A role-based collaborative process design on crowdsourcing systems

Rodrigue Aimé Djeumen Djatcha

▶ **To cite this version:**

Rodrigue Aimé Djeumen Djatcha. A role-based collaborative process design on crowdsourcing systems. Africain Conference on Research in Computer Science and Applied Mathematics, Polytech School of Thiès, Oct 2020, Thiès, Senegal. hal-02904553v2

HAL Id: hal-02904553
https://hal.science/hal-02904553v2

Submitted on 2 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - ShareAlike 4.0 International License

A role-based collaborative process design

on crowdsourcing systems

Rodrigue Aimé Djeumen Djatcha

Faculty of Sciences, University of Douala - LIRIMA-FUCHSIA team
Email:djeumenr@yahoo.fr

ABSTRACT. 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 crowsourcing 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 mechanisms 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.

RÉSUMÉ. Le crowdsourcing est un modèle de processus métier collaboratif, dans lequel les tâches sont externalisées. Dans ces systèmes, on distingue deux types d'intervenants: les demandeurs qui externalisent des tâches et la foule ou contributeurs qui effectuent ces tâches. Nous considérons un intervenant à la fois comme un acteur et un composant logiciel autonome, évoluant sur une plateforme, disposant de mécanismes d'interaction avec son environnement et de compétences métiers. Un ensemble d'intervenants interagissant dans un contexte dynamique pour résoudre un problème, est un système collaboratif distribué,désigné par système de crowdsourcing. Dans un tel système, le concept de rôle est central, car chaque intervenant doit disposer d'un cadre spécifique dans lequel il collabore. Traditionnellement, les systèmes collaboratifs perdent en flexibilité si leur conception est rôle-centrée, car seuls les mécanismes statiques de description de rôles, basés sur des concepts intuitifs sont disponibles. Nous proposons dans cet article, une amélioration consistant en quatre choses: (1)définir clairement ce qu'est une tâche externalisable, (2)spécifier les rôles clairement et rigoureusement, tout en assurant la flexibilité de la collaboration, (3) fournir des mécanismes de changement de rôle et, (4) fournir une base abstraite, pour la conception de systèmes de crowdsourcing et les mécanismes de monitoring et de vérification de workflows, pour les activités menées dynamiquement par un système.

KEYWORDS : Role-based approach, Dynamic workflow, Separation of concerns, SOD, Interface of role, Crowdsourcing Systems, Distributed collaborative systems, Guarded attribute grammar.

MOTS-CLÉS : Approche rôle-centrée, workflows dynamiques, séparation de préoccupations, SOD, interface de rôle, systèmes de crowdsourcing, systèmes collaboratifs distribués, GAG.

1. Introduction

Crowdsourcing concept, introduced by Jeff Howe [4], is a collaborative business process model, in which tasks are carried out by a crowd. Simply, an organization can request an online community for a voluntary accomplishment of a task, according to what both parties have a mutual benefit. In a crowdsourcing system, organizational objectives are top-down, while creative activity is bottom-up organized; we therefore say crowdsourcing is a mixed mode organized process [3]. Crowdsourcing processes are based on three pillars, namely: requester, crowd and crowd task [5]. ***Requester*** is a natural or legal person, who requests the power and wisdom of the crowd, for the performance of a given service. 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 confidentiality on contributors data. ***Crowd*** is a community of contributors, taking part in a crowdsourcing activity, whose characteristics are: diversity (variety of skills,spatiality,...), anonymity, importance, completeness or adequacy. ***Crowd task*** is an activity in which the crowd participates. It can be a large-scale data collection, a co-creation task, or an innovation task. Characteristics of a crowd task are: modularity, complexity, solvability, automaticity, contributor-centric. Crowd skills orchestration, consists of distributing tasks to contributors, aggregating results and continuing the process based on the partial results obtained. According to task complexity, acceptance of the result and gratification granted criteria, there are mainly 3 strategies for orchestrating crowd services [6, 7]: *market* or strategy for large scale micro-tasking; *contest* or strategy for co-creativity, *auction* or strategy for innovation.

A crowdsourcing system is an environment, where a crowdsourcing process is deployed; it integrates practices of business process modeling, inspired by service-oriented architecture (SOA) moreover, service calls may not be hierarchically organized; it establishes connection between crowd, individual actors and machines. It allows services composition by assembling a suite of small specialized independent services, offering respectively modular decomposition basis for services, an architecture organized around business skills, focusing on product but not the project. Communication here is either Restful, SOAP like or by messages through channels; governance and data management are decentralized. Defining properly a stakeholder requires specifying two things: (i) its *functional goal* or *business skills*, i.e interface (services he requires and services he provides) and business rules (functionalities targeted by the stakeholder); (ii) its *infrastructure mechanisms* or *intrinsic skills* i.e storage for persistence, communication mechanisms, sensing tools, etc [15]. In this work our interest is crowdsourcing processes design for large scale micro-tasking. We consider a crowdsourcing system, as a set of independent stakeholders providing precise services. Stakeholder infrastructure is thus perceived as an *actor*, while its functional goal is a *role*. In fact, in such a system, each actor can be assigned several roles and several actors could play one instance of role. Actors may not have same intrinsic skills, and several occurrences of a role could exist at the same time in the system. Considering all role in a given context of a system, we get a formal reasoning basis for business goals of the whole system. As crowdsourcing system use case, consider a city participatory management case [16], with processes depict on figure 2, 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.

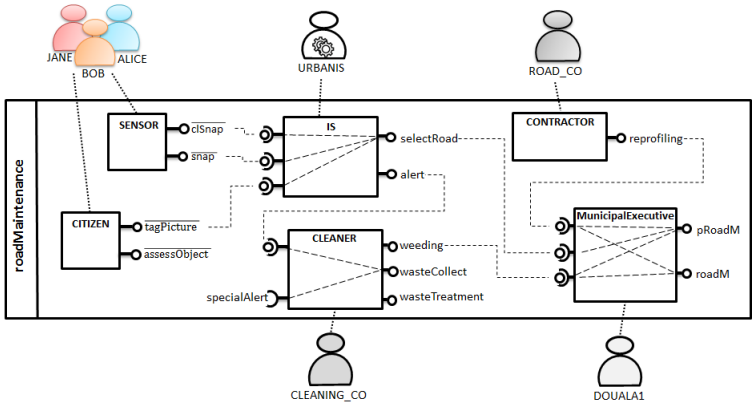


Figure 1 – A crowdsourced road maintenance activity

Role concept is central in any collaborative system, as each actor must have a clear framework within which he collaborate. A role specifies both what the system expects from actors, but also what actors expects from the system; thus avoiding that an actor be overwhelmed by information (or tasks) not necessary. Traditionally, collaborative systems lose flexibility as soon as their design is role-based, because only static role description mechanisms, based on intuitive concepts, are available [2], and certainly, a dynamic context of collaboration, further complicates a design of such systems, since entities involved, evolve over time in number and in skills. An improvement could be to provide four things: (1)define clearly what an outsourceable task or crowd task is, (2)specify roles clearly and rigorously, while ensuring flexibility for collaboration, (3)provide role switching mechanisms and, (4) provide an abstract basis, for crowdsourcing system design and workflow monitoring and checking mechanisms. This design approach implies a rigorous definition in advance, of the set of roles and the relationships between them, necessary to describe the functioning of a target domain. Role being a particular concern in the system. In fact, a role-based design approach is similar to a separation of concern technique [1, 11], applied to business process design. It is implemented by specifying activities defining process's steps, as well as flows describing coordination of these activities, as it may be done with BPMN orchestration or UML collaboration or activities diagrams.

2. Role & role interface modeling and basic mechanisms

Common definitions and terminologies - a *role type* is the perception one actor has of another actor [13]. A role type is specified uniquely, and an *actor* 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 an actor. It may not always be easy to differentiate between a role and an actor. For instance, consider the MAYOR and CSOFFICER entities, as parts

of collaboration within a process of civil status certificate issuing at appendix A1. Being able to distinguish between these entities, which is role type (or simply role) and which is actor, can be rather complex.The sharp distinction between an actor and a role type is based on both concepts of foundation and semantic rigidity [14]. An entity is considered to be *founded*, if its specification implies a dependency or relationship with another entity. Rather it 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 an actor), whereas CSOFFICER is founded and semantically non-rigid (CSOFFICER is a role); in other words, a MAYOR plays the CSOFFICER role. Hence an actor is unfounded and semantically rigid, while a role is founded and semantically non-rigid.

Role and crowdsourcing grammatical tasking model - In this work we will consider, role type and role as equivalents, simply called role. A role specifies the set of actions a given actor can take in the system.

Definition 2.1 *A role r is given by a couple $(\mathcal{G}, R)$ where $\mathcal{G}$ is a guarded attribute grammar (GAG)*[1] *[10] specifying role r business skills, and R it's associated interface of role.*

Business skills are expressed as grammar production rules (or business rules), describing job decomposition, in the form

$$s_0 \rightarrow \boldsymbol{T_0 \cdots T_m}\ s_1 \cdots s_n\ \overline{s}_0 \cdots \overline{s}_p \tag{1}$$

where s_0 is a *defined service*, $s_1 \cdots s_n$ are *used services* and $\boldsymbol{T_0 \cdots T_m}$ are actor's *intrinsic skills*, finally $\overline{s}_0 \cdots \overline{s}_p$ are *crowd tasks*. According to this rule, 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 actor playing that role. Within the 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***. *Crowd tasks* are defined by business rules like $\overline{s}_i \rightarrow \boldsymbol{T_0 \cdots T_m}$ $(0 \leq i \leq p)$ i.e they are type of services requiring only actors intrinsic skills, to be carried out and so, they 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, term *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. For instance, if we reconsider the crowdsourcing system described on figure 2, 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).

$$\begin{array}{rcl} \overline{clSnap} & \rightarrow & \boldsymbol{CONTEXT\ LOCATION\ SNAP} \\ \overline{snap} & \rightarrow & \boldsymbol{SNAP} \end{array} \tag{2}$$

1. As the goal is to describe processes, we are not yet interested with attributes in guarded attribute grammars.

A CITIZEN according to business skills in rule (3), can tag a picture and assess object, in both cases by inputting some data.

$$\begin{array}{rcl} \overline{tagPicture} & \rightarrow & \boldsymbol{INPUT} \\ \overline{assessObject} & \rightarrow & \boldsymbol{INPUT} \end{array} \tag{3}$$

Lastly according to rule (4), 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).

$$\begin{array}{rcl} selectRoad & \rightarrow & \overline{clSnap}\ \ \overline{snap}\ \ \overline{tagPicture} \\ alert & \rightarrow & \varepsilon \end{array} \tag{4}$$

Interface of role - an *interface of role* [8, 9] is some abstraction of GAG $\mathcal{G}$ associated to a role, specifying what services are provided, which external services are required to carry them out and an over-approximation of dependencies between required and provided services termed *potential dependencies*. Interface disregards internal tasks. See [9] for interfaces derivation from a GAG.

Definition 2.2 *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 disjoint sub-sets ${}^\bullet R$ and $R^\bullet$ from Ω, such that ${}^\bullet R = R^{-1}(\Omega) = \{A \in \Omega \mid \exists B \in \Omega\ (A, B) \in R\}$ and $R^\bullet \supseteq R(\Omega) = \{B \in \Omega \mid A \in \Omega\ (A, B) \in R\}$.*

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. 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.

$$\begin{array}{l} {}^\bullet IS = \left\{\overline{clSnap}, \overline{snap}, \overline{tagPicture}\right\} \\ IS = \left\{(\overline{clSnap}, selectRoad), (\overline{snap}, selectRoad), (\overline{tagPicture}, selectRoad), (_, alert)\right\} \\ IS^\bullet = \{selectRoad, alert\} \end{array} \tag{5}$$

An interface provide several basic operations [8, 9] as:

Restriction ($\restriction$): Let $O \subseteq \Omega$ a subset of services. A *restriction* of interface R to O denoted $R \restriction O$ is given by $R \restriction O = \{(A, B) \in R \,|\, B \in O\}$ With $(R \restriction O)^\bullet = O \cap R^\bullet$ and ${}^\bullet(R \restriction O) = R^{-1}(O \cap R^\bullet)$ respectively. Considering 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; useful when just some skills of a multi-skilled role are needed.

Cascade composition test ($\rtimes$): Let two roles R_1 and R_2; cascade composition of those roles holds iff $R_1^\bullet \cap {}^\bullet R_2 = \emptyset \wedge {}^\bullet R_1 \cap R_2^\bullet \neq \emptyset$. We denote $R_1 \rtimes R_2$ their *cascade product test* (or $R_2 \ltimes R_1$ since this operation is commutative). When cascade composition holds, the resulting role required services set is $({}^\bullet R_1 \setminus R_2^\bullet) \cup {}^\bullet R_2$, and provided services set is $R_1^\bullet \cup R_2^\bullet$.

Direct Product($\times$) Let R_1 and R_2 two composable role interfaces. If $R_1^\bullet \cap {}^\bullet R_2 = \emptyset$ and $R_2^\bullet \cap {}^\bullet R_1 = \emptyset$, 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 $(R_1 \times R_2)^\bullet = R_1^\bullet \cup R_2^\bullet$.

3. Collaborations and collaboration schemes

In a targeted domain, a *context of collaboration* is the set of roles available instantly for the realization of some activities. A context of collaboration is dynamic, i.e roles involved vary over time. For instance, we will consider the context on Figure 3 Appendix A4, as a running collaborative context, for the next parts of this work. We talk of collaboration between two roles, when there is a service dependency between them. This dependency can be direct, in which case it is a *direct collaboration*; likewise, it can be indirect being an *indirect collaboration*.

Definition 3.1 *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.*

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 (Appendix A2), determines potential direct collaborations (or *potential dependencies*) of r_0, in a context $\mathbf{R}$, in which r_0 is member and such that $\forall r_i \in \mathbf{R},\ r_i = (\mathcal{G}_i, R_i)$. Applying algorithm 1 on context (figure 3), will result to graph $\mathcal{C} = \{(\{x\}, r_0, r_3)\,, (\{x\}, r_4, r_3)\,, (\{t\}, r_2, r_3)\,, (\{w\}, r_1, r_3)\}$.

Collaboration schemes - from any context $\mathbf{R}$, a collaboration scheme or induced potential dependencies graph 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 $\mathbf{R}$, may leads to collaboration scheme $\mathcal{C} = \{(\{a\}, r_5, r_0)\,,\ (\{m\}, r_6, r_4)\,, (\{y\}, r_0, r_2)\,,\ (\{z\}, r_1, r_2)\,,\ (\{w\}, r_1, r_3)\,,\ (\{x\}, r_4, r_3)\,,\ (\{x\}, r_0, r_3)\,, (\{t\}, r_2, r_3)\}$.

Potential workflow of a service - the induced workflow of a service s_0, describes how this service will be issued; it 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 associated required services, and returns a list of couples made up of a required services set L and the role r requesting these services. We also let $iPDG(\mathbf{R}, \emptyset) \upharpoonright L_k = \{(l_0, r_i, r_j) \mid \forall (L_k, r_k) \in dependOn(s_0, \mathbf{R}),\ \ l_0 \in L_k\, and\, r_j = r_k\}$ 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 $(l_0 \in L_k\, et\, r_j = r_k)$. Algorithm 3 (Appendix A2), generates the potential workflow of a given service s_0, from

a context $\mathbf{R}$, having an associated induced potential dependencies graph ($\boldsymbol{iPDG}(\boldsymbol{R}, \emptyset)$). For instance, the potential workflow of the service u, obtained from that algorithm in the context of previous figure 3, is given by $workflow(\{u\}, \mathbf{R}) = \{ (x, r_4, r_3), (x, r_0, r_3), (t, r_2, r_3), (m, r_6, r_4), (a, r_5, r_0), (y, r_0, r_2), (z, r_1, r_2)\}$

Factorizing a workflow - a collaboration scheme may include several alternatives in supplying the same service, as the one of figure 4(a)-Appendix A3, where service x is provided by roles r_0 and r_4 respectively. 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(b)-Appendix A3.

Definition 3.2 *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. 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 schemes.

4. Activity in collaborative context

Definition 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 $workflow(\{s_0\}, \mathbf{R})$.*

Let us consider the process of delivering service u, given by previous $workflow(\{u\}, \mathbf{R})$. 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 4.1 *Two activities $activity_{s_0}$ and $activity_{s_1}$ are equivalent, and we denote by $activity_{s_0} \equiv activity_{s_1}$, iff $s_0 = s_1$ and $workflow(\{s_0\}, \mathbf{R}) \equiv workflow(\{s_1\}, \mathbf{R})$ i.e they deliver the same service in context $\mathbf{R}$, with $s_0, s_1 \in \Omega$.*

Proof 1 *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 $R = \times_{i=1}^{m} R_i$ and $R' = \times_{i=1}^{n} R_i$ 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.*

Atomicity of activities - an activity for a given service s_0, is said to be atomic [1], if it has only one occurrence of role supplier for each required service in that activity. For example, the atomic forms of previous activity $activity_u$ are respectively: $activity_{u_0} = (u, [(x, r_0, r_3), (t, r_2, r_3), (a, r_5, r_0), (y, r_0, r_2), (z, r_1, r_2)])$, $activity_{u_1} = (u, [(x, r_4, r_3), (t, r_2, r_3), (m, r_6, r_4), (y, r_0, r_2), (z, r_1, r_2)])$

Definition 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* $workflow(s_0, \mathbf{R})$, $r_i = r_k$.

Activity decomposition - An activity can be progressively fragmented into a set of atomic activities. The principle of decomposition, is based on roles (concern), 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 activities obtained are atomic [1]. For this, associated workflow must be factorized; if at the end of this, $\mathcal{F} - collaborations$ exist, then activity is decomposable, according to principles of algorithm 5 (Appendix A2), and figure 5 (Appendix A4).

Proposition 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* $activity'_{s_0}$ *if and only if* $workflow_0(s_0, \mathbf{R}) \neq workflow_1(s_0, \mathbf{R})$ *and* $factorize(\ workflow_0(s_0, \mathbf{R})) = workflow_1(s_0, \mathbf{R})$

Proof 2 *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 3.2.*

Activity Realizability - refers to the possibility of carrying out an activity, in a finite number of stages, and rendering provided service. This assumes that all necessary roles are available instantly; we say it is a *favorable context*. Termination of an activity, is conditioned by the fact that its workflow must contain autonomous roles as triggers. Algorithm 6 (Appendix A2), describes feasibility principles for a workflow, by applying a pattern matching 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.

REMARK. — An activity $activity_{s_0} = (s_0, workflow(s_0, \mathbf{R}))$ is said to be *quasi 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 $C \in \mathbf{decomp}(workflow(s_0, \mathbf{R}), \emptyset)$, $\mathbf{realisable}(\{s_0\}, C) = (True, \emptyset)$.

5. Actor of a distributed collaborative system

Definition 5.1 *An actor* a_τ *is given by a couple* $(\mathbf{R}_\tau, \mathcal{C}_\tau)$, *where* $\mathbf{R}_\tau$ *is the set of potential roles that actor can play, and* $\mathcal{C}_\tau$ *is the set of constraints on those potential roles.*

Let $a_\tau = (\mathbf{R}_\tau, \mathcal{C}_\tau)$ an actor, r_i and r_j two roles; association between actor a_τ and roles r_i and r_j is materialized by $r_i, r_j \in \mathbf{R}_\tau$. We will say for instance that r_i,r_j are actor's a_τ *potential roles*. Four constraint values can be defined on actor's potential roles [13], namely: $\boldsymbol{Dcr}$ 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; $\boldsymbol{Phb\,(r_i, r_j)}$ and so, actor a_τ playing role r_i, cannot play role r_j.

An actor can play one or more roles, within an activity, or in several parallel activities; then three cases of "play" relationship can be distinguished: **Playing several roles in an**

activity an actor can play more than one role within an activity; provided that those roles do not provide same services. An actor 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 6(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 $activity_{s_0}^{'} = (s_0, [\cdots, (B, r_0^{'}, r_2), (C, r_2, r_0^{'}), \cdots])$, as illustrated in the figure 6(b). **Competing activities** an actor can contribute in several activities at the same time, either by playing the same role each time (see figure 6 (c)), or by having different roles. In all these cases, each of these activities is implemented individually as in case 1 above. **Crowd role played by several actors** within an activity, several instances of actors with same intrinsic skills, can play the same occurrence of a role; in a workflow, if role r is played by several actors instances (figure 6 (d)), then any collaboration with r will be on a crowd task, and r is a crowd role. A pattern matching implementation of play relation between an actor and his potential roles, is given by equation 9 on Appendix A3.

6. Conclusion

This work focused on a role-based design approach, in a micro-tasking crowdsourcing system context i.e those dynamic system where every actor can be assigned several roles, can come in and go out of the system as he pleases and, several instances of actors can play the same role occurrence at a given moment, in the system. In our approach, we made a distinction between actor i.e infrastructure of a stakeholder and role i.e his business skills. Considering all skills in the system at a moment, procure an abstract basis for reasoning about business goals of the whole system. We defined a rigorous specification method for roles some basic operations, namely collaboration and roles reconfiguration. We also provide decomposition mechanisms and workflow monitoring and checking tools, role switching and workflow simplification mechanisms, the play relationships between an actor and his assigned roles, and the constraints between roles. This work is a prelude to a role-based design approach, for distributed collaborative systems. An immediate continuation is to reconsider a more complex system, as co-creative and innovative crowdsourcing systems, i.e a context where there are no pre-established rules, clarify conception of those type of processes, and describe interactions between actors.

7. References

[1] ARTUR CAETANO , ANTONIO RITO SILVA, JOSÉ TRIBOLET, "Business Process Decomposition - An Approach Based on the Principle of Separation of Concerns", *Enterprise Modelling and Information Systems Architectures*, vol. 5,num. 1,2010.

[2] H. ZHU, "Role mechanisms in collaborative systems", *International Journal of Production Research, Taylor & Francis*, vol. 44,2006.

[3] DAREN C. BRABHAM, "Using Crowdsourcing In governement", *IBM Center for the Business of Government*,2013.

[4] JEFF HOWE, "The Rise of Crowdsourcing", *Wired Magazine*, vol. 14,num. 6,2006.

[5] MAHMOOD HOSSEINI , KEITH PHALP , JACQUI TAYLOR, ALI RAIAN, "The four pillars of crowdsourcing: A reference model.", *Bajec, Marko and Collard, Martine and Deneckère, Rébecca. RCIS-IEEE*,2014.

[6] STEFANO TRANQUILLINI, FLORIAN DANIEL, PAVEL KUCHERBAEV, FABIO CASATI, “Modeling, Enacting, and Integrating Custom Crowdsourcing Processes”, *ACM Trans. Web*, vol. 9,num. 2,2015.

[7] PAVEL KUCHERBAEV, FLORIAN DANIEL, STEFANO TRANQUILLINI, MAURIZIO MARCHESE, “Crowdsourcing Processes: A Survey of Approaches and Opportunities.”, *IEEE Internet Computing*, vol. 20,num. 2,2016.

[8] ERIC BADOUEL, RODRIGUE AIMÉ DJEUMEN DJATCHA, “Interfaces of Roles in Distributed Collaborative Systems”, *CARI 2018 - Colloque Africain sur la Recherche en Informatique et Mathématiques Appliquées*,2018.

[9] ERIC BADOUEL, RODRIGUE AIMÉ DJEUMEN DJATCHA, “A Calculus of Interfaces for Guarded Attribute Grammars”, *https //hal.inria.fr/hal-02145920/file/Interfaces.pdf:PDF*,2019.

[10] ERIC BADOUEL, LOÏC HÉLOUËT, GEORGES-EDOUARD KOUAMOU, CHRISTOPHE MORVAN, ROBERT NSAIBIRNI FONDZE JR, “Active Workspaces: Distributed Collaborative Systems Based on Guarded Attribute Grammars”, *ACM SIGAPP Applied Computing Review*, vol. 15,num. 2,2015.

[11] KRZYSZTOF CZARNECKI, ULRICH W. EISENECKER, “Generative programming - methods, tools and applications”, *Addison-Wesley*,2000.

[12] STEPHAN BÖGEL, STEFAN STIEGLITZ, CHRISTIAN MESK, “A role model-based approach for modeling collaborative processes”, *Business Process Management Journal*, vol. 20,num. 4,2014.

[13] DIRK RIEHLE, THOMAS GROSS, “Role Model Based Framework Design and Integration”, *SIGPLAN Not.*, vol. 33, num. 10,1998.

[14] NICOLA GUARINO, “Concepts, attributes and arbitrary relations: Some linguistic and ontological criteria for structuring knowledge bases”, *Data & Knowledge Engineering*, vol. 8,num. 3,1992.

[15] EDWARDS, W. KEITH, “Policies and Roles in Collaborative Applications”, *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work. CSCW ’96, New York, NY, USA*,1996.

[16] K. BENOUARET, R. VALLIYUR-RAMALINGAM, F. CHAROY, “CrowdSC: Building Smart Cities with Large-Scale Citizen Participation”, *IEEE Internet Computing*, vol. 17,num. 6,2013.

Appendix

A1. E-administration collaboration use case

Consider an E-administration application for issuing civil status certificates, described on figure 2. This scheme shows collaborations between roles DECLARER, SECRETARY, CSOFFICER and JUDICIARY_AUTHORITY; roles played respectively by actors BOB, AGENT, (MAYOR, DIPLOMAT) and PROSECUTOR as stakeholders in an issuance civil status certificate (birth, marriage, death) activity[2].

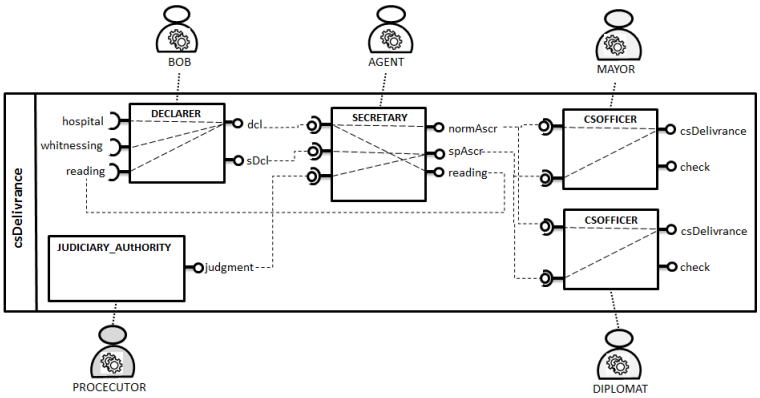


Figure 2 – Microservice system civil status certificate issuance

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 business rules in (6).

$$\begin{array}{rcl} dcl & \rightarrow & \boldsymbol{INPUT}\ withnessing \\ & | & \boldsymbol{INPUT}\ hospital \\ & | & \boldsymbol{INPUT}\ reading \\ sDcl & \rightarrow & \boldsymbol{INPUT} \end{array} \tag{6}$$

A SECRETARY according to business skills in rule (7), is responsible of declarations correction and transcription in related registers, whether these declarations are normal ($normAscr$) or special ($spAscr$).

$$\begin{array}{rcl} normAscr & \rightarrow & \boldsymbol{TRANSCRIPTION}\ dcl \\ spAscr & \rightarrow & \boldsymbol{TRANSCRIPTION}\ sDcl\ judgment \\ reading & \rightarrow & \boldsymbol{CORRECT}\ dcl \end{array} \tag{7}$$

Lastly according to rule (8), a CSOFFICER is responsible of issuing civil status certificates and certain checks. Checks rules are not yet expressed and may be done later.

2. According to Cameroonian laws

$$\begin{array}{rcl} csDelivrance & \rightarrow & \boldsymbol{SIGNUP}\ normAscr \\ & | & \boldsymbol{SIGNUP}\ spAscr \\ check & \rightarrow & \varepsilon \end{array} \tag{8}$$

A2. Related algorithms

Algorithm 1: Role potential dependencies graph (rPDG) calculus

input: $r_0 = (\mathcal{G}_0, R_0)$, $\mathbf{R}$
output: $\mathcal{C}$ *//set of potential collaborations of role* r_0.
  $\mathcal{C} \leftarrow \emptyset$
  rPDG$(r_0, \mathbf{R}) =$
    forall r_i *in* $\mathbf{R}$
      if $R_0 \bowtie R_i$ *then* $\mathcal{C} \cup \{({}^{\bullet}R_0 \cap R_i^{\bullet}, r_i, r_0)\}$

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

input: $\mathbf{R}$
    $\mathbf{R}'$ *//roles whose rPDG have already been determined, initially empty.*
output: $\mathcal{C}$
  $\mathbf{R}' \leftarrow \emptyset$
  iPDG$(\mathbf{R}, \mathbf{R}') =$
    if $(r_i$ *in* $\mathbf{R})$ *and* $(\mathbf{R} \neq \emptyset)$ *then*
      rPDG$(r_i, \mathbf{R} \cup \{\mathbf{R}' \cup \{r_i\}\}) \cup$ ***iPDG***$(\mathbf{R} \setminus \{r_i\}, \mathbf{R}')$
    else ***iPDG***$(\mathbf{R} \setminus \{r_i\}, \mathbf{R}')$

Algorithm 3: Determining a potential workflow for a service

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 4: Factorizing a service

input: $\mathcal{C}$
output: $\mathcal{C}'$
$\mathcal{C}' \leftarrow \emptyset$
factorize$(\mathcal{C}) =$
forall c in $P(\mathcal{C})$
if $|c|$ ==1 then *//c is like* $\{({}^{\bullet}R_0 \cap R_1^{\bullet}, r_1, r_0)\}$
$\mathcal{C}' \cup c$
else if $|c| > 1$ then *//c is like* $\{({}^{\bullet}R_0 \cap R_1^{\bullet}, r_1, r_0), \cdots, ({}^{\bullet}R_0 \cap R_{|c|}^{\bullet}, r_{|c|}, r_0)\}$
$\mathcal{C}' \cup \left\{({}^{\bullet}R_0 \cap R_1^{\bullet}, \{r_1, \cdots, r_{|c|}\}, r_0) \mid r_{1 \leq i \leq |c|} \in provider\left({}^{\bullet}R_1 \cap R_{|c|}^{\bullet}, c\right) \; and \; r_0 == requester(c)\right\}$

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.

Algorithm 5: Atomic decomposition of an activity's workflow

input: $\mathcal{C}'$ //factorized workflow of the activity .
output: $\mathcal{C}$ //set of potential atomic workflows.
$\mathcal{C} \leftarrow \emptyset$
decomp$(\mathcal{C}', \mathcal{C}) =$
forall c in $\mathcal{C}'$
if $|provider(c)| == 1$ then //c is like $\{({}^{\bullet}R_0 \cap R_1^{\bullet}, \{r_1\}, r_0)\}$
decomp$(\mathcal{C}' \setminus \{c\}, \mathbf{insert}\,(c, \mathcal{C}))\}$
else if $|provider(c)| > 1$ then //c is like $({}^{\bullet}R_0 \cap R_1^{\bullet}, \{r_1, \cdots, r_{|c|}\}, r_0)$
decomp$(\mathcal{C}' \setminus \{c\}, \mathbf{mdup}\,(c, \mathcal{C}))\}$

$\mathbf{insert}\,(c, \mathcal{C})$, insert collaboration c, in the different atomic workflows of $\mathcal{C}$, for which c is necessary.
$\mathbf{mdup}\,(\{({}^{\bullet}R_0 \cap R_1^{\bullet}, \{r_1, \cdots, r_{|c|}\}, r_0)\}, \mathcal{C})$ 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.

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

input: $Serv$ //set of required services
$\mathcal{C}$ //activity workflow
output: $(Bool, Serv)$

realizable$(\emptyset, _) = (True, \emptyset)$ //the workflow is realizable
realizable$(Serv, \emptyset) = (False, Serv)$ // not realizable, $Serv$ are required services
realizable$(Serv, c \in \mathcal{C})$
$\mid Serv \cap label(c) == \emptyset$ =***realizable***$(Serv \cup req, \mathcal{C} \setminus \{c\})$
$\mid Serv \cap label(c) \neq \emptyset$ =***realizable***$(Serv' \cup req, \mathcal{C} \setminus \{c\})$
where
$Serv' = Serv \upharpoonright (x \in Serv \wedge x \notin label(c))$
$req = \mathbf{dependOn}(label(c), provider(c))$

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.

A3. Implementation of the "play" relationship

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

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

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" relationship while taking account of constraints on roles, are described by equations 9 given below:

$$\begin{aligned}
\overline{play}\,(a_\tau, r_i, activity_{s_0})_{Dcr} &= play\,(a_\tau, R_i, activity_{s_0}) \\
\overline{play}\,(a_\tau, r_i, activity_{s_0})_{Imp(r_i,r_j)} &= play\,(a_\tau, R_i \times R_j, activity_{s_0}) \\
\overline{play}\,(a_\tau, r_i, activity_{s_0})_{Eqv(r_i,r_j)} &= play\,(a_\tau, R_i \times R_j, activity_{s_0}) \\
&\quad \mathbf{or}\ play\,(a_\tau, R_j \times R_i, activity_{s_0}) \\
\overline{play}\,(a_\tau, r_i, activity_{s_0})_{Phb(r_i,r_j)} &= play\,(a_\tau, R_i, activity_{s_0}) \\
&\quad \mathbf{and\ not}\ play\,(a_\tau, R_j, activity_{s_0})
\end{aligned} \tag{9}$$

A4. Figures

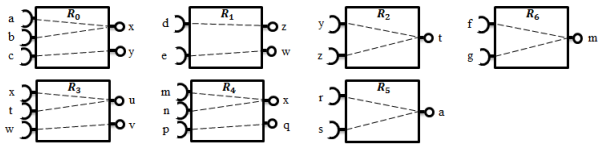


Figure 3 – Running collaboration context of a system

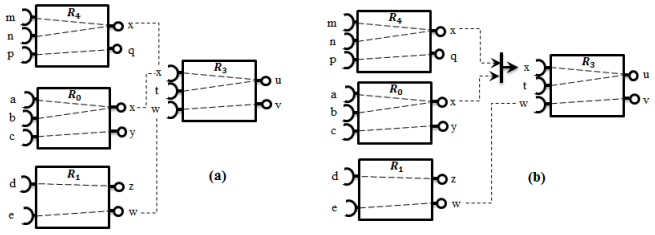
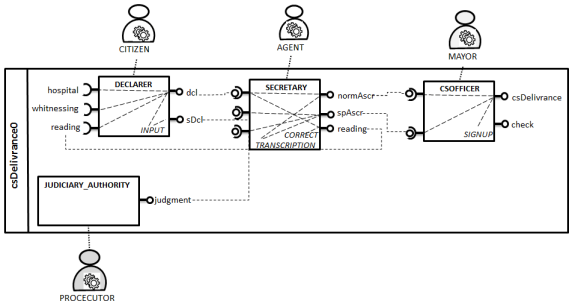
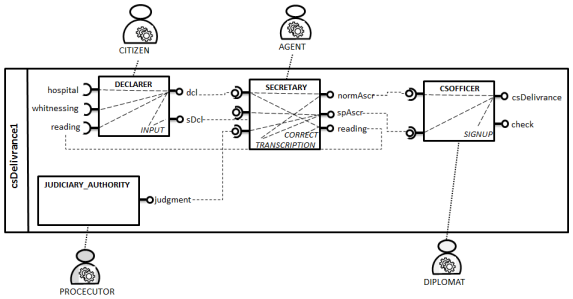


Figure 4 – Factorizing a workflow - (a) a collaboration scheme. (b) the factorized collaboration scheme equivalent to the one on (a).

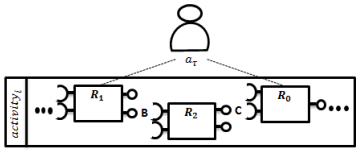


(a) Process of issuing a civil status certificate by the mayor

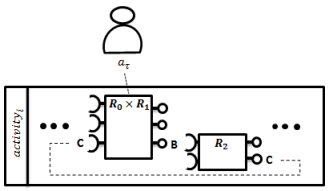


(b) Process for the issuance of a civil status certificate by a diplomat

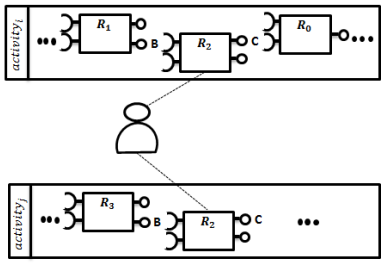
Figure 5 – Decomposition of the previous activity on figure 2, into two sub-activities, $csDelivrance0$ and $csDelivrance1$



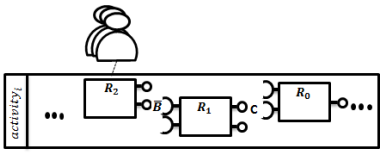
(a) Actor a_τ playing two roles



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



(c) An actor involved in two parallel activities



(d) Several actors playing same role r_2 in an activity

Figure 6 – relation play scenarios