic business process management based on process change patterns. In 2007 International Conference on Convergence Information Technology (ICCIT 2007), pages 1154–1161, 2007.

[24] Toma RUSINAITE, Olegas VASILECAS, and Diana KALIBATIENE. A systematic literature review on dynamic business processes. Baltic J. Modern Computing, 4(3):420–427, 2016.

[25] Wil M. P. van der Aalst. Business process management: A comprehensive survey. Hindawi Publishing Corporation, ISRN Software Engineering, February 2013.

[26] Mathias Weske. Business Process Management: Concepts, Languages, Architectures. Springer-Verlag, Berlin, Heidelberg, 2007.

[27] Mathias Kirchmer. Business Process Management: What Is It and Why Do You Need It?, pages 1–28. Springer International Publishing, Cham, 2017.

[28] Barbara Weber Simon Forster, Jakob Pinggera. Collaborative business process modeling, 2012.

[29] Olegas Vasilecas, Diana Kalibatiene, and Dejan Lavbič. Rule- and context-based dynamic business process modelling and simulation. Journal of System and Software, 122(C):1–15, December 2016.

[30] Edsger W. Dijkstra. On the Role of Scientific Thought, pages 60–66. Springer New York, New York, NY, 1982.

[31] Aspiring Craftsman. The art of separation of concerns. https://aspiringcraftsman.com/2008/01/03/art-of-separation-of-concerns/, January 2008.

[32] Robert C. Martin. OCP: The Open-Closed Principle(Object Mentor SOLID Design Papers). objectmentor.com, 1996.

[33] Robert C. Martin. LSP: The Liskov Substitution Principle(Object Mentor SOLID Design Papers). objectmentor.com, 1996.

[34] Robert C. Martin. DIP: The Dependency Inversion Principle(Object Mentor SOLID Design Papers). objectmentor.com, 1996.

[35] Robert C. Martin. Design Principles and Design Patterns(Object Mentor SOLID Design Papers). objectmentor.com, 2000.

[36] Robert C. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Robert C. Martin Series. Prentice Hall, Boston, MA, 2017.

[37] Robert C. Martin. SRP: The Single Responsibility Principle(Object Mentor SOLID Design Papers). objectmentor.com, 2005.

[38] Robert C. Martin. ISP: The Interface Segregation Principle(Object Mentor SOLID Design Papers). objectmentor.com, 1996.

[39] Mehmet Aksit, Bedir Tekinerdogan, and Lodewijk Bergmans. The six concerns for separation of concerns. In Workshop on Advanced Separation of Concerns (ECOOP 2001), 01 2001.

[40] Daniel Jackson and Michael Jackson. Separating concerns in requirements analysis: An example. In Rigorous Development of Complex Fault-Tolerant Systems [FP6 IST-511599 RODIN project], pages 210–225, 2006.

[41] Colin Atkinson and Thomas Kuehne. Separation of concerns through stratified architectures, 2000.

[42] Jonathan Aldrich. Challenge problems for separation of concerns. In In OOPSLA 2000 Workshop on Advanced Separation of Concerns, 2000.

[43] Lodewijk Bergmans and Mehmet Aksit. Composing crosscutting concerns using composition filters. Communications of the ACM, 44:51–57, 2001.

[44] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Aksit and Satoshi Matsuoka, editors, ECOOP'97 - Object-Oriented Programming, 11th European Conference, Jyväskylä, Finland, June 9-13, 1997, Proceedings, volume 1241 of Lecture Notes in Computer Science, pages 220–242. Springer, 1997.

[45] Alberto Sillitti, Tullio Vernazza, and Giancarlo Succi. Service oriented programming: A new paradigm of software reuse. In Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools, ICSR-7, pages 269–280, Berlin, Heidelberg, 2002. Springer-Verlag.

[46] Miroslaw Malek. The NOMADS republic - a case for ambient service oriented computing. In 2005 IEEE International Workshop on Service-Oriented System Engineering (SOSE 2005), 20-21 October 2005, Beijing, China, pages 9–12, 2005.

[47] Remco Dijkman and Marlon Dumas. Service-oriented design: a multi-viewpoint approach. International Journal of Cooperative Information Systems, 13(4):337–368, 2004.

[48] Didier Parigot and Baptiste Boussemart. Architecture orientée services dynamique: D-soa. Technical Report inria-00342310, Inria, 004, route des Lucioles BP 93 F-06902 Sophia-Antipolis cedex, France, November 2008.

[49] Guy Bieber, Lead Architect, and Isd Ci. Introduction to service-oriented programming. In Openwings, URL = http://www.openwings.org, 2001.

[50] E. A. Brewer, R. H. Katz, Y. Chawathe, S. D. Gribble, T. Hodes, Giao Nguyen, M. Stemm, T. Henderson, E. Amir, H. Balakrishnan, A. Fox, V. N. Padmanabhan, and S. Seshan. A network architecture for heterogeneous mobile computing. IEEE Personal Communications, 5(5):8–24, Oct 1998.

[51] Thomas Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[52] Thomas Erl, Ricardo Puttini, and Zaigham Mahmood. Cloud Computing: Concepts, Technology & Architecture. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2013.

[53] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: State of the art and research challenges. Computer, 40(11):38–45, November 2007.

[54] Eiko Yoneki. ECCO: Data centric asynchronous communication. Technical Report UCAM-CL-TR-677, University of Cambridge, Computer Laboratory, December 2006.

[55] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. ACM Comput. Surv., 35(2):114–131, June 2003.

[56] Ralph Mietzner, Christoph Fehling, Dimka Karastoyanova, and Frank Leymann. Combining horizontal and vertical composition of services. In Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2010. IEEE Computer Society, 2010.

[57] Tony Ho Tran. User-centered design: Definition, examples, and tips. https://www.invisionapp.com/inside-design/user-centered-design-definition-examples-and-tips/, June 2019.

[58] Sergey Gladkiy. User-centered design: Process and benefits. https://uxplanet.org/user-centered-design-process-and-benefits-fd9e431eb5a9, June 2018.

[59] Eric Badouel, Loïc Hélouët, Georges-Edouard Kouamou, Christophe Morvan, and Nsaibirni Robert Fondze, Jr. Active workspaces: Distributed collaborative systems based on guarded attribute grammars. ACM SIGAPP Applied Computing Review, 15(3):6–34, October 2015.

[60] Paul Klint, Ralf Lämmel, and Chris Verhoef. Toward an engineering discipline for grammarware. ACM Trans. Softw. Eng. Methodol., 14(3):331–380, July 2005.

[61] Donald Knuth. Semantics of context-free languages. MATHEMATICAL SYSTEMS THEORY, 2(2):127–145, 1968.

[62] H. H. Vogt, S. D. Swierstra, and M. F. Kuiper. Higher order attribute grammars. SIGPLAN Not., 24(7):131–145, June 1989.

[63] Jintae Lee, George M. Wyner, and Brian T. Pentland. Process grammar as a tool for business process design. MIS Quarterly, 32(4):757–778, December 2008.

[64] Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. In Proceedings of the 5th International Conference on Business Process Management, BPM'07, page 288–304, Berlin, Heidelberg, 2007. Springer-Verlag.

[65] Robert Nsaibirni. A Guarded Attribute Grammar Based Model for User Centered, Distributed, and Collaborative Case Management Case of the Disease Surveillance Process. Theses, Université de Yaoundé I, April 2019.

[66] David Cohn and Richard Hull. Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull., 32:3–9, 2009.

[67] Mahmood Hosseini, Keith Phalp, Jacqui Taylor, and Raian Ali. The four pillars of crowdsourcing: A reference model. In Marko Bajec, Martine Collard, and Rébecca Deneckère, editors, RCIS, pages 1–12. IEEE, 2014.

[68] Pavel Kucherbaev, Florian Daniel, Stefano Tranquillini, and Maurizio Marchese. Crowdsourcing processes: A survey of approaches and opportunities. IEEE Internet Computing, 20(2):50–56, 2016.

[69] Stefano Tranquillini, Florian Daniel, Pavel Kucherbaev, and Fabio Casati. Modeling, enacting, and integrating custom crowdsourcing processes. ACM Trans. Web, 9(2):7:1–7:43, May 2015.

[70] Fares Laroui. Definition and types of collaboration in business. https://www.exoplatform.com/blog/2020/11/11/definition-and-types-of-collaboration-in-business/, November 2020.

[71] William Johnson and Roberto Filippini. Internal vs. external collaboration: What works. Research-Technology Management, 52, 05 2009.

[72] Andrej Rus and Marko Orel. Coworking: A community of work. Teorija in Praksa, 52(6):1017–1038, December 2015.

[73] Clay Spinuzzi. Working alone, together: Coworking as emergent collaborative activity. Journal of Business and Technical Communication, pages 399–441, 2012.

[74] Jeff Howe. The rise of crowdsourcing. Wired Magazine, 14(6), 06 2006.

[75] Anand Pramod Kulkarni, Matthew Can, and Björn Hartmann. Collaboratively crowdsourcing workflows with turkomatic. In Steven E. Poltrock, Carla Simone, Jonathan Grudin, Gloria Mark, and John Riedl, editors, CSCW, pages 1003–1012. ACM, 2012.

[76] Daren C. Brabham. Using crowdsourcing in governement. IBM Center for the Business of Government, 2013.

[77] Katarzyna Kubiak and Anna Wziatek-Kubiak. Business models innovation based on crowds: a comparative study. International Journal of Management and Economics, 55(2):127–147, 2019.

[78] K. Benouaret, R. Valliyur-Ramalingam, and F. Charoy. Crowdsc: Building smart cities with large-scale citizen participation. IEEE Internet Computing, 17(6):57–63, Nov 2013.

[79] Tatiana De Feraudy and Mathieu Saujot. Une ville plus contributive et durable: crowdsourcing urbain et participation citoyenne numérique. Studies $N°$04/17, Iddri, Paris, France(04/17):72p, 2017.

[80] Barbara J. Grosz. Collaborative systems (aaai-94 presidential address). AI Magazine, 17(2):67, Mar. 1996.

[81] Jr Nsaibirni Robert Fondze and Gaetan Texier. User interactions in dynamic processes: Modeling user intractions in dynamic collaborative processes using active workspaces. In Proceedings of CARI 2016, pages 109–116, 2016.

[82] D. Harel and A. Pnueli. Logics and models of concurrent systems. chapter On the Development of Reactive Systems, pages 477–498. Springer-Verlag, Berlin, Heidelberg, 1985.

[83] Martín Abadi and Leslie Lamport. Composing specifications. ACM Trans. Program. Lang. Syst., 15(1):73–132, January 1993.

[84] Martin Abadi and Gordon D. Plotkin. A logical view of composition. THEORETICAL COMPUTER SCIENCE, 114:3–30, 1993.

[85] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering (FSE), ACM, pages 109–120. Press, 2001.

[86] Philip Merlin and Gregor V. Bochmann. On the construction of submodule specifications and communication protocols. ACM Trans. Program. Lang. Syst., 5(1):1–25, January 1983.

[87] Jean-Baptiste Raclet. Residual for component specifications. Electronic Notes in Theoretical Computer Science, 215:93 – 110, 2008. Proceedings of the 4th International Workshop on Formal Aspects of Component Software (FACS 2007).

[88] Glenn Shafer. A mathematical theory of evidence. Princeton University Press, 1976.

[89] Lofti Zadeh. Fuzzy sets as the basis for a teory of possibility. Fuzzy sets and Systems, 1:3–28, 1978.

[90] H. Zhu. Role mechanisms in collaborative systems. International Journal of Production Research, 44(1):181–193, 2006.

[91] Krzysztof Czarnecki and Ulrich W. Eisenecker. Generative programming - methods, tools and applications. Addison-Wesley, 2000.

[92] Oppl Stefan and Rothschädl Thomas. Separation of concerns in model elicitation – role-based actor-driven business process modeling. In Zehbold C. (eds) S-BPM ONE - Application Studies and Work in Progress. S-BPM ONE 2014. Communications in Computer and Information Science, volume 422. Springer, Cham, 2014.

[93] Oumaima Saidani and Selmin Nurcan. A role-based approach for modeling flexible business processes. In Workshop on Business Process Modelling, Development, and Support, pages 111 – 120, Luxembourg, 2006.

[94] Devis Bianchini, Cinzia Cappiello, Valeria De Antonellis, and Barbara Pernici. Semantic service design for collaborative business processes in internetworked enterprises. In Carlos Alberto Heuser and Günther Pernul, editors, Advances in Conceptual Modeling - Challenging Perspectives, pages 2–11, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[95] Devis Bianchini, Cinzia Cappiello, Valeria De Antonellis, and Barbara Pernici. P2s: A methodology to enable inter-organizational process design through web services. In Pascal van Eck, Jaap Gordijn, and Roel Wieringa, editors, Advanced Information Systems Engineering, pages 334–348, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[96] Mike Winters. Bpmn and microservices orchestration, part 1 of 2: Flow languages, engines, and timeless patterns. https://zeebe.io/blog/2018/08/bpmn-for-microservices-orchestration-a-primer-part-1/, August 2018.

[97] Bernd Rücker. The microservices workflow automation cheat sheet. https://blog.bernd-ruecker.com/the-microservice-workflow-automation-cheat-sheet-fc0a80dc25aa, November 2018.

[98] Object Management Group (OMG). Documents associated with unified modeling language(uml) version 2.5 - beta 2. http://www.omg.org/spec/UML/2.5/Beta2/.

[99] Haibin Zhu. A role agent model for collaborative systems. In in Proc. Int. Conf. Inf. and, pages 438–444, 2003.

[100] A. Hermann, Hendrik Scholta, S. Bräuer, and J. Becker. Collaborative business process management - a literature-based analysis of methods for supporting model understandability. Wirtschaftsinformatik und Angewandte Informatik, 2017.

APPENDIX A

HASKELL IMPLEMENTATIONS

A.1 Implementing a relation

```
module Relation where
import qualified Data.Set as Set
import qualified Data.List as List
import Util

type Relation a = [(a,a)]

--Knowing required and provided services of a relation
toSet::(Eq a,Ord a) => ((a,a) -> a) -> Relation a -> Set.Set a
toSet _ [] = Set.empty
toSet f (r:rs) = Set.insert (f r) (toSet f rs)

-- getting provided services of a relation
provided::(Eq a,Ord a) => Relation a -> Set.Set a
provided = toSet snd

-- getting required services of a relation
required::(Eq a,Ord a) => Relation a -> Set.Set a
```

```
required = toSet fst

-- Sequential composition of two R1 and R2
seqComp::Eq a => Relation a -> Relation a -> Relation a
seqComp [] _ = []
seqComp _ [] = []
seqComp (l:ls) rs = oneComp l rs ++ seqComp ls rs

--  Checking composition possibility between a relation instance (x,y)
-- and a relation R2
oneComp::Eq a => (a,a) -> Relation a -> Relation a
oneComp _ [] = []
oneComp (x,y) (r:rs)= if y == fst r then
               (x,snd r): oneComp (x,y) rs else oneComp (x,y) rs

-- restriction of a relation R to a subset of provided services  O in Out(R)
restriction::Eq a => [a] -> Relation a -> Relation a
restriction [] _ = []
restriction (l:ls) rs = toSelect snd l rs ++ restriction ls rs

--Checking relations providing a given provided service
toSelect::Eq a => ((a,a) -> a) -> a -> Relation a -> Relation a
toSelect _ _ [] = []
toSelect f l (r:rs) = if l == f r then
             r: toSelect f l rs else toSelect f l rs

--Reflexive-Transitive closure of a relation
refTransClosure::(Eq a,Ord a) => Relation a -> Relation a
refTransClosure  = refClosure.transClosure

-- reflexive closure
refClosure:: (Eq a,Ord a) => Relation a -> Relation a
```

```
refClosure [] = []
refClosure (x:xs) = List.nub ((x:xs)
      ++ [(y,y) | y <- Set.toAscList (required (x:xs))
      ++ Set.toAscList (provided (x:xs))])

-- transitive closure
transClosure::(Eq a) => Relation a -> Relation a
transClosure [] = []
transClosure closure
|closure == closureUntilNow = closure
|otherwise = transClosure closureUntilNow
where closureUntilNow =
    List.nub $ closure ++ [(a,c)|(a,b)<-closure, (b',c)<-closure, b == b']

-- cyclicity test
isCyclic::(Eq a,Ord a) => Relation a ->Bool
isCyclic [] = False
isCyclic r =
   hasCycle  (Set.toAscList (Set.union (required r) (provided r)))  r


-- is a quasi-Interface ?
isQuasi::(Eq a,Ord a) => Relation a -> Bool
isQuasi [] = False
isQuasi r = not(isCyclic (refTransClosure r))

-- relations union
union::(Eq a) => Relation a -> Relation a -> Relation a
union xs ys = List.nub (xs ++ ys)

-- checking dependencies within a relation
depend::(Eq a) => a -> Relation a -> [a]
```

```
depend _ [] = []
depend s (r:rs) =
   if s == snd r then fst r : depend s rs else depend s rs
```

A.2 Implementation of a role interface

```
module Interface where
import qualified Data.Set as Set
import Data.List()
import Relation

type Interface a = (Set.Set a,Relation a,Set.Set a)

-- is closed interface?
isClosed::Interface a -> Bool
isClosed (x,r,_) = null x && null r

-- is empty interface?
isEmpty::Interface a -> Bool
isEmpty (x,r,y) = null x && null r && null y

--get empty interface
empty::Interface a
empty = (Set.empty,[],Set.empty)

--check interfaces composability
areComposable::(Eq a,Ord a) => Interface a -> Interface a  -> Bool
areComposable (_, x1, o1) (_,x2,o2) =
    not(isCyclic (transClosure (x1 `union` x2)))
    && Set.null (Set.intersection o1 o2)

-- check possible collaboration
```

```
-- i.e check possibility of cascede composition between two interfaces
collaborate:: (Eq a,Ord a) => Interface a -> Interface a -> Bool
collaborate x1 x2 =
  areComposable i1 i2
    && not (Set.null (Set.intersection (output x2) (input x1)))

-- Union of two interfaces
unionI::(Eq a,Ord a) => Interface a -> Interface a -> Interface a
unionI x1 x2 = (x,y,z)
where
x = Set.union (input x1) (input x2)
y = relation x1 `union` relation x2
z = Set.union (output x1) (output x2)

-- getting required services of an interface
input:: Interface a -> Set.Set a
input (i,_,_) = i

-- getting provided services of an interface
output:: Interface a -> Set.Set a
output (_,_,o) = o

-- getting interface relation
relation:: Interface a -> Relation a
relation (_,r,_) = r

-- checking service dependencies whithin an interface
dependOn::(Eq a) => a -> Interface a -> [a]
dependOn s i = depend s (relation i)

sdependOn::(Eq a) => [a] -> Interface a -> [a]
sdependOn [] _ = []
```

```
sdependOn (x:xs) i =
    if null req then  sdependOn xs i else req ++ sdependOn xs i
         where  req = depend x (relation i)

dependOn0::(Eq a) => a -> Interface a -> ([a], Interface a)
dependOn0 s i = (depend s (relation i), i)

mdependOn::(Eq a) => [a] -> [Interface a] -> [a]
mdependOn [] _ = []
mdependOn _ [] = []
mdependOn (x:xs) crowd =
      concatMap (dependOn x) crowd ++ mdependOn xs crowd

mdependOn0::(Eq a) => [a] -> [Interface a] -> [([a],Interface a)]
mdependOn0 [] _ = []
mdependOn0 _ [] = []
mdependOn0 (x:xs) crowd =
    filter (\(f,_)-> f/=[]) (map (dependOn0 x) crowd) ++ mdependOn0 xs crowd
```

A.3 Implementing a collaboration

```
module Collaboration where
import Interface
import qualified Data.Set as Set
import Data.List(intersect)

type Collaboration a = ([a],Interface a,Interface a)

label::Collaboration a -> [a]
label (xs ,_,_) = xs

requester:: Collaboration a -> Interface a
```

```
requester (_ ,_,r) = r

provider:: Collaboration a -> Interface a
provider (_,r,_) = r

whoSupply::(Ord a) => a -> [Interface a] -> [Interface a]
whoSupply s =  seekService s output

whoNeeds::(Ord a) =>a -> [Interface a] -> [Interface a]
whoNeeds s =  seekService s input

--
seekService::(Ord a) => a -> (Interface a -> Set.Set a)
                            -> [Interface a] -> [Interface a]
seekService _ _ [] = []
seekService s h (i:is) =
  if Set.member s (h i) then  i : seekService s h is
  else seekService s h is


rGDPi::(Eq a, Ord a) => Interface a -> [Interface a] -> [Collaboration a]
rGDPi _ [] = []
rGDPi r (i:is) =
     if not (collaborate r i) then rGDPi r is
         else (x,y,z) : rGDPi r is
         where
             x = Set.toAscList (Set.intersection (input r) (output i))
         y = i
          z = r

gDPi::(Eq a, Ord a) => [Interface a] -> [Interface a] -> [[Collaboration a]]
gDPi [] _ = []
```

```
gDPi (i:is) prec = rGDPi i (xs++is) : gDPi is xs  where
                            xs = i:prec



sGDPi::(Eq a, Ord a) => a -> [Interface a] -> [Collaboration a]
sGDPi _ [] = []
sGDPi s crew = hx [s] crew (concat (gDPi crew []))



decidable::(Eq a, Ord a) => [a] -> [Collaboration a] ->(Bool,[a])
decidable [] _ = (True, [])
decidable xs [] = (False, xs)
decidable xs (c:cs)
    | null (xs `intersect` label c) = decidable (xs++req) cs
        | not (null (xs `intersect` label c)) = decidable (xxs ++ req) cs
     where
       xxs = [x | x <- xs, x `notElem` label c]
       req = sdependOn (label c) (provider c)
```

A.4 Implementing an $\mathcal{F}$-collaboration

```
module TCollaboration where
import Interface
import Collaboration
import qualified Data.Set()
import Data.List()

data TCollaboration a = NoCol
                    | Col [a] (Interface a) (Interface a)
                        | OrCol [a] [Interface a] (Interface a)
                        deriving Show
```

```
groupCol::(Eq a, Ord a) => [Collaboration a] -> [[Collaboration a]]
groupCol [] = []
groupCol (c:cs) =
   fst (subCol (label c) (c:cs)) : groupCol (snd (subCol (label c) (c:cs)))

subCol:: (Eq a, Ord a) => [a] -> [Collaboration a]
                           -> ([Collaboration a],[Collaboration a])
subCol xs cs = (x,c) where
                x = filter(\(l,_,_)-> l == xs) cs
               c = filter(\(l,_,_)-> l /= xs) cs

ortransform::(Eq a, Ord a) =>[Collaboration a] -> [TCollaboration a]
ortransform cs = ortrans (groupCol cs)

ortrans::[[Collaboration a]] -> [TCollaboration a]
ortrans = map orT


orT::[Collaboration a] -> TCollaboration a
orT [] = NoCol
orT [c] =  Col (label c) (provider c) (requester c)
orT (c:cs) = OrCol (label c) (map provider (c:cs)) (requester c)
```

A.5 Implementing activity

```
module Activity where
import Interface
import TCollaboration
import Collaboration

type Activity a = (a,[Collaboration a])
```

```
process::Activity a -> [Collaboration a]
process = snd

svce::Activity a -> a
svce = fst

decomposable::TCollaboration a -> Bool
decomposable NoCol = False
decomposable Col {} = False
decomposable OrCol{} = True

isDecomposable::(Eq a, Ord a) => Activity a -> Bool
isDecomposable act =
      foldr ((||).decomposable) False (ortransform (process act))


decomp::Eq a =>[TCollaboration a] -> [[TCollaboration a]]
                                        -> [[TCollaboration a]]
decomp [] prefix = prefix
decomp (NoCol:cs) prefix = decomp cs prefix
decomp (OrCol s p r :cs) prefix = decomp cs (mdup (s,p,r) prefix)
decomp (Col s r0 r1:cs) prefix = decomp cs (inser (Col s r0 r1) prefix)


inser::Eq a => TCollaboration a -> [[TCollaboration a]]
                                         -> [[TCollaboration a]]
inser _ [] = []
inser (Col s p r) (xs:xss) =
        pref : inser (Col s p r) xss where
               pref = if isSupplied r xs then  xs ++ [Col s p r] else xs
```

```
isSupplied::Eq a => Interface a -> [TCollaboration a] -> Bool
isSupplied _ [] = False
isSupplied r (Col _ _ r0:cs) = r == r0 || isSupplied r cs


dup::([a],[Interface a],Interface a) -> [TCollaboration a]
                                        -> [[TCollaboration a]]
dup (_,[],_) _= []
dup (_,_,_) [] = []
dup (s,p:ps,r) cs = (cs ++ [Col s p r]) : dup (s,ps,r) cs

mdup::([a],[Interface a],Interface a) -> [[TCollaboration a]]
                                        -> [[TCollaboration a]]
mdup (_,[],_) _= []
mdup (s,p:ps,r) [] = [Col s p r] : mdup (s,ps,r) []
mdup (s,ps,r) (cs:css) = dup (s,ps,r) cs ++ mdup (s,ps,r) css

dec::(Eq a, Ord a) => Activity a -> (Bool,[a])
dec(_,[]) = (True,[])
dec(s0,cs) = decidable [s0] cs
```