A Secure Framework for Dynamic Task Delegation in Workflow Management Systems

Khaled Gaaloul

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR Sciences et Technologies

Une Approche Sécurisée pour la Délégation Dynamique de Tâches dans les Systèmes de Gestion de Workflow

THÈSE

présentée et soutenue publiquement le 05/10/2010

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Khaled Gaaloul

Composition du jury

Rapporteurs :	Salima Benbernou, Professeur à l'Université Paris Descartes, LIPADE Chihab Hanachi, Professeur à l'Université Paul Sabatier, IRIT
Examinateurs :	Claude Godart, Professeur à l'Université Henri Poincaré, LORIA Selmin Nurcan, Maître de Conférences à l'Université Paris 1, CRI Andreas Schaad, Senior Researcher, SAP AG, Allemagne Samir Tata, Professeur à Telecom SudParis, INF
Directeur de thèse :	François Charoy, Maître de conférence, HDR, à l'Université Henri Poincaré, LORIA

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Résumé

Les systèmes de gestion de workflow font maintenant partie de l'environnement classique des grandes organisations. Ces systèmes sont cependant aujourd'hui considérés comme trop rigides et de nombreux travaux ont pour but d'introduire de la flexibilité dans la modélisation et l'exécution de leurs procédés. Dans cette problématique, la prise en compte de la flexibilité organisationnelle est une étape importante. C'est à cette dernière que nous allons nous intéresser à travers un mécanisme particulier : la délégation de tâches. En effet, la délégation est un mécanisme qui permet d'obtenir une certaine flexibilité organisationnelle dans un système de gestion de workflow. Elle permet également d'assurer une forme de délégation des autorisations dans un système de contrôle d'accès. Dans ce contexte, une délégation sécurisée de tâches implique la présence d'un ensemble d'évènements de délégation et de règles définissant les possibles délégations d'autorisation ainsi que les moyens de contrôler les politiques associées.

Dans ce mémoire, nous définissons une approche sécurisée pour la délégation dynamique de tâches dans les systèmes de gestion de workflow. Pour ce faire, nous identifions les évènements spécifiques du modèle de tâches correspondant à la délégation qui entrainent des changements dynamiques de la politique d'autorisation. Puis, nous montrons comment notre approche permet de contrôler dynamiquement les autorisations liées à la délégation et comment elle peut être intégrée dans les systèmes de contrôle d'accès existants. Afin de contrôler le comportement de délégation et de spécifier ses politiques d'autorisation, nous recueillons les événements pertinents qui définissent le chemin d'exécution des tâches ainsi que les politiques générées pour la délégation. Finalement, nous proposons une technique qui automatise les politiques de délégation et qui permet d'accroître la conformité des changements dus à la délégation dans la politique d'autorisation existante.

Mots-clés: Workflow, tâche, délégation, contrôle d'accès, politique d'autorisation.

Abstract

Task delegation presents one of the business process security leitmotifs. We currently observe a move away from predefined strict workflow modelling towards dynamic approaches supporting flexibility on the organisational level and dynamic authorisation on the security level. One specific approach is that of task delegation. Delegation defines a mechanism that bridges the gap between both workflow and access control systems. There are two important issues relating to delegation, namely allowing task delegation to complete, and having a secure delegation within a workflow. Delegation completion and authorisation enforcement are specified under specific constraints. Constraints are defined from the delegation context implying the presence of a fixed set of delegation events to control the delegation execution.

In this dissertation, we aim to reason about delegation events to model task delegation and to specify delegation policies dynamically. To that end, we present an event-based task delegation model to monitor the delegation process. We then identify relevant events for authorisation enforcement to specify delegation policies. Subsequently, we propose a task-oriented access control model to address these requirements. Using our access control model, we analyse and specify delegation constraints into authorisation policies. Moreover, we propose a technique that automates delegation policies using event calculus to control the delegation execution and to increase the compliance of all delegation changes in the existing policy of the workflow.

Keywords: Workflow, access control, task, delegation, authorisation policy.

Remerciements

Je voudrais exprimer mes sentiments les plus sincères envers les personnes qui sans lesquelles ce travail de thèse n'aurait pas pu voir le jour. Leur aide, accompagnement et soutien m'ont été indispensables afin de pouvoir aboutir aux contributions de ma thèse.

Je voudrais tout d'abord exprimer ma reconnaissance envers tous les membres du jury pour la grande attention qu'ils ont bien voulu porter à mon travail.

Je suis très reconnaissant à Monsieur François Charoy, mon directeur de thèse, qui m'a encadré et dirigé dans mes recherches tout au long de ces années. Je lui dis ma gratitude pour l'aide compétente qu'il m'a apportée, pour ses encouragements et pour la confiance qu'il m'a toujours témoignée.

Mes plus chaleureux remerciements vont également à Monsieur Claude Godart, Professeur à l'Université Henri Poincaré, pour m'avoir accueilli dans son équipe, pour sa disponibilté et pour ses conseils précieux pendant les moments les plus difficiles de ma thèse. Qu'il trouve ici l'expression de ma profonde reconnaissance.

Je remercie très sincèrement mes rapporteurs Madame Salima Benbernou, Professeur à l'Université Paris Descartes, et Monsieur Chihab Hanachi, Professeur à l'Université Paul Sabatier, pour avoir bien accepté d'être mes rapporteurs et pour avoir bien voulu lire et évaluer mon travail de thèse. Je les remercie pour leurs lectures approfondies de mon mémoire de thèse, pour tout le temps qu'ils m'ont accordé et pour les remarques très constructives qu'ils m'ont données et qui ont été bénéfiques à la réalisation de ce manuscrit.

Je remercie également Monsieur Samir Tata, Professeur à Telecom SudParis, Monsieur Andreas Schaad, Senior Researcher à SAP Allemagne, et Madame Selmin Nurcan, Maître de Conférences à l'Université Paris 1, pour leur participation au jury de cette thèse et le temps qu'ils ont bien voulu consacrer à l'évaluation de mon travail.

Je remercie l'ensemble des membres de SAP Research avec qui j'ai eu le plaisir de travailler ces trois dernières années. Je pense particulièrement à Philip Miseldine, Andreas Schaad, Zoltan Nochta, Christian Wolter, Achim Brucker, Mathias Kohler et Helmut Petritsch. Mes plus amicaux remerciements vont aussi à mes collègues du LORIA, notamment Walid Fdhila, Bilel Nefzi, Ehteshem Zahoor, Nawal Guermouche et Gérald Oster qui ont permis de faire de cette expérience de thèse une expérience riche tant scientifiquement que humainement. Je tiens à remercier aussi toutes les personnes qui travaillent dans l'ombre mais qui répondent toujours présentes quand nous avons besoin d'elles, et particulièrement Isabelle Herlich et Rudiger Winter.

Enfin, je voudrais remercier ma famille à qui je dédie ce travail, ma tante et sa famille, en particulier mes cousins Sahbi et Ziad pour leurs soutiens et encouragements. Merci à tous mes amis, en particulier Walid, Bilel, Morfus, Imotep, Samu, Phil, Mohsen, Mohamed et à tous ceux que je vois encore et ceux que le vent a emmené vers d'autres rives.

Je dédie ce travail à mes parents à qui je dois TOUT, à ma sœur et mes frères, surtout mon grand frère Walid mon exemple et mon meilleur soutien. À la mémoire de mes grands-parents Ahmed et Aicha, vous me manquez énormément. Que vos âmes reposent en paix.

Contents

List of Figures

List of Tables

Résumé de la Thèse

1 Introduction

Les systèmes de gestion de workflow font maintenant partie de l'environnement classique des grandes organisations. Ces systèmes permettent la gestion et l'automatisation des procédés métiers et organisationnels. Un procédé est classiquement défini comme un ensemble d'activités coordonnées, également appelées tâches [WFM99]. Les systèmes existants sont cependant aujourd'hui considérés comme trop rigides et de nombreux travaux ont pour but d'introduire de la flexibilité dans la modélisation et l'exécution des procédés. Dans cette problématique, la prise en compte de la flexibilité organisationnelle, c'est à dire des acteurs humains interagissant avec des systèmes de workflow structurés, est une étape importante. C'est à cette dernière que nous allons nous intéresser à travers un mécanisme particulier : la délégation de tâches [Sch03].

La délégation permet de façon dynamique de transférer l'exécution d'une tâche de la personne à laquelle elle était assignée à un autre utilisateur tout en franchissant la barrière des rôles définies par l'organisation. En effet, si nous considérons le graphe des relations entre les rôles d'une organisation, la délégation va permettre, de façon temporaire, exceptionnelle, de transgresser certaines règles liées à ce graphe. Il faudra donc s'assurer que cette délégation entraine les délégations d'autorisation nécessaires dans le système de contrôle d'accès qui est en général basé sur le graphe organisationnel. En particulier, la délégation d'une tâche à un subordonné, ou la délégation d'une tâche à un membre d'un autre département de l'organisation va nécessiter des adaptations dynamique de la politique de sécurité. Cette délégation doit elle même faire partie de la politique de sécurité de l'organisation.

Normalement, les organisations établissent un ensemble de politiques de sécurité qui règlent la façon de gérer les procédés métiers et les ressources [AW05]. Une politique simple peut spécifier comment une tâche peut être assignée à un utilisateur. Une politique plus complexe peut spécifier des contraintes d'autorisation supplémentaires pour permettre la délégation. Les contraintes d'autorisation de délégation sont définies par rapport à des évènements sur les couches de contrôle, de données et d'assignement des tâches du workflow [GSFC08]. Une délégation sécurisée de tâches implique la présence d'un ensemble d'évènements de délégation défini et de règles définissant les possibles délégations d'autorisation ainsi que les moyens de contrôler les politiques associées.

L'essentiel du travail fait dans le domaine des contraintes de sécurité et des droits d'accès ne traite pas de la délégation de façon assez détaillée. En effet, les travaux traitant du contrôle d'accès basé sur le modèle RBAC ("Role Based Access Control") [SCFY96], qui permettent de définir des politiques de délégation manquent de flexibilité. L'essentiel des possibilités ou des règles de délégations sont définies à priori [SRS+05]. Les travaux traitant des systèmes d'autorisation ne considèrent pas la possibilité de contrôler des politiques dynamiques, c'est à dire susceptible de changer en fonction d'évènements de contexte [BFA99]. Actuellement, les requêtes sur un système de contrôle d'accès sont sans état. La réponse à une requête donnée n'est valide qu'à l'instant ou la requête est faite. Si cette réponse change en raison d'une adaptation de la politique, aucun mécanisme n'existe pour transférer cette nouvelle réponse au demandeur initial de façon proactive. Ce mécanisme est cependant vital pour permettre une délégation dynamique d'autorisation. Lorsque nous déléguons une tâche, une adaptation de la politique peut être nécessaire

en fonction des évènements de délégation. Le délégué, la personne à qui nous déléguons, peut acquérir de nouveaux droits grâce à cette délégation mais il peut les reperdre en cas de révocation de cette délégation. Ces évènements de délégation sont donc liés à des changements dynamiques des autorisations. Ceci ne peut être négligé par un système de sécurité avancé permettant la délégation de tâche de manière dynamique et assurant son intégration de façon automatique.

Dans ce mémoire, nous allons donc proposer la définition d'une approche permettant la délégation dynamique d'autorisation. Celle ci devra supporter des politiques proactives dans un système de contrôle d'accès. Nous motiverons cette approche par un exemple nécessitant la délégation de tâches. Nous identifierons les évènements spécifiques du modèle de tâches correspondant à la délégation qui entrainent des changements dynamiques de la politique. Nous séparerons les différents aspects de la délégation entre les utilisateurs, les tâches et les évènements, et nous les spécifierons en termes de politiques de délégation. Les politiques définies incluront les comportements nécessaires pour permettre les interactions en temps réel assurant la délégation dynamique d'autorisation. Nous présenterons ensuite les environnements de contrôle d'accès existants et discuterons leur fonctionnalités et leur limitations. Puis, nous montrerons comment notre approche permet de contrôler dynamiquement les autorisations liées à la délégation et comment elle peut être intégrée dans les systèmes existants. Pour finir, nous présenterons une technique pour l'intégration des politiques de délégation dans les politiques prédéfinies. Afin de contrôler le comportement de délégation et de spécifier ses politiques, nous recueillerons les événements pertinents qui définissent à la fois le chemin d'exécution des tâches ainsi que les politiques générées pour la délégation. En utilisant, le calcul événementiel, nous proposerons une technique qui automatise les politiques de délégation et qui permet d'accroître la conformité des changements dus à la délégation dans la politique d'autorisation existante.

2 État de l'art

La plupart des travaux réalisés dans le domaine des contraintes de modélisation et de sécurité et pour les systèmes de gestion de workflow ont peu de travaux connexes portant sur la délégation [AW05, Ven03]. Cette observation est corroborée par les études réalisées par Russel *et al.* [RvdAHE05] et Hung *et al.* [HK03]. Ils ont souligné que les solutions existantes, telles que le Modèle d'autorisation de workflow (WAM) [AW05], restent statiques et ne supportent pas les contraintes de sécurité dynamiques tel que la délégation d'autorisation.

Le modèle d'autorisation de workflow (WAM) présente un cadre conceptuel, logique et un modèle d'exécution qui se concentre sur l'application des flux d'autorisation entre les tâches et leurs dépendances [AW05]. Bien que WAM aborde la synchronisation des flux d'autorisation avec le workflow et assure une spécification des contraintes temporelles d'une façon statique, il ne suffit pas à assurer la sécurité des workflows en général et de la délégation de tâches en particulier. En effet, nous avons besoin de contrôler une délégation en nous basant sur les différents aspects du workflow tels que les tâches, les événements et les données.

Russel *et al.* ont proposé une approche supportant la délégation [RvdAHE05]. Ils

ont décrit le cycle de vie d'un élément de travail sous la forme d'un diagramme d'états-transitions en mettant l'accent sur les allocations de ressources. Le principal inconvénient de cette approche est le fait d'être statique. De plus, elle ne tient pas compte d'autres événements (transitions) lors de l'exécution de la délégation et ne supporte pas les interactions dynamiques au sein des workflows.

Le modèle de contrôle d'accès extensible XACML (eXtensible Access Control Markup Language) a été développé afin de définir de manière uniforme la spécification des politiques de contrôle d'accès dans le langage XML [Tim05]. Les politiques sont décrites par des règles qui peuvent être restreintes par des conditions. Ces règles s'appliquent à des cibles définies par un ensemble de sujets, de ressources et d'actions. Ces spécifications sont reprises lors d'une requête, où elles seront confondues avec les attributs de la requête (e.g. le demandeur comme sujet, la tâche comme ressource, la permission comme action). Les résultats ou les effets d'une politique peuvent être : Permis, Refus, Non applicable ou Indéterminée. Le standard XACML actuel ne fournit pas de support explicite de délégation. Cependant, il y a eu quelques essais d'extension du modèle pour des règles de délégation mais qui restent loin d'être concrètes pour des politiques réelles de délégation [SRS+05].

Chadwick et al. [CON06] ont proposé une solution XACML supportant une délégation dynamique des autorisations. L'approche décrit une entité appelée le service de validation des attributs. Elle a pour but de se synchroniser avec le module XACML de décisions ("Policy Decision Point") pour la prise de décisions d'autorisation. L'architecture proposée offre un moyen souple et dynamique pour gérer les informations d'identification, mais elle ne couvre pas tous les aspects de délégation dynamique.

Seitz et al. [SRS+05] ont étudié la manière dont un système de gestion d'autorisation, utilisant le langage XACML, peut être étendu à des mécanismes de délégations administratives. Ils ont mis au point un module qu'ils ont appelé "Delegent" pour la gestion des modifications autorisées sur des politiques. Ce travail a été implémenté par la suite dans une solution logicielle appelée Axiomatics pour la gestion de contrôle d'accès. L'idée est de fournir un outil d'administration pour la politique de contrôle d'accès supportant la mise à jour de ces politiques. Cependant, cette administration reste passive et sans états. Elle manque de réactivité pour supporter des changements dynamiques de politiques. Nous avons besoin d'une approche réactive pour tenir compte des événements de délégation lors de l'administration des politiques d'autorisation.

À notre connaissance, il n'y a pas de travaux spécifiques à la délégation de tâches dans les systèmes de workflow. De plus, la plupart des travaux qui traitent des problèmes d'autorisation relatifs à la délégation ignorent l'aspect organisationnel du workflow et restreignent la délégation à un simple problème d'attribution de rôles [BS00]. Outre le besoin de collaboration, il y aussi le besoin en interactions humaines inter-organisationnelles qui sont définies par des mécanismes de délégation ad-hoc, ce qui est le cas dans notre exemple de motivation e-gouvernemental (voir section 3). En effet, le fait de déléguer une tâche exige plus d'efforts de spécifications liées à des événements spécifiques de délégation. Dans la section 4 nous définissons un modèle de délégation de tâches qui fait à la fois parti du procédé métier du workflow ainsi que des contraintes de sécurité liées à l'autorisation. À partir de ce modèle, nous identifions les évènements permettant de déterminer les politiques dynamiques de délégation afin d'assurer des décisions proac-

tives lorsque des événements sont déclenchés au cours de la délégation des tâches (voir section 6). Dans la section 7, nous nous intéressons au problème d'administration des politiques d'autorisation en développant un outil d'intégration des politiques dynamiques de délégation dans la politique globale des workflows.

3 Contexte et problématique

3.1 Exemple de motivation

Pour illustrer la problématique que nous voulons développer, nous allons nous appuyer sur un exemple tiré d'un cas réel impliquant la délégation de tâches. Dans le cadre des procédures criminelles au niveau européen, ont été mises en place des procédures d'assistance judiciaire mutuelle. Ceci permet par exemple à un état membre de faire exécuter une mesure de protection des témoins par un autre état dans une procédure criminelle [R4e06]. Nous ne décrivons ici que la partie Eurojust (European Judicial Cooperation Unit) du pays A de la procédure d'assistance mutuelle ("Mutual Legal Assistance", MLA). Elle consiste à recevoir une requête d'assistance d'un membre d'Europol (European Police Office) pour pouvoir la traiter et la transmettre à l'autorisation concernée du pays B (voir figure 1). L'utilisatrice Alice, qui a le rôle *Procureur* est assignée à une partie du procédé d'Eurojust A. Les activités du procédé sont représentées comme des tâches.

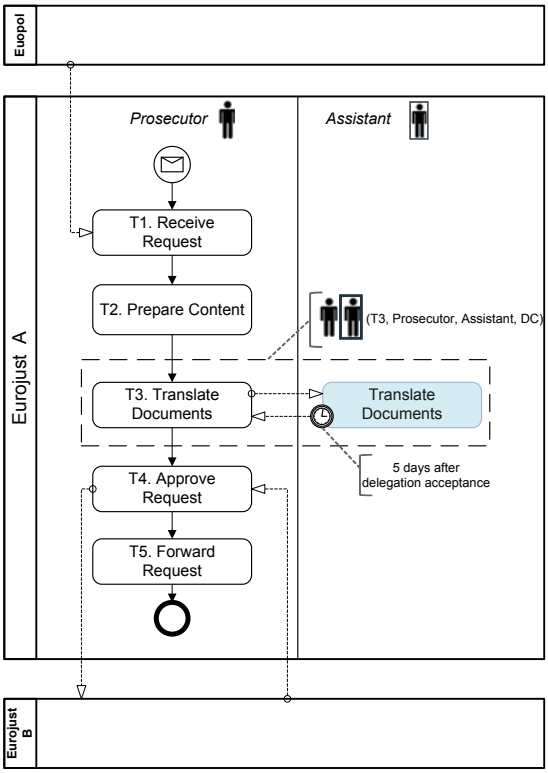

Figure 1: Scénario de délégation MLA

Nous avons utilisé la notation BPMN (Business Process Modeling Notation) pour modéliser notre exemple. BPMN définit une notation standard pour la modélisation de procédés métiers [OMG06]. Dans ce scénario, la tâche "Translate Documents" T3 n'est accessible au départ que par le *Procureur* Alice. Ceci est défini par la politique du procédé. Cette tâche est de longue durée et il est prévu 5 jours pour la terminer. Alice n'est pas disponible pour la réaliser pour cause de maladie et doit donc la déléguer à Bob. Bob a pour rôle *Assistant* et est subordonné au *Procureur* dans la hiérarchie de l'organisation. La délégation doit fournir les moyens suffisants pour permettre ce type de flexibilité organisationnelle. Ainsi, Alice et Bob sont respectivement le *délégant* et le *délégué*.

Une politique peut être définie comme un moyen pour définir les accès aux ressource d'une tâche. Nous définissons une politique d'autorisations P pour le procédé MLA. Pendant une délégation, la politique P est mise à jour pour que l'utilisateur Bob puisse réaliser la tâche à la place d'Alice. Bob revendique la tâche. Une requête de contrôle d'accès est effectuée. L'accès est accordé et Bob peut exécuter la tâche. Après deux jours, Alice revient au travail. Elle veut reprendre la tâche qui lui était affectée. Elle réclame à nouveau la tâche. En raison de sa qualification, la tâche est réaffectée à Alice et retirée de la liste de tâches de Bob. La politique de sécurité doit être à nouveau mise à jour pour refléter le fait que seule Alice a maintenant accès à la tâche. La requête d'accès qui avait permis à Bob d'accéder aux ressources de la tâche retourne maintenant une décision de refus.

Dans les environnements de contrôle d'accès classique, un mécanisme qui préviendrait Bob que son accès à une ressource est annulé automatiquement n'existe pas. Il n'est pas possible de révoquer une réponse donnée par une requête d'accès précédente. En outre, un contrôle manuel des droits d'accès couramment accordés pour l'exécution de tâche serait long, couteux et source d'erreurs. Un mécanisme dynamique permettrait d'informer dynamiquement le délégué d'un changement de politique en fonction des évènements de délégation. Ceci nécessite de supporter certaines interactions spécifiques dans l'architecture de contrôle d'accès utilisée. Ces interactions concernent en particulier les évènements de délégation de tâches qui doivent être capturés et retournés à l'émetteur de la requête pour prendre les mesures appropriées.

3.2 Analyse et discussion

Nous allons nous baser sur un modèle de délégation basé sur les rôles pour contrôler des interactions inter humaines dans le contexte de tâches de longue durée. Nous faisons l'hypothèse que l'exécution d'une tâche est atomique et que la délégation d'autorisation est accordée uniquement au délégué. Ainsi, nous ne considérons pas la possibilité de faire de la délégation en cascade ou de la délégation partielle [ZOS03]. Lors d'un évènement de délégation, nous définissons comme suit la relation de délégation :

Definition 1 (Relation de Délégation) *Une relation de délégation* $RD \subseteq T \times U \times U \times 2^{DC}$ *avec T l'ensemble des tâches, U l'ensemble des utilisateurs et DC l'ensemble des contraintes de délégation. Une relation de délégation pour une tâche* t_i *est définie comme suit :* $RD = (t_i, u_1, u_2, \{DC\})$*, où* t_i *est la tâche déléguée,* u_1 *le délégateur et* u_2 *le délégué.*

Les contraintes de délégation correspondent aux droits de déléguer par rapport à la politique définie pour le procédé. Par exemple, *DC* est basé sur la hiérarchie de rôles (RH) d'Eurojust, où l'*Assistant* Bob est sous l'autorisation du *Procureur* Alice. De plus, *DC* implique aussi une délégation temporaire. Une période de délégation doit être définie. Bob n'est pas autorisé à dépasser la date limite de T3 (5 jours de travail). Nous définissons la relation de délégation pour T3 ainsi :
RD = (T3,Alice,Bob,{RH,5 jours}).

La disponibilité d'un mécanisme dynamique ou proactif d'exécution des politiques de sécurité est vital pour permettre la délégation de tâches de longue durée. Cela nécessite d'être capable de capturer les évènements de délégation et de les transmettre à l'émetteur de la requête pour faire les mises à jour nécessaires dans le système de contrôle d'accès. Actuellement, nous savons contrôler les droits d'accès d'une délégation à travers une adaptation de la politique, pour permettre au délégué d'effectuer la tâche déléguée. Ensuite nous devons mettre à jour la relation de délégation dans la politique du procédé chaque fois qu'un tel évènement est produit. Cela consiste en l'introduction de nouvelles règles d'autorisation pour le délégué. Si cette règle change suite à une révocation, une nouvelle réponse devra être transmise au délégué dynamiquement.

En plus, nous avons besoin de contrôler le comportement de la délégation et de spécifier ses politiques d'autorisation. Nous aurons besoin d'une technique qui automatise les politiques de délégation et qui permet par la suite d'accroître la conformité des changements dus à la délégation dans la politique globale d'autorisation.

4 Politiques de délégation de tâches basées sur un modèle événementiel

Dans cette section, nous nous intéressons à l'identification d'événements qui pourraient être à l'origine de la définition de nouvelles politiques d'autorisation. Tout d'abord, nous présentons notre modèle de délégation de tâches, où nous définissons les transitions comme des événements décrivant le cycle de vie d'une tâche. Ensuite, nous analysons les événements capable d'appliquer de nouvelles règles de changement aux politiques de délégation. En effet, notre préoccupation est d'assurer une politique de délégation dynamique supportant l'exécution d'une tâche déléguée. Le but est de pouvoir répondre aux changements dynamiques qui peuvent se produire lorsqu'une tâche est déléguée. Ceci est motivé par le fait que lorsqu'un événement se produit, notre politique de gestion d'accès doit répondre à ce changement conformément à l'évolution de l'exécution de la tâche déléguée.

4.1 Modèle de délégation de tâches (MDT)

Nous définissons un modèle de délégation de tâches (appelé MDT). Ce modèle est basé sur les spécifications du cycle de vie d'une tâche définies par la WfMC [WFM99]. MDT définit un diagramme UML (Unified Modeling Notation) d'états/transitions d'une tâche au sein d'un procédé de workflow. Une tâche une fois crée, est généralement attribuée à un utilisateur. L'utilisateur peut par la suite l'exécuter ou la déléguer à quelqu'un d'autres. La délégation dépend du droit dont dispose cet utilisateur pour requérir une demande de délégation.

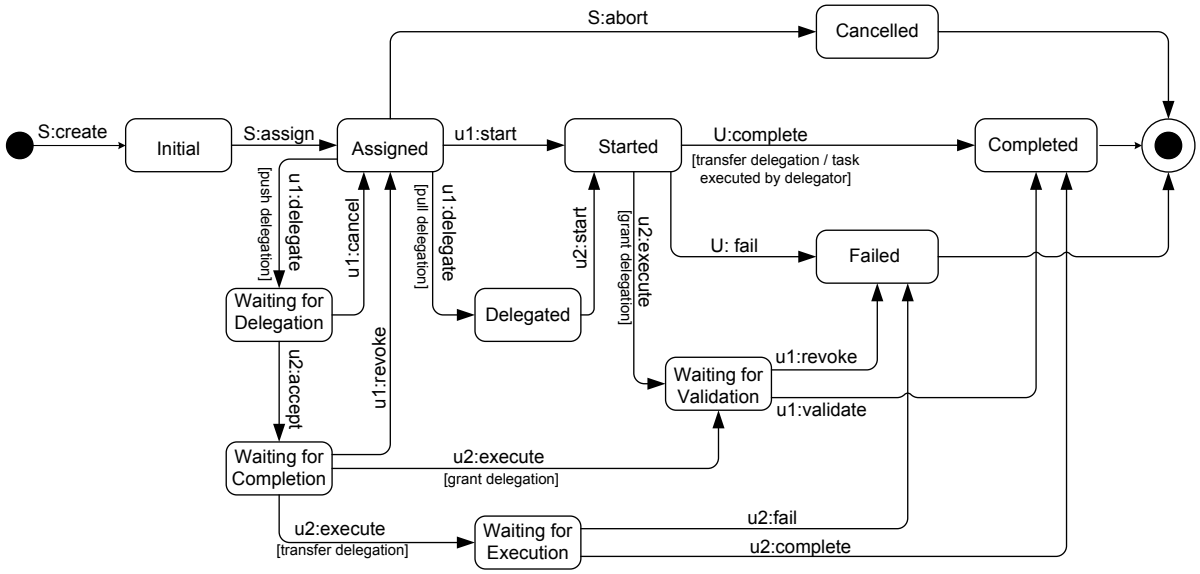

Figure 2: Modèle de délégation de tâches (MDT)

Les événements intermédiaires définissent le contrôle de délégation de tâches (e.g. *"delegate"*, *"cancel"*, *"revoke"*). Par exemple, si le délégant a besoin d'annuler son action, le TDM doit proposer une alternative pour annuler la délégation (l'événement "cancel") et revenir à l'état précédent : "Assigned". Le contrôle de délégation reste interne au modèle de tâche et son comportement suit le flux de contrôle habituel, où "Completed", "Cancelled" et "Failed" représentent les états finaux d'une tâche (voir figure 2).

Il faut noter que les préfixes S ou U indiquent que les transitions sont soit initiées par le système de workflow, soit une ressource humaine (un utilisateur). Nous définissons u_1 et u_2 appartenant à l'ensemble des utilisateurs U, avec respectivement u_1 et u_2 le délégant et le délégué.

4.2 Analyse de besoins en sécurité

Dans notre modèle MDT, nous définissons les transitions comme des événements qui contrôlent le comportement de délégation. Nous avons enrichi ce modèle avec des propriétés décrivant le mode et le type de délégation. En effet, une tâche déléguée peut s'exécuter de différentes manières selon le contexte de son exécution. Un délégant peut avoir à sa

disposition une liste de délégués potentiels qui peuvent exécuter la tâche pour son compte. Il s'agit dans ce cas d'un mode *"Pull"*. Une autre propriété concerne le type d'attribution de privilèges (permissions) afin d'accéder aux ressources de la tâche pour l'exécuter. Dans la littérature, nous distinguons deux types : *"grant"* et *"transfer"* [CK06]. Dans ce qui suit, nous analysons les exigences en sécurité qui doivent être prises en compte pour définir les politiques de délégation basées sur les événements de délégation (voir figure 2).

- **Mode de délégation** : Il définit la manière dont une requête de délégation est faite. Le mode *"Pull"* suppose que le délégant dispose d'une liste de délégués potentiels qui peuvent exécuter la tâche pour son compte. Le mode *"Push"* suppose qu'un délégant est en attente d'acceptation d'un délégué correspondant au profil de la requête (e.g. rôle). Nous définissons les évènements suivants *"accept"*, *"cancel"* et *"revoke"* comme les évènements relatifs au mode *"Push"*.

- **Type de délégation** : Il existe deux types de délégation : *"grant"* et *"transfer"*. Une délégation de type *"grant"* permet à la fois au délégant et au délégué que le droit d'accès (privilèges) soit disponible. À ce titre, le délégant garde toujours le contrôle sur sa tâche et a le droit de valider (*"validate"*) ou de révoquer (*"revoke"*) le travail du délégué. Une délégation de type *"transfer"* donne tous les droits au délégué. Il n'y a pas de validation requise et la tâche est terminée (*"complete"*/*"fail"*) par le délégué.

- **La délégation d'autorisation** : Elle permet à un délégant de céder une partie de ses privilèges à un délégué qui, à priori, ne possèdent pas les autorisations nécessaires pour exécuter la tâche. Par exemple, *"delegate"* définit un événement qui va déclencher la délégation de la tâche. Ainsi, *"delegate"* va entraîner l'application d'une nouvelle politique de contrôle d'accès pour le délégué. Nous allons assurer une nouvelle règle dans la politique afin d'autoriser le délégué à exécuter la tâche déléguée.

- **Application du contrôle d'accès** : Le but est de garantir une politique de délégation dynamique. Par exemple, l'évènement *"cancel"* implique la révocation de la délégation où le délégant va reprendre le contrôle sur sa tâche et, par conséquent, annuler la décision précédente. Cette annulation définit une nouvelle règle dans la politique en appliquant un refus d'accès instantané au délégué.

Par la suite, nous classons les événements de délégation et identifions les relations entre ces événements, les propriétés de délégation et leur impact sur les politiques d'autorisation (voir tableau 1).

En effet, pour une délégation de type "grant" le mode Push est basé sur les évènements : *u1:delegate*, *u2:accept*, *u1:cancel*, *u2:execute*, *u1:validate* et *u1:revoke*. Les changements de politiques peuvent s'opérer lors des évènements : *u1:delegate*, *u2:accept* et *u1:revoke*. Pour mieux expliquer le tableau, nous considérons les deux exemples suivants :

Évènements	Mode Push		Mode Pull		Changement de politiques
	Grant	Transfer	Grant	Transfer	
u1:delegate	✓	✓	✓	✓	✓
u2:accept	✓	✓			✓
u1:cancel	✓	✓			
u2:execute	✓		✓		
u1:validate	✓		✓		
u1:revoke	✓		✓		✓
U:fail		✓		✓	
U:complete		✓		✓	

Table 1: Politiques d'autorisations basées sur les évènements de délégation

Exemple 1 : Dans le scénario e-gouvernemental que nous avons présenté, nous pouvons observer une politique de délégation dynamique régie par les évènements qui peuvent avoir lieu au cours de l'exécution de la tâche T3. L'utilisateur Alice est de retour avant la terminaison de délégation. Alice devrait révoquer l'effet de délégation en annulant ce qui a été effectué par Bob. Alice sera en mesure d'exécuter la tâche T3 et ainsi annuler la politique d'autorisation initialement déléguée à Bob. L'événement *"revoke"* va mettre à jour la politique globale et une notification sera transmise à Bob pour procéder aux actions nécessaires d'annulation.

Exemple 2 : L'événement *"fail"* est défini dans les deux modes de délégation (Push et Pull). Il intervient lors d'une délégation de type *"transfer"*, où un délégué termine la tâche sans besoin de validation. Les politiques de délégation prendront fin suite à la terminaison de la tâche et aucun changement de politiques n'est nécessaire.

5 Les modèles de contrôle d'accès dans les workflows

En matière de gestion des autorisations en général, le modèle RBAC est largement adopté. Il permet aux administrateurs d'attribuer des rôles aux utilisateurs. Dans notre travail de thèse, nous essayons d'étendre ce modèle dans le but de résoudre les problèmes liés à la délégation. En effet, la délégation est un mécanisme qui permet à un utilisateur d'attribuer un sous-ensemble de ses autorisations à d'autres utilisateurs qui ne les possèdent pas initialement.

Dans ce qui suit, nous définissons un modèle de contrôle d'accès basé sur le modèle RBAC. Notre objectif sera par la suite de raisonner sur la délégation de tâches du point de vue de l'organisation (ressources humaines) et celui des procédés (ressources matérielles) afin d'analyser et de spécifier les contraintes de délégation.

5.1 Modèle de contrôle d'accès basé sur la tâche (CAT)

Nous proposons un modèle de contrôle d'accès basé sur la tâche (appelé CAT) (voir figure 3). Formellement, nous identifions les ensembles *U, R, OU, T, P, S* et *TI* en tant

que respectivement les ensembles des utilisateurs, de rôles, des unité d'organisations, des tâches, des permissions, des sujets et des instances de tâches. Un sujet définit un utilisateur choisissant un rôle lors d'une session (instanciation de la tâche). Cela permet de gérer l'attribution (l'affectation) des instances de tâches ($claimed_{by}$) lors de l'activation d'un rôle par un utilisateur. Nous définissons la relation RH (Rôle Hiérarchique), avec RH un ordre partiel dans R. r_i et $r_j \in R$, si r_i est un rôle supérieur à r_j, alors r_i hérite automatiquement des permissions de r_j.

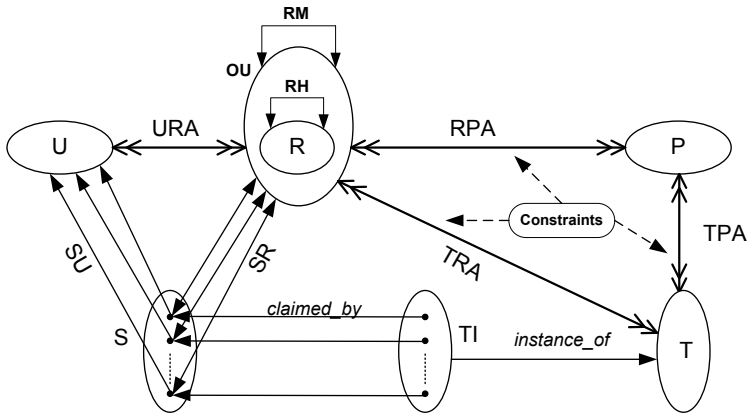

Figure 3: Modèle de contrôle d'accès basé sur la tâche (CAT)

Nous définissons aussi la relation RM (Role Mapping), $RM \subseteq OU_i \times OU_j$, avec OU_i et OU_j deux unités d'organisations. RM définit les rôles externes qui accèdent aux ressources distribuées inter-organisations. Concrètement, cela signifie que si $r_k \in OU_i$, $r_l \in OU_j$, et r_l est un rôle projeté sur r_k, alors r_l hérite automatiquement des permissions de r_k.

Définitions des relations :

- $URA \subseteq U \times R$, définit la relation d'attribution des rôles aux utilisateurs.
- $RPA \subseteq R \times P$, définit la relation d'attribution des permissions aux rôles.
- $TPA \subseteq T \times P$, définit la relation d'attribution des permissions aux tâches.
- $TRA \subseteq T \times R$ définit la relation d'attribution des tâches aux rôles.

Définitions des fonctions :

- $SU: S \rightarrow U$ une fonction qui définit l'association du sujet à son utilisateur.
- $SR: S \rightarrow R$, une fonction qui définit l'association d'un sujet à son rôle, avec $SR(s) = r, (SU(s), r) \in URA\}$ et le sujet s a comme permission $p|(r, p) \in RPA\}$.

- $instance_{of}$: $T \rightarrow TI$, une fonction qui définit l'association d'une tâche à ses instances de tâches.

- $claimed_{by}$: $TI \rightarrow S$, une fonction qui définit l'affectation d'une instance de tâche à un sujet tel que s = $claimed_{by}$(t_i) avec :
$\{t_i = instance_{of}(t), (r, u) \in URA | (SR(s) = r \bigwedge SU(s) = u), (t, r) \in TRA\}$.

Définitions des Contraintes :

Ici, nous parlons des contraintes de séparation de devoir ("*Separation of duty*" : SoD) et de l'attachement de devoir ("*Binding of duty*" : BoD). Nous définissons les relations entre les tâches pour les SoD et BoD comme suit :

TT_{SOD} : $\{(t_i, t_j) \in T \times T \mid t_i$ *est exclusive avec* $t_j\}$

TT_{BOD} : $\{(t_i, t_j) \in T \times T \mid t_i$ *est attachée avec* $t_j\}$

Si $(t_i, t_j) \in TT_{SOD}$, alors t_i and t_j ne peuvent pas être affectées au même utilisateur. Si $(t_i, t_j) \in TT_{BOD}$, alors t_i and t_j doivent être affectées au même utilisateur.

Contributions et Motivations :

Nous modélisons l'affectation des permissions aux tâches et aux rôles en nous basant sur les besoins en ressources humaines (les utilisateurs) et matérielles (les objets métiers). Le tuple (P,T,R) définit les relations TRA, TPA and RPA qui spécifient le contexte d'exécution d'une tâche (voir définition 2).

Definition 2 (Les Conditions d'affectation d'une tâche) *Une tâche* t *est affectée à un rôle* r *si :* $(t, r) \in TRA \Rightarrow \{p \in P | (t, p) \in TPA\} \subset \{p | (r, p) \in RPA\}$.

La principale contribution est de préciser les conditions d'affectation des tâches basée sur le modèle de contrôle d'accès (voir figure 3). En effet, deux conditions doivent être vérifiées pour satisfaire la relation TRA. La première condition est liée aux ressources de la tâche définies dans TPA. La deuxième condition est liée à l'attribution de la tâche à un utilisateur qui doit disposer des permissions requises. La validation de cette condition se fait au niveau de la relation RPA.

5.2 Utilisation du modèle CAT pour la délégation de tâches

Le modèle CAT définit la liste de délégués potentiels (RPA) qui peuvent satisfaire les conditions d'affectation d'une tâche déléguée (TPA). Nous définissons une méthode de contrôle d'accès sur la délégation des tâches en utilisant le modèle CAT. Dans ce qui suit,

Algorithm 1: Définition des délégués et des privilèges pour une tâche

Entrées : DR // Relation de Delegation
u_1, u_2 // utilisateurs $\in U$
r_1, r_2 // rôles $\in R$
t_i, t_j // tâches $\in T$
$DR = \emptyset$ /*Initialisation de DR*/
$\{(u_1, r_1), (u_2, r_2)\} \subseteq URA$ /*Attribution des rôles aux utilisateurs*/
$\{(r_1, p_{r1}), (r_2, p_{r2})\} \subseteq RPA$ /*Attribution des permissions aux rôles*/
$(t_i, p_{ti}) \in TPA$ /*Attribution des permissions à la tâche avec $p_{ti} \subset p_{r1}$*/
$(t_i, r_1) \in TRA$ /*Attribution de la tâche au rôle*/
$SU(s_1) = u_1$ /*Association du sujet à son utilisateur*/
$SR(s_1) = r_1$ /*Association du sujet à son rôle*/
$SU(s_2) = u_2$
$SR(s_2) = r_2$
$t_{i1} = instance_{of}(t_i)$
$t_{j1} = instance_{of}(t_j)$
$s_1 = claimed_{by}(t_{i1})$ /*t_{i1} assignée à s_1*/
Precondition :
$(\nexists\ t_j1 \mid (TT_{SOD}(\mathrm{t}_i,\mathrm{t}_j),\ s_2 = claimed_{by}(\mathrm{t}_{j1}))$ AND $(TT_{BOD}(\mathrm{t}_i,\mathrm{t}_j),\ s_1 = claimed_{by}(\mathrm{t}_{j1})))$
Postcondition :
$p_{ti} \subset p_{r2} \Rightarrow s_2 = claimed_{by}(\mathrm{t}_{i1})$ /*t_{i1} déléguée à s_2*/
$p_{ti} \not\subset p_{r2} \Rightarrow p'_{r2} \leftarrow p_{r2} \cup p_{ti}$ /*p'_{r2} est l'ensemble des permissions associées à s_2 pour la durée de délégation*/
Résultat : DR = $(\mathrm{t}_i,\mathrm{u}_1,\mathrm{u}_2,\{DC\})$
$\mathrm{DR}_1 \leftarrow$ instanceOf(DR)
$\mathrm{DR}_1 = (\mathrm{t}_{i1},\mathrm{s}_1,\mathrm{s}_2,\{DC\})$ /*Instanciation de la relation de délégation*/

nous définissons un algorithme (voir algorithme 1) pour décrire la façon dont les délégués seront gérés et si des privilèges sont nécessaires pour satisfaire les conditions de délégation.

La principale contribution de cette méthode est de préciser les conditions d'affectation des tâches déléguées tout en identifiant le délégué et les privilèges. En effet, la méthode est basée sur les exigences actuelles des instances d'une tâche. Ainsi les privilèges délégués dépendent seulement de l'instance de la tâche et non pas de tous les privilèges accordés à la tâche type. Le but est de déléguer seulement les permissions nécessaires à l'exécution de cette instance.

En premier lieu, nous vérifions si les relations TPA et RPA sont valides, alors la tâche t_i est déléguée au délégué u_2 en se basant sur la définition 2. En deuxième lieu, si u_2 membre du rôle r_2 n'a pas l'autorisation nécessaire pour exécuter t_i et qu'il n'y a pas de conflits avec la politique d'autorisation globale (BoD or SoD), alors le délégant lui accordera les privilèges nécessaires à l'instance de la tâche t_{i1}.

6 Une Approche sécurisée pour la délégation de tâches

Dans cette section, nous développons une approche sécurisée pour la délégation de tâches. Notre approche a pour but de spécifier les contraintes exprimées dans notre modèle de

contrôle d'accès en termes de politiques d'autorisation. Nous présentons une architecture modulaire assurant une gestion dynamique pour la délégation de privilèges. Notre approche permet de supporter des politiques de contrôle d'accès proactives régies par les évènements de délégation définis dans la section précédente.

Notre approche sera implantée au sein de systèmes existants de contrôle d'accès dans le cadre de la délégation de tâches. Pour ce faire, lorsqu'une requête est émise, elle est ensuite stockée de manière à informer le délégué si la décision politique à cette requête vient de changer. En effet, la réponse peut évoluer en fonction des changements de politiques se produisant au cours d'une délégation. Des évènements tels que l'annulation, la révocation ou la non validation d'une tâche déléguée doivent forcément entraîner une nouvelle décision à la réponse précédente (voir les changements de politiques identifiés dans le tableau 1). De ce fait, les requêtes antérieures seront réévaluées, et le délégué sera informé que ses droits d'accès ont changé. L'implémentation de cette approche se base sur des systèmes existants de contrôle d'accès. L'idée est de développer un module dynamique comme extension à ces systèmes afin de supporter des politiques de délégation proactives.

6.1 Spécification des politiques d'autorisation à partir du modèle CAT

Une autorisation permet la liaison explicite entre un sujet (utilisateur), une ressource de tâche (objet métier) et ses droits (actions). Cette liaison est définie sur la base des spécifications du modèle CAT. Il inclut les entités définies dans les principales relations : URA, TRA, RPA et TPA.

Une autorisation exprime le droit d'accès d'un utilisateur sur les ressources d'une tâche, où une autorisation définit le droit d'exécuter une action sur une ressource. La définition de l'autorisation doit être précisée dans une politique (voir définition 3). Une politique de contrôle d'accès spécifie l'accès aux ressources d'une tâche à un niveau plus élevé.

Definition 3 (Politique) *Nous définissons une politique* $P \subseteq target \times rule \times 2^C$, *avec* target *(cible) le domaine d'application de la politique,* rule *un ensemble de règles définissant les décisions de la politiques, et* C *l'ensemble des contraintes validant le résultat des règles.*

Une pseudo expression formelle d'un *target* (cible) est :

```
<Autorisation>
<Sujet>[role]
<Resource>[object]
<Action>[operation]
<Tâche>[task type]
< /Authorisation>
```

Un exemple d'une politique où la décision est "Permit" pour un sujet ayant comme rôle *Procureur* sur la tâche T1 "receive request" (voir figure 1) est :

```
<Poliy>
<target>[Procureur1,MLA1,read,T1]
<rule>[Permit]
<C>[none]
< /Poliy>
```

Dans la suite, nous présentons un cadre de contrôle d'accès (ACF) pour la délégation. ACF est définie comme un ensemble de composants logiciels qui accepte les demandes d'accès aux ressources, analyse ces demandes, et retourne une réponse fondée sur cette analyse.

6.2 Vue d'ensemble de l'architecture

Nous présentons les principaux composants de l'architecture qui va assurer la gestion de politiques proactives lors de la délégation de tâche. L'architecture et ses différents modules sont décrits dans la figure (voir figure 4).

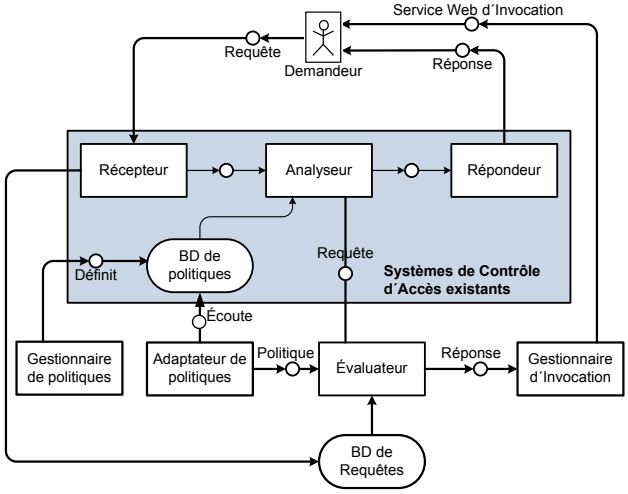

Figure 4: Un cadre sécurisé pour des politiques de délégation proactives

- **Gestionnaire de politiques** : Il permet à un administrateur de définir des politiques de contrôle d'accès. Grâce à une interface graphique, l'administrateur peut naviguer dans la base de données des politiques, sélectionner un document, modifier une politique (la cible, le sujet, les règles d'autorisation), et préciser les décisions pour les éléments sélectionnés. Par exemple, un administrateur définit une politique d'autorisation P qui retourne comme réponse *"permis"* (autoriser) pour la tâche *cible* T3 ayant comme *sujet* l'utilisateur Alice dont le *rôle* est *Procureur*. Dans notre contexte de délégation, nous assumons qu'un délégant est autorisé à administrer des

politiques, et ainsi à définir des règles de délégation. Les politiques de délégation encapsulent les attributs d'identification du délégué pour les besoins d'authentification et d'autorisation.

- **Cadre de contrôle d'accès (CCA)** : Il est défini comme un ensemble de composants logiciels qui gèrent les requêtes d'accès à une ressource donnée en analysant les informations de cette requête dans la Base de données (BD) des politiques. Les informations récupérées de cette requête rassemblent les attributs du demandeur (l'émetteur de la requête) ainsi que la ressource requise (cible). Ces informations seront traitées dans le CCA et une réponse sera émise par la suite. Pour illustrer l'architecture originale d'un CCA, nous décrivons les principaux composants impliqués dans la gestion de contrôle d'accès. Une requête est émise par le demandeur, qui est reçue par le composant *Récepteur*. La requête est transmise ensuite au composant *Analyseur* qui va vérifier les informations dans la base de données. Une réponse est générée enfin par le composant *Répondeur* qui envoie le résultat de la décision (e.g. permis, refus, indéterminé ou non applicable) au demandeur.

- **Module des politiques dynamiques** : Il définit notre approche pour supporter des décisions de politiques proactives. Notre approche consiste à étendre l'architecture du CCA avec des composants supplémentaires. Lorsque le *Récepteur* reçoit une requête, il enregistre cette requête dans une *BD de Requêtes*. L'*Adaptateur* de politiques interroge la BD de politiques pour voir si un nouvel évènement qui pourrait changer la décision précédente a été produit. Il procède en envoyant cette information au composant *Évaluateur* qui va comparer avec l'ancienne politique stockée dans la BD de requêtes. En cas de changement, la nouvelle politique sera renvoyée à l'*Analyseur* et une nouvelle réponse sera émise. Un nouvel évènement tel que *revoke* va entraîner une modification de l'ancienne politique de délégation et ainsi une mise à jour dans la BD des politiques d'autorisation sera faite. Au niveau utilisateurs, un service d'invocation via un point de contact va notifier le délégué de la nouvelle décision (voir figure 4).

6.3 Déploiement

Sur le plan architectural, les requêtes doivent être réévaluées à chaque changement de politiques. Pour cela, nous aurons besoin d'un mécanisme de stockage des requêtes précédentes ainsi que de leurs résultats. En effet, si la réévaluation d'une requête génère un résultat différent du premier résultat enregistré, le CCA doit informer le demandeur du nouveau résultat. Ainsi, il doit exister un mécanisme qui déclenche une nouvelle évaluation lorsqu'il détecte un changement de politique. Ces effets de changement de politiques seront capturés automatiquement et transmis au demandeur, dans ce cas le délégué, pour procéder aux mesures appropriées à ce changement. En outre, nous définissons un élément d'*Invocation* qui va acheminer cette information au délégué.

Sur le plan du langage, il faut définir de nouveaux constructeurs pour la description de la méthode d'invocation que le CCA utilisera pour contacter le demandeur. Il s'agit de développer un *point de contact* pour le demandeur. Dans une architecture orientée services

(SOA), ce point de contact peut être un point final d'un service ("service endpoint") qui pourra être invoqué par le système (voir le service web d'invocation dans figure 4). De ce fait, toutes les politiques d'accès doivent être centralisées et référencées par l'architecture SOA, qui sera protégée. Nous définissons un point d'accès unique et nous enregistrons les services web dans notre CCA.

Sur le plan technique, le *Gestionnaire de politiques* génère des politiques dans lesquelles il intègre des attributs d'authentification et d'autorisation [GSFC08]. Des fournisseurs d'authentification tels que les autorités de certification numérique délivrent des certificats au demandeur afin de traiter sa requête. À ce stade, le *Récepteur* agit comme un élément d'application de la politique ("Policy Enforcement Point") pour supporter la demande d'accès et contrôler la décision de la politique. Par exemple, un Certificat d'Attributs (CA) est délivré au délégué à des fins d'authentification et d'autorisation [CO02]. Un CA assure l'intégrité, la protection et la non-répudiation des informations échangées par l'intermédiaire d'une signature numérique. Le *Récepteur* obtient le certificat d'attributs du délégué et calcule ses permissions par la suite. Ces attributs seront validés par rapport à la politique (par exemple, le rôle du demandeur, la période de validité). Une fois que le délégué a été authentifié avec succès, il tentera d'effectuer des actions sur les ressources de la tâche spécifiée. À chaque tentative, le *Récepteur* transmet la demande d'accès à l'*Analyseur* pour décider. Les résultats de décisions (permis, refus, ou non applicable) seront alors envoyés via le *Répondeur*.

Une nouvelle réévaluation d'une nouvelle politique requiert de nouveaux CA pour des requêtes ultérieures par rapport aux changements de politiques. Par exemple, une révocation implique l'annulation du CA délivré précédemment pour le délégué. Actuellement, des techniques comme des certificats temporaires ou des listes de révocation de certificats sont basées sur la notion de temps, et par conséquent, ne répondent pas aux exigences des événements de délégation. Pour y remédier, nous proposons un environnement orienté services, où un service est appelé à prendre contact avec le délégué. En définissant un accord mutuel entre les deux instigateurs (le délégant et le délégué), des mesures appropriées seront prises suite au déploiement de ce service. Dans notre étude de cas, Bob sera amené à annuler son travail sur la tâche T3 en libérant les ressources et en fermant sa session d'accès.

Afin de contrôler le comportement de la délégation et de spécifier ses politiques d'autorisation, nous aurons besoin d'une technique qui automatise les politiques de délégation et qui permette par la suite d'accroître la conformité des changements dus à la délégation dans la politique d'autorisation existantes. Ceci sera abordé dans la section suivante.

7 Intégration des politiques de délégation basées sur des événements

Dans cette section, nous présentons une technique pour l'intégration des politiques de délégation dans les politiques existantes. Notre objectif est de raisonner sur les événements de délégation de façon dynamique. Pour ce faire, nous utilisons notre modèle de délégation de tâches (MDT) basé sur les événements. Afin de contrôler le comportement

de délégation et de spécifier ses politiques d'autorisation dynamique, nous recueillons les événements pertinents qui définissent à la fois le chemin d'exécution des tâches ainsi que les politiques générées pour la délégation de l'autorisation. En utilisant, le calcul événementiel, nous proposons une technique qui automatise les politiques de délégation et qui permet d'assurer la conformité des changements dus à la délégation dans la politique existante.

7.1 Le calcul événementiel

Cette section récapitule brièvement les fondements du calcul événementiel (plus de détails dans l'annexe) et présente quelques notions fondamentales. Historiquement, le calcul événementiel a été introduit par Kowalski et Sergot [KS89] comme formalisme logique de programmation pour présenter des événements et leurs effets, particulièrement dans les applications de base de données. Il est un cadre logique dans lequel il est possible d'inférer ce qui est vrai, à certains instants, étant donné un ensemble d'événements et leurs effets et permet de raisonner sur les effets d'actions réelles sur des états locaux.

Le calcul est basé sur des axiomes généraux concernant les notions d'événements, leurs propriétés et les périodes de temps où ces propriétés se vérifient. Les événements lancent et/ou terminent les périodes de temps où une propriété se vérifie. Pendant que les événements se produisent, les axiomes généraux permettent d'inférer de nouvelles propriétés qui sont jugées vraies dans le nouvel état modélisé, et impliquent l'arrêt d'autres propriétés qui ne sont plus jugées vraies. Le calcul événementiel, largement étudié formellement, est basé sur un modèle d'événements, d'états et de relations de cause à effet et intègre la persistance qui assure qu'une propriété persiste tant qu'un événement ne vient pas l'interrompre.

Prédicat	**Interprétation**
$happens(a, t1, t2)$ $happens(a, t)$	Le déclenchement de l'action a s'est produit entre les instants $t1$ et $t2$ Prédicat simplifié de $happens(a, t1, t2)$ qui se produit à t
$holdsAt(f, t)$	Le *fluent* f est vrai à à l'instant t
$initiates(a, f, t)$	Le *fluent* f devient vrai après l'action a à l'instant t
$terminates(a, f, t)$	Le *fluent* f devient faux après l'action e à l'instant t
$Initially_P(f)$	Le *fluent* f est vrai à partir de l'instant initial, c-à-d 0

Table 2: Prédicats du calcul événementiel

Ce formalisme est largement suffisant pour introduire les concepts de base de notre modèle de délégation de tâches. En effet, il fournit un cadre pour le raisonnement temporel en utilisant la logique des prédicats du premier ordre. Dans ce cadre il est possible de maintenir une représentation dynamique à un niveau d'abstraction élevé en se basant sur les axiomes fournis durant l'exécution d'une tâche déléguée. La formalisation utilisée pour représenter les actions et leurs effets est basée sur la logique du premier ordre. Comme pour tout langage du premier ordre, il faut commencer par choisir l'ontologie sous-jacente c'est-à-dire les symboles de prédicats, les fonctions et les symboles de fonctions. Les concepts

importants dans l'ontologie d'un calcul événementiel sont : les actions qui représentent les événements et les *fluents* qui sont des variables changeant au cours d'un intervalle de temps. En effet, nous utilisons les *fluents* pour décrire les différents états du modèle MDT (e.g. délégué, annulé).

Comme nous utilisons une version simplifiée du calcul événementiel, seuls les prédicats de base sont décrits ici (voir tableau 2). Les événements et les actions ne sont pas différenciés et les actions seront considérées de manière instantanée.

7.2 Modélisation de la délégation en utilisant le calcul événementiel

Les entités de base dans le modèle proposé sont les tâches. En calcul événementiel, elles peuvent être considérées comme des *sorts* dans lesquels des *fluents* sont crées. Un *sort* définit le type d'objet représentant la tâche déléguée. Ensuite, chaque tâche peut être dans des *états* (states) différents en cours d'exécution. Par exemples, les états *"assigned"*, *"delegate"* ou *"revoke"* changent avec le temps et peuvent donc être considérés comme des *fluents* dans le calcul événementiel.

sort task
fluent Initial(task), Assigned(task), Delegated(task), Started(task)...

event Create(task)
[task, time] Initiates(Create(task), Initial(task) ,time).
event Assign(task)
[task, time] Initiates(Assign(task), Assigned(task) ,time).
[task, time1] Happens(Assign(task), time1) $\rightarrow$ {*time2*} *HoldsAt(Initial(task), time2)*
& $time1 > time2$

Figure 5: Un calcul événementiel basé sur le MDT

Le calcul événementiel présenté ci-dessus, définit d'abord les *sorts* et *fluents* qui marquent les différents états de la délégation. De la même façon, nous pouvons définir les événements et leurs axiomes pour compléter le modèle MDT.

7.3 Construction des politiques de délégation

Dans cette section, nous analysons les besoins en sécurité dont auraient besoin les politiques de délégation. Nous utilisons le calcul événementiel pour implémenter une technique capable de générer des nouvelles règles d'autorisation automatiquement.

Une règle de politique peut comporter des conditions et des obligations qui sont utilisées pour identifier les différents cas dans lesquels une politique peut devenir applicable. Nous définissons une règle en tant qu'un tuple *(effect,condition,obligation)* dont le résultat est une décision de permission ou de refus. Notons qu'une obligation peut requérir une évidence pour valider l'exécution de la tâche [Sch07]. Une règle sera par la suite définie

dans la politique de délégation pour mettre à jour la politique existante (voir définition 4).

Definition 4 (Politique de Délégation) *Nous définissons une politique de délégation* $P_D \subset P$*, la politique d'autorisation et* $P_D = (target_D, rule_D, C_D)$*, avec* $target_D =$ DR *(la relation de délégation),* $rule_D \subseteq$ rule, $C_D \subset C$ *et* $C_D =$ DC $\bigcup$ events *avec DC l'ensemble des contraintes de délégation.*

Dans ce qui suit, nous analysons les exigences en sécurité pour le mode *push* et nous présentons un tableau qui récapitule les événements susceptibles de générer de nouvelles règles d'autorisation (voir tableau 3).

Les évènements de délégation	Le mode Push		Ajout de règles
	Grant	Transfer	
u1:delegate	✓	✓	Ajout de règles selon l'acceptation
u2:accept	✓	✓	Ajout de règles selon le type d'exécution
u1:cancel	✓	✓	Pas d'ajout
u2:execute/Grant	✓		(Permit,Push,Grant:Evidence)
u2:execute/Transfer		✓	(Permit,Push,Transfer:NoEvidence)
u1:validate	✓		Pas d'ajout
u1:revoke	✓		(Deny,Push,Grant)
u2:fail		✓	Pas d'ajout
u2:complete		✓	Pas d'ajout

Table 3: Politiques de délégation pour le mode Push

Revenons à l'exemple, nous pouvons observer une politique dynamique de délégation pour la tâche T3. Initialement, T3 est déléguée à l'*Assistant* u_2 et la politique de délégation pour T3 est : P_D = *(RD,Permit,{Push,5 jour})* (voir tableau 3/*u2:execute/Grant*). Pendant ce temps, le *Procureur* u_1 est de retour au travail avant que la délégation soit terminée. Le *Procureur* n'est pas satisfait de l'avancement des travaux et révoquera ce qui a été effectué par son *Assistant*. L'événement *revoke* sera mis à jour dans la politique, et une règle de refus est alors ajoutée. Ainsi, la politique de délégation pour T3 va générer une nouvelle règle :
P_D = *(RD,Deny,{Push,Grant})* (voir tableau 3/*u1:revoke*).

Notons que la règle de révocation générée dépend de la relation de délégation *RD*. Pour ce faire, nous avons déterminé l'octroi de droits d'accès basé sur le statut de la tâche en cours et les besoins de ses ressources en utilisant un modèle de contrôle d'accès basé sur les tâches (CAT) présenté dans la section précédente.

7.4 Modélisation des politiques de délégation en calcul événementiel

Afin de modéliser les politiques de délégation basées sur les événements, nous introduisons des nouvelles *sorts* appelées *effect*, *condition*, et *obligation* au calcul événementiel. Les

effets possibles *(effect)* seront *Deny* et *Permit*, et les conditions sont les modes *Push* et *Pull*. Les instances possibles pour une obligation seront *Grant, Transfer, Evidence* et *NoEvidence* relatif au modèle de délégation de tâche MDT. Nous avons également ajouté une action *AddRule(effect, condition, obligation)* comme événement pour supporter les changements de politiques.

Dans ce qui suit, nous présentons un algorithme pour calculer les règles de délégation basées sur les événements (voir algorithme 2).

Algorithm 2: Calcul des politiques de délégation de type Grant

Entrée : DPolicy // Delegation policy
sort // task, effect, condition, obligation
effect // Permit, Deny
fluent // AddRule(effect, condition, obligation)
event //AddPolicyRule(effect, condition, obligation)
time //time1 ≥ time2
while *(HoldsAt(AddRule(effect, condition, obligation)) AND obligation = grant)* **do**
  if *(Happens(PushDelegateAcceptExecuteGrant(task), time1) = true* **then**
    AddPolicyRule(Permit, Push, Evidence) / Autorisation de permission */*
  else
    if *(Happens(PushDelegateAcceptRevokeGrant(task), time1) = true* **then**
    AddPolicyRule(Deny, Push, Grant) / Annuler l'autorisation avec une règle de refus */*

Sorties : DPolicy = (AddPolicyRule(effect, condition, obligation), time1)

Les changements de politiques définis dans cet algorithme précisent que dès lors que certaines actions sont déclenchées suites à des événements tels que *"execute"* ou *"revoke"*, ils provoquent un changement de politique et ajoutent, en plus, une nouvelle règle dans la politique existante. Par exemple, *PushDelegateAcceptExecuteGrant* représente l'événement *"execute"* avec la permission *"grant"* une fois que la demande *PushDelegation* a été acceptée par un délégué (voir tableau 3).

Revenons à l'exemple. Une politique de délégation est définie pour la tâche T3. L'événement "u1:delegate" (avec u1 le *Procureur* Alice) va enclencher l'action *PushDelegateAcceptExecuteGrant(T3)*. Il s'agit d'une délégation en mode Push et de type Grant. Le résultat est *AddPolicyRule(Permit, Push, Evidence)* où l'évidence définit la période de délégation (5 jours). La politique de délégation pour T3 est ainsi :
P_D = *(RD,Permit,{Push,5 jour})*.

7.5 Automatisaton de la délégation

Notre approche basée sur les événements assure l'automatisation de la délégation. L'automatisation est nécessaire tant pour l'achèvement des tâches que la spécification de leurs politiques. En raisonnant sur les événements de délégation, nous offrons une solution capable de prévoir l'exécution de la délégation et d'accroître le contrôle et la conformité de la politique du workflow.

Pour ce faire, nous utilisons l'outil de calcul évènementiel DECReasoner comme outil de démonstration (voir annexe A). Notre raisonnement est basé sur le modèle de délégation de tâches MDT. Pour le raisonnement, un plan est défini pour l'objectif fixé. En référence à notre modèle, le but est d'avoir une tâche terminée, annulée ou échouée.

Nous ajoutons le but (goal) *[task] HoldsAt(Completed(task),15) | HoldsAt(Failed(task),15) | HoldsAt(Cancelled(task),15)* pour notre raisonneur de calcul évènementiel avec la valeur 15 représentant la largeur maximale d'exécution pour le "timepoint". En effet, une valeur inférieure risque de ne pas atteindre le but dans TDM, c.à.d. une tâche terminée, annulée ou échouée. L'invocation du raisonneur de calcul nous donnera toutes les solutions possibles (appelées plans) pour la réalisation de l'objectif. Le plan qui suit reprend l'exemple de motivation, où la tâche T3 est déléguée selon le mode Push (voir figure 6).

```
1389 variables and 7290 clauses
relsat solver
1 model
—
model 1:
0 Happens(Create(T3), 0).
1 +Initial(T3).
2 Happens(Assign(T3), 2).
3 +Assigned(T3).
4 Happens(PushDelegate(T3), 4).
5 +WaitingDelegation(T3).
6 Happens(PushDelegateAccept(T3), 6).
7 +WaitingCompletion(T3).
8 Happens(PushDelegateAcceptExecuteGrant(T3), 8).
Happens(AddPolicyRule(Permit, Push, Evidence), 8).
9 +RuleAdded(Permit, Push, Evidence).
+WaitingValidation(T3).
10 Happens(PushDelegateAcceptExecuteGrantValidate(T3), 10).
11 +Completed(T3).
—
;DECReasoner execution details
0 predicates, 0 functions, 12 fluents, 20 events, 90 axioms
encoding 0.5s - solution 0.2s - total 0.9s
```

Figure 6: Plan de délégation

Dans les changements de politiques deux scénarios sont possibles. Le premier scénario est l'intégration d'une nouvelle politique d'autorisation parce que les conjectures (conditions ou obligations) sont valables. Le deuxième scenario correspond à la modification de cette règle suite à un évènement déclencheur tel que la révocation (e.g. révocation de la tâche T3 déléguée à Bob par Alice). Le modèle de calcul évènementiel peut être enrichi afin de permettre des changements de politique minimaux. En effet, nous pouvons tirer parti d'une technique de vérification qui nous permet de choisir le meilleur candidat pour la délégation en se basant sur les performances de ce dernier. Les performances sont liées

à la réussite et l'accomplissement de la tâche et ainsi la réduction d'ajout de nouvelles règles dans la politique d'autorisation existante.

8 Conclusion et perspectives

Définir une politique de contrôle d'accès dynamique supportant les exigences de délégation est loin d'être une tâche triviale. Dans ce mémoire, nous avons présenté les problèmes et les exigences que demande un modèle de délégation de tâches. Nous avons également présenté à un niveau conceptuel, les différents éléments qui sont nécessaires pour la mise en oeuvre de telles politiques. La motivation de ce travail est inspiré de scenarios réels relatifs aux procédés e-gouvernementaux, où les besoins d'interactions humaines se font de plus en plus sentir. Nous considérons la délégation de tâches comme un support à la flexibilité organisationnelle dans des systèmes de workflow. Elle permet également d'assurer une forme de délégation des autorisations dans un système de contrôle d'accès. Pour ce faire, nous avons montré que les politiques de délégation peuvent changer selon des évènements spécifiques. Nous avons défini la nature de ces événements basés sur notre modèle de tâche MDT, et avons décrit leurs interactions avec les règles de décisions des politiques respectives. Lorsque des événements appropriés se produisent, nous définissons la façon dont ils sont capturés et traités. Dans ce contexte, nous avons proposé une extension aux systèmes de contrôle d'accès afin de permettre la spécification et le traitement des politiques de délégation dynamiques.

De plus, nous avons présenté les problèmes et les exigences d'intégration des politiques de délégation dans les systèmes de workflow. Pour ce faire, nous avons proposé une technique basée sur les événements afin d'assurer l'automatisation et la gestion des politiques de délégation. En effet, nous avons défini des évènement spécifiques à la délégation que nous avons intégré dans un modèle de calcul événementiel pour raisonner sur les règles d'autorisation générées suite à une délégation. L'automatisation est nécessaire tant pour l'accomplissement des tâches que par la spécification de leurs politiques. En raisonnant sur les événements de délégation nous offrons une solution capable de prévoir l'exécution de la délégation et d'accroître le contrôle et la conformité de la politique d'autorisation existante du workflow.

Pour la partie implémentation, nous nous sommes intéressés à la spécification des évènements de délégation au sein d'un standard de contrôle d'accès dans le langage XML. En plus, nous avons travaillé sur une extension du méta modèle en XACML. Cela nous servira de base, par la suite, pour communiquer avec les modules de réévaluation, d'adaptation et d'invocation orientés services.

Nos travaux futurs vont aussi dans la direction de la gestion des performances basée sur l'extraction de l'historique des délégation. En effet, la gestion d'un tel historique peut être une approche intéressante pour la vérification et l'audit.

Thesis Manuscript: A Secure Framework for Dynamic Task Delegation in Workflow Management Systems

Chapter 1

Introduction

1.1 Background and Motivation

The pace at which business is conducted has increased dramatically over recent years, forcing many companies to re-evaluate the efficiency of their business processes [Ven03]. In the classical software engineering approach, the organisational context and the related security requirements in the business process are often considered during the design phase, but internal controls for human-centric processes are later defined and implemented in a manual fashion disjoint from predefined models due to a semantic gap and a poor understanding of the organisational aspect [CSBE08, ZM04b].

Workflow management systems automate the management and coordination of organisational or business processes. Business processes automation requires the specification of process structures as well as the definition of resources involved in the execution of these processes. While the modeling of business processes and workflows is well researched, the link between the organisational elements and process activities is less well understood and does not consider the organisational aspect when considering heavily human-centric interactions in workflow systems. Moreover, the roots of contemporary workflow management solutions are oriented towards the automation of human-centric processes, thereby discounting the organisational aspect of workflow solutions with little or no human intervention.

Security is an essential and integral part of workflow management systems. Protecting application data in workflow systems through access control policies has been widely discussed. Sandhu *et al.* proposed a series of role-based access control (RBAC) models [SCFY96, BS00] and discussed a variety of constraints and authorisation policies including the role hierarchy and the separation of duties. The central idea of the RBAC model is that access rights are associated with roles, to which users are assigned in order to get appropriate authorisations. Within organisations, and thus in workflow applications, the concept of a hierarchy of participants/roles is prevalent. Participants are users who are placed in one or more units at different hierarchical levels. Therefore, it is important to discuss the differences between the proposed organisational unit hierarchy and their corresponding access control models to manage human resources dynamically within organisations. For that reason, access control mechanisms have to deal with such organisational flexibility

and to be more adaptable with dynamic environments to a certain extent.

In previous work [Sch07, GC], we observed a tendency moving away from strict enforcement approaches towards mechanisms supporting exceptions that are difficult to foresee when modelling a workflow. Actually, business processes execution are determined by a mix of ad-hoc as well as human-centric processes. This highly dynamic environment must be supported by mechanisms allowing flexibility, security and on-the-fly shift of rights and responsibilities both on a (atomic) task level and on a (global) process level [R4e06]. One specific set of mechanisms ensuring human-centric interactions is that of task delegation.

We define task delegation as a mechanism for assigning tasks and its access rights from one user to another user. The user who performs a delegation is referred to as a "delegator" and the user who receives a delegation is referred to as a "delegatee". Delegation is a mean to support human interactions within an organisation. It can be very useful for real-world situations where a user who is authorised to perform a task is either unavailable or too overloaded with work to successfully complete it. It is frequently the case that delaying these tasks executions will violate time constraints on the workflow, thereby impairing the entire workflow execution. Therefore, delegation is a suitable approach to handle such exceptions and to ensure alternative scenarios by making workflow management systems more flexible and secure.

To the best of our knowledge, most of the work done in the area of workflow and access control systems does not treat delegation in sufficient details and deserves more investigations. On one hand, existing work in the domain of organisational management in workflows remain static and lack of flexibility [AW05, CK08a]. On the other hand, current access control mechanisms are relatively stateless and rigid [SRS+05, BFA99]. At present, responses arising from access control requests are stateless such that a response is given to a particular request which is valid and true only at the time the request is made. If, however, this response changes due to a policy adaptation, no mechanism currently exists that allows to ensure dynamic delegation of authority. Such a mechanism is vital for supporting a secure framework for dynamic task delegation in workflow management systems.

1.2 Thesis Objectives

In this dissertation, we aim to address issues related to the organisational needs and security requirements for task delegation within workflow systems. Concretely, delegation works on ensuring flexible execution to support alternatives with unavailable workflow's actors to cope with the organisation rigidity. Since we aim to ensure a user-to-user delegation with heavily human interactions, we need to support additional requirements related to the organisation flexibility and the authorisation policy definition. In order to tackle these problems, we need to address two important issues, namely allowing task delegation to complete, and having a secure delegation within a workflow.

Allowing task delegation to complete requires a model that forms the basis of what can be analysed during the delegation process. Securing delegation implies the controlled propagation of authority during task execution. The monitoring of task delegation presents an essential step to ensure delegation completion. A delegated task goes through different

states to be terminated. States depends on generated events during delegation. Events such as *revoke* or *validate* are an integral part of the delegation process. Dealing with that, we define an event-based task delegation model that can fulfill all these requirements. Our model aspires to offer a full defined model supporting all kind of task delegation.

Moreover, we consider task delegation as an advanced security mechanism supporting dynamic policy enforcement. We define an approach to support dynamic delegation of authority within an access control framework. The novelty consists of reasoning on authorisation based on task delegation events, and specifying them in terms of delegation policies. When one of these events changes, our access policy decision may change implying dynamic delegation of authority. To do so, we propose a task delegation framework to support automated enforcement of delegation policies. In order to monitor the delegation process and to specify the authorisation policies in an automated manner, we gather specific events that will define both the task execution path and the generated policies. Dealing with that, we develop a technique that automates delegation policies using *event calculus* to control the delegation execution and to increase the compliance of all delegation changes in the authorisation policy.

1.3 Thesis Structure

The remainder of this dissertation is organised as follows. Chapter 2 presents the context and the problematic of this thesis. We introduce the context of the thesis related to the organisational management in workflow systems. In particular, we focus on the organisational needs and security functionalities supporting human-centric interactions in workflows. One type of mechanism supporting human interactions is that of task delegation. We then motivate our work with an e-governmental scenario supporting task delegation. From our motivating example, we identify the problematic and we present the major challenges and contributions of the thesis.

Chapter 3 outlines the state of the art of this thesis. We present fundamental concepts to the understanding of workflow management systems as well as the security in information systems. Our work is oriented access control requirements in business processes based on the workflow modelling. Dealing with that, we present existing work on workflow, business process and access control systems and we highlight their functionalities and limitations to support a secure framework for dynamic task delegation in workflow systems.

Chapter 4 presents the first part of our approach. We define a novel approach for modelling task delegation in workflow systems. We answer the interrogations defined in the problematic related to the motivation factors, the delegation model and the access control model using a detailed taxonomy for delegation. By leveraging the foundations described in chapter 3, we define a task delegation model (TDM) that not only meets the requirements at the organisational level, but also enables performing delegation in a secure and flexible manner at the security level. To do so, we define additional requirements related to the organisation flexibility such as user-to-user negotiation. Moreover, we propose a task oriented access control model (TAC) based on the RBAC model to address security requirements for delegation.

Chapter 5 presents the second part of our approach. It deals with the definition of a dynamic delegation of authority to support authorisation policies in access control systems. It is partly based on the insights gained in chapter 4. We answer the interrogations defined in chapter 2 related to the delegation of authority and the integration of delegation policies in existing access control systems. To that end, we analyse delegation events impact on authorisation policies. We present a framework ensuring dynamic delegation of authority and we show how proactive policy decisions will be implemented on existing access control frameworks. Finally, we propose a technique based on event calculus that gathers specific events and integrate delegation policies. We aim to control the delegation execution and to ensure the compliance of delegation changes in the existing authorisation policy.

In chapter 6, we implement the development environment for the delegation framework. We apply our assumptions, techniques and models defined in chapters 4 and 5 to test them and deploy them in a real development environment supporting delegation. We firstly focus on the organisational aspect in order to support human interactions in general and user-to-user delegation in particular. Dealing with that, we implement a delegation plugin supporting negotiation within a workflow runtime engine. Then, we consider the security requirements to support delegation policies. To do so, we develop a policy plugin within an access control system and we discuss its functionalities and limitations.

In chapter 7, we conclude and recapitulate the main contributions of our work. We outline also the limits that will be a part of our future works and discuss the future directions of our research.

Chapter 2

Context and Problematic

Contents

2.1 Introduction

Many of the complex day to day applications in large organisations are conducted using workflow management systems (WfMS). Workflow systems automate the management and coordination of organisational or business processes. Within organisations, and thus in workflow applications, the concept of a hierarchy of participants/roles is prevalent.

Participants are users who are placed in one or more units such as departments, divisions, or groups, and they have different bosses, at different hierarchical levels. It is important to discuss the differences between the proposed organisational unit hierarchy and their corresponding access control models to manage human resources within organisations.

The organisational context and the related security requirements are often considered during the design phase in the business process, but internal controls for human-centric processes are later defined and implemented in a manual fashion disjoint from predefined models due to a semantic gap and a poor understanding of the organisational aspect [CSBE08, ZM04b]. In this dissertation, we aim to address issues related to the organisational management with regards to user's assignment, task's definition and resource's access from design time to runtime in workflow systems. The main contribution is to bridge the gap between organisational needs and security functionalities supporting human interactions in workflows. This chapter is detailed as follows : we present the context of the thesis by introducing concepts related the organisational resources analysis, the strategy of work allocation in authorisation policies, the resources integration, their utilisation and their monitoring in workflows. Then, we present the thesis context which is motivated by an e-governmental scenario supporting delegation. Finally, we identify the problematic, we present the major challenges of the thesis and we conclude.

2.2 Context: Organisational Management in Workflow Systems

Traditionally, workflows have been used by business organisations for modeling and controlling the execution of business processes to achieve a business objective [WFM99]. In the context of a workflow, a process is composed of a number of activities which are connected in the form of a directed graph. An activity describes a piece of work that forms one logical step within a process. During execution, an activity instance includes tasks or services which are human implementations or computerised implementations of an activity [WFM99]. Here, our main concern is human activities (so-called tasks) where a human must be involved in performing manual activities.

Business processes automation requires the specification of process structures as well as the definition of resources involved in the execution of these processes. While the modeling of business processes and workflows is well researched, the link between the organisational elements and process activities is less well understood and does not consider the organisational aspect of workflow applications [Law97, RvdAHW06]. Organisational elements define workflow's resources in terms of tasks, users and data. It reflects human interactions within workflows. The importance of human involvement in workflow applications has recently been pointed out by several work which identified excessive activity automation and poor design of work assignment strategies as critical issues in workflow projects [Con02, Sch07].

Moreover, the roots of contemporary workflow management solutions are oriented towards the automation of human-centric processes, thereby discounting the organisational aspect of workflow solution and focusing exclusively on the coordination of control flow

structures. Proposed standards such as WSCL (Web Services Conversation Language) [Ari02] and BPEL4WS (Business Process Execution Language for Web Services) [Ton03] focused on the technical coordination of inter-enterprise processes, with little or no human intervention. In this chapter, we aim to give an overview of the organisational aspects to support human-centric interactions in the context of a workflow life cycle, and to develop guidelines for the design of a workflow-enabled organisation during the different phases of a workflow life cycle. A workflow-enabled organisation is a mean to manage organisational elements (e.g. participants) in workflow systems [ZM04a].

2.2.1 Resource management in the workflow life cycle

As related in the literature review, the workflow is made of tasks, where a task defines a unit of work that at each invocation performs the binding between different resources needed to complete a specific part of the workflow [RvdAHE05]. The resources that may be involved are different. We distinguish material and human resources for business objects and workflow actors, respectively. Generally, the manipulation of material resources is interfaced by one or several entities called applications or services.

A resource model contains the definition of human and material resources that are involved in the execution of a workflow model [Bal98]. While the resource model is a structured representation of organisational entities, it should be noted that both this model as well as the elements contained therein follow a life cycle and change over time. Therefore, a workflow management system not only needs to provide a mechanism to represent the organisational elements involved in the execution of workflows, but it also needs to provide mechanisms for continuous change within these elements.

The workflow life cycle reflects the desire to continuously improve the performance of business processes by monitoring the present, analysing the past, and planning for the future [WGHS99]. The workflow life cycle shown in figure 2.1 is built on the approach proposed by Zur Muehlen [ZM04a]. The black circles in figure 2.1 indicate the upcoming sections that refer to the resource management in the workflow life cycle (see sections : 2.2.2, 2.2.3, 2.2.4 and 2.2.5).

The cycle starts with a *goal specification and environmental analysis* phase that defines an initial analysis of the project goals and the organisational structures and rules surrounding the new system. This phase is followed by a *process design* phase, during which the overall process structure is engineered, the resulting workflow model is designed, and the resources involved in the process execution are specified. This includes the modeling of organisational structures as well as the definition of assignment policies and conflict resolution mechanisms.

The completed workflow models are input of the *process implementation* phase. During this phase, the workflow solution is integrated with surrounding information systems. In terms of resource management, access to existing resource databases and security mechanisms need to be established. The synchronisation of tasks responsibilities with application access rights is of paramount importance in this phase. If errors are made regarding this synchronisation, tasks might be assigned to users who have insufficient access rights to execute the applications necessary for the fulfillment of the pending task.

Parallel to the process enactment phase *process monitoring* takes place. On the technical side the performance of the workflow management system itself is measured, while on the organisational side metrics such as the length of work queues, the idle time of resources, or the wait time of pending tasks are supervised.

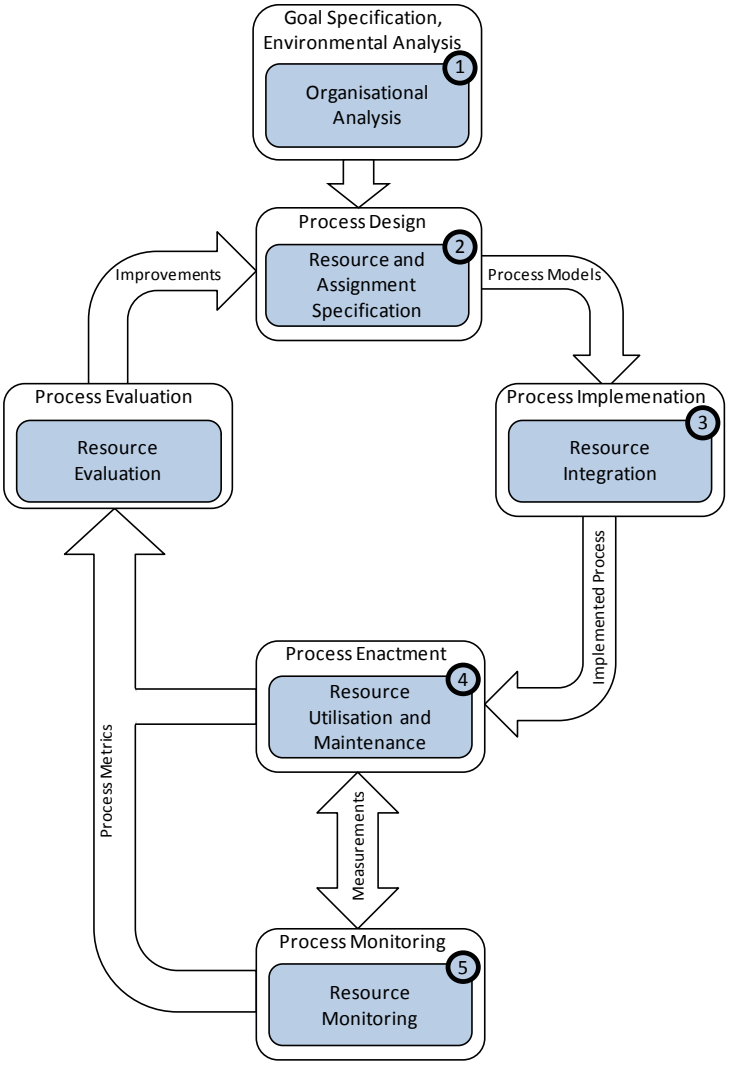

Figure 2.1: Workflow life cycle

During the *process enactment* phase, individual instances of the workflow model are created and coordinated by the workflow enactment service. The process participants which are part of the workflow-enabled organisation are notified of pending tasks through their worklists and can select and activate these tasks. Upon completion of a task, control is handed back to the workflow enactment service. Process participants may be human resources, material resources, or a combination of both. *The process evaluation* phase completes the workflow cycle. During this phase, the execution of workflow instances is analysed based on the execution protocols (the so-called audit trail) [ZM04a].

2.2.2 Organisational resources analysis

During the design time, the workflow application designer has to design both the structure of the business process to be automated, and the structure of the resources that carry out the process. Resources and workflow's tasks are linked through the construct role [CKO92]. From a process perspective, a role is a subject to authorisations that define permissions (operations) for the execution of a task. From a resource perspective, a role represents a granted authorisation for a workflow actor (so-called user). Based on these two perspectives, the design of the resource model can follow two different directions namely the material and human resources. Material resources define business objects and the way to use them. Human resources define the actors of the workflow.

From a material resource perspective, we define permissions as functions with operations to manipulate business objects. From a human resource perspective, we define a subject as an assigned user who is member of a role to claim a task instance. The task execution is added to the subject worklist. It defines the set of task instances claimed by this subject. The access to resources will be dependent on the execution model of the task. Figure 2.2 shows a meta model for task-based resource model, which analyses the possible ways the resources access can be defined during the task execution.

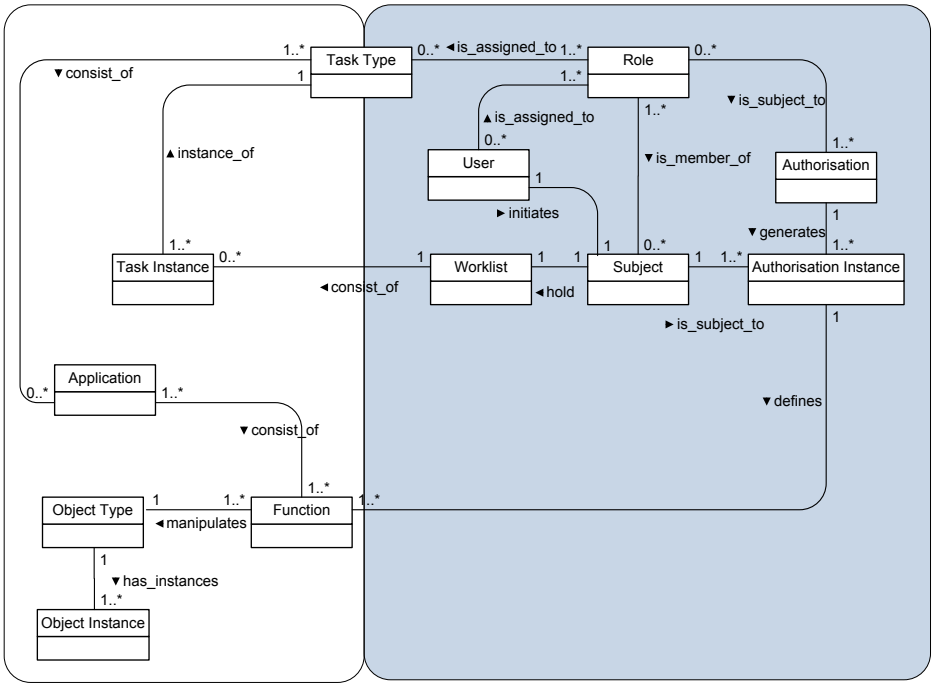

Figure 2.2: A model for task-based organisational structures

In figure 2.2, we define a task as a set of applications or services that are accessed by subjects via specific functions. These applications consist of functions that manipulate business objects. From one task several task instances can be generated. Note that we distinguish task type element from task since we assume that a task represents an instantiation of a task type during execution, equally for business objects. A task instance

corresponds to an actual execution of a task. This specific execution of the task (a task instance) is allocated to only one subject through its unique worklist, where a subject defines a user selecting a role during runtime.

We aim to address issues related to the organisational management in workflow systems with regards to user's assignments, task's definition and resource's access. We mainly focus on both material and human resources to analyse task requirements. We can identify from our analysis the main relationships between tasks, roles and resources :

- The set of *Tasks*, *Applications*, *Objects* and *Functions* defines the requirements that are necessary to carry out a task.
- The set of *Roles* assigned to a task.
- The set of *Authorisations* defines the condition of assignment that a role must have as permissions to execute a task.
- The set of *Subjects*, *Worklists*, *Task Instances* and *Authorisation Instances* defines the task execution requirements.

Based on this specification, we define the organisational resources analysis based on task requirements. As mentioned before, this will lead us to investigate the organisational needs as well as the security requirements for human tasks in workflows, thereby defining the strategy of work allocation, the resources integration, their utilisation and monitoring. Finally, it is important to note that there is no single ideal model for a workflow-enabled organisation. In workflow-based environments, the structure of the organisational entities as well as the structure of the business processes can be adjusted to find an optimal mix of resource and process efficiency [Mue99, ZM04a].

2.2.3 Definition of assignment and synchronisation policies

Assignment policies determine the strategy for work allocation among process participant candidates [ZM04a]. Candidates define users who may perform a task. A task corresponds to a single unit of work. Each executing task is termed a work item [RvdAHE05]. Upon the instantiation of a workflow task, the workflow enactment service places work items on the worklists of users who are determined using a process of role resolution (see figure 2.3). A pending task is placed on a shared worklist as a public work item. All qualified users have access to this shared worklist. Once a user (performer) chooses to perform the pending task (step 1 in figure 2.3), the workflow enactment service removes the public work item from the shared worklist and places it on the private worklist of the designated user (steps 2.a and 2.b in figure 2.3). This solution is based on the meta model for task-based organisational structure defined in figure 2.2. It allows the workflow enactment service to hold a master table of all pending work items, with access restrictions based on the roles required by the individual tasks and held by the users.

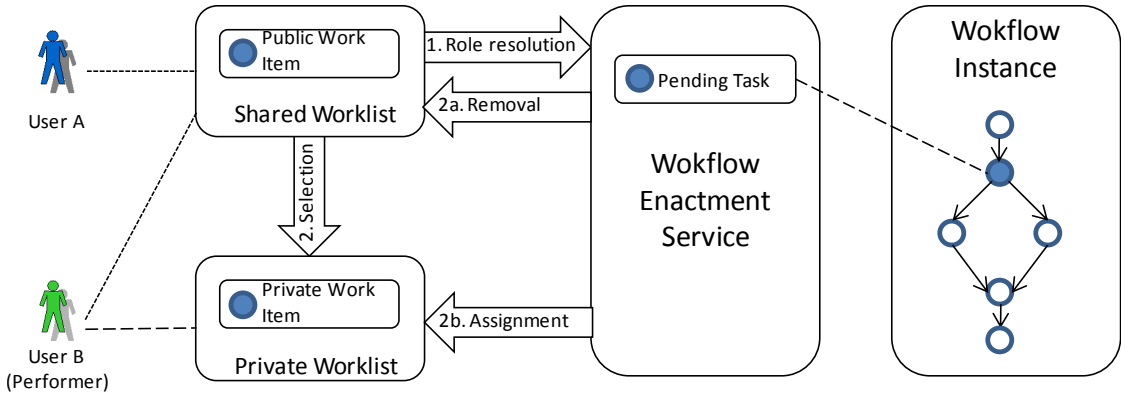

Figure 2.3: Role resolution procedure at runtime

For the assignment of pending tasks, different strategies can be implemented. These strategies have an impact on how the workflow enactment service prioritises tasks and notifies users performers. The assignment policies have different work allocation strategies, and are based on the research of Hoffmann and Zur Muehlen [ZM04a]. These strategies are identified based on task and scheduling properties. In the following, we identify the main properties for the assignment policies :

- **1. Task properties**

  ☐ Task execution : The form of task execution (individual or collaborative) prescribes, how many users may select a work item for execution.

  ☐ Decision hierarchy : It describes whether a task can be passed on from a workflow performer to another delegatee performer. This act of substitution is common during the absence of a resource [WKB07]. Task delegation challenges and issues are part of this dissertation and will be discussed in the upcoming sections.

- **2. Scheduling**

  ☐ The planning of new work items : It describes the behavior of the workflow enactment service, when new tasks become executable. For example, a net change strategy would only determine the assignment for the new work item, while a re-planning strategy would re-allocate all work items that have not yet been started.

  ☐ Time of notification : The workflow enactment service can notify performers about pending tasks either upon the availability of these tasks, at the latest start time, or at an arbitrary time.

- ☐ The queuing of work item : It can be performed either using a queue, ensuring that work items are selected in the order in which they become available; a pool, where resources can choose freely between available work items; or a combination of the two, where resources select a collection of work items at a time.

Once a work item has been placed on a shared worklist, synchronisation policies determine how it can be accessed by individual workflow participants [TH99]. In the following, we discuss properties related to assignment policies :

- User selection : It can be either direct or indirect. During direct assignment, the workflow system may check the availability of the selected users. In case of absence, a delegation may be supported by the workflow system for a new assignment or an exception is raised. Indirect selection is based on the set of roles that may be assigned to the task which allow the grouping of individual human resources (see figure 2.2).

- Assignment specification : It can be either static or dynamic. Using the static assignment, a task is associated with a set of predefined roles for workflow's actor. A dynamic assignment takes data from the current workflow instance into account for the selection of qualified performers (subjects in figure 2.2). This data can either relate to the current task instance or to the business objects processed in the current workflow instance.

- The assignment of pending task : It can be either performed in a push or a pull manner. Using the push mechanism, the workflow system determines the user who should perform the selected task. Using a pull mechanism, a user requests the next work item for execution.

The definition of assignment and synchronisation policies is a very important phase during the workflow deployment. We discussed issues related to resources hierarchy and their unavailability that may require delegation. Delegation is a mechanism that has to be taken into account during policies specification within workflow systems.

2.2.4 Resource integration

A workflow application coordinates tasks, users, data and applications. We define a task as a set of applications that are accessed by users, in this case performers, via specific functions on specific data (e.g. business objects). Consequently, all these elements need to be integrated to ensure the functionality of the workflow systems. *User integration* is required by the workflow system to keep track of the workflow's actors available for tasks assignment. *Data integration* is required to make relevant data accessible to the workflow system. This can be achieved by connecting the system to databases used by external application systems. *Application integration* describes the ability of the workflow

system to invoke external application systems during the enactment of a process [Mue99, BGMG02].

In addition, the use of existing security infrastructures is another important feature of workflow applications. *Security integration* relates to the use of existing authentication and authorisation mechanisms through the workflow system, such as single-sign-on, certification techniques and role-based access control mechanisms. Authentication and authorisation issues are important to support our secure delegation framework and will detailed in chapters 4 and 5.

2.2.5 Organisational maintenance at runtime

While the sequence of activities within a workflow definition is relatively stable, the resource model task-based needs to be constantly monitored at runtime to reflect the changes on the organisational level (see figure 2.2). Zur Muehlen [ZM04a] identified two categories of changes. It can be either macro changes at the entity level, or micro changes at the attribute level.

Changes at the macro level include the arrival and departure of members of the organisation, changes of the association between users and their roles, and the creation of new units within an organisation. If the resource information is stored within the workflow application itself, user accounts and their associated privileges need to be updated in the organisational policies. Changes at the micro level relate to changes in the authorisation profile of users such as the granting of additional authorisations, or the revocation of temporary privileges. This can be the case when delegating a task and so granting new privileges for the delegatee [GSFC08].

In addition to task assignment purposes, experience data is also useful during the handling of exceptions due to constraints such as workload, lack of human resources or deadlines that may overdue tasks. Instead of a hard-coded exception handling scheme, the workflow enactment service could assign an escalating task to a more experienced user or a substitute, if experience information is maintained in the system. Therefore, it is useful to leverage an experience-based auditing by taking additional parameters into account, such as the current workload of potential performers or their performances ranking to ensure task completion and to optimise workflow execution [Bar03].

2.2.6 Summary

In this section, we have identified the major aspects of resources management following the workflow life cycle. After a discussion of the human-centric interactions within a workflow and their involvement on both technical and organisational levels, we presented a task-based resource model which combines human and material resources for a task-oriented modelling within a workflow. Subsequently, we outlined variations in the assignment and synchronisation policies for the runtime allocation of tasks to performers, and discussed the maintenance of the different resources.

The definition of assignment and synchronisation policies is a very important phase during the workflow deployment. We discussed issues related to human resources hierarchy and unavailability. This defines exceptions on the organisational level, when initial

performers are not able to execute a task. We motivated delegation as a mechanism that has to be taken into account to support such exceptions. To the best of our knowledge, most of the work done in the area of workflow and access control systems does not treat delegation in sufficient details and deserves more investigations [AW05, CK08a]. We currently observe, when substituting members of the organisation, a move away from predefined strict workflow resources modelling towards approaches supporting flexibility on the organisational level. In addition, the assignment and synchronisation of policies have to consider the propagation of resource allocation and its authorisation during delegation.

2.3 Problem Statement : How to ensure a secure task delegation in workflow systems ?

The delegation of a task can be very useful for real-world situations where a user who is authorised to perform a task is either unavailable or too overloaded and so he will ask another user to act on his behalf. It is frequently the case that delaying these task executions will violate time constraints on the workflow, impairing the entire workflow execution. Hence, delegation is a suitable approach to handle such exceptions and to ensure alternative scenarios by making workflows more flexible and efficient.

Moreover, organisational flexibility depends on the rearrangement of the organisation members when delegating a task. We have to decide how to ensure the delegation of authority to a delegatee to access data's resources. The delegation of authority has to be defined in the access control systems. It expresses new delegation policies enforcement defined in a dynamic manner and integrated in the existing policy. To that end, we need to express task delegation requirements on both the organisational and the security levels of a workflow.

2.3.1 Motivating example : e-Government workflow scenario

Electronic government (e-Government) is creating a comfortable, transparent, and cheap interaction between government and citizens. e-Government is the civil and political conduct of government, including services provision, using information and communication technologies. The prerequisites for an e-Government enactment strategy are the achievement of a technological interoperability of platforms and a deeper cooperation and security at the organisational level using workflow management systems.

We introduce an e-Governmental workflow scenario related to the European administrations collaboration. Europol (European Police Office) and Eurojust (European Judicial Cooperation Unit) are two key elements of the European system of international collaboration within the areas of law enforcement and justice. A specific scenario for this collaboration is the Mutual Legal Assistance (MLA) [R4e06].

Mutual Legal Assistance (MLA) defines a collaborative workflow scenario involving national authorities of two European countries regarding the execution of measures for protection of a witness in a criminal proceeding. Here we describe the MLA process cross Eurojust organisations A and B, and detail the different business actors and resources models involved in the process. Basically, the work of the two organisations consists of

receiving the request of assistance from the Europol member in order to process it and send it the concerned authority in country B. Eurojust infrastructure integrates systems such as MLA service and CMS (Case Management System) to process data on the individual cases on which Eurojust national members are working in temporary work files (see figure 2.5).

Eurojust defines an organisational hierarchy working together to achieve common goals. Figure 4.2 illustrates the organisational role hierarchy and users role memberships in the Eurojust organisation.

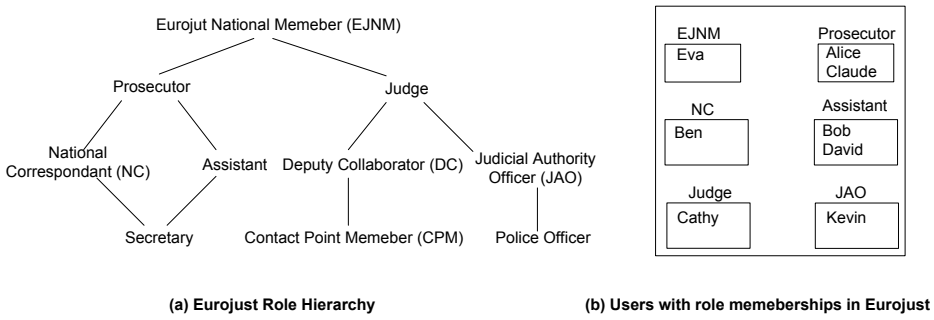

Figure 2.4: An example of organisational role hierarchy and users in Eurojust

Note that we define users Alice and Bob as members of Eurojust A with roles *Prosecutor* and *Assistant*, respectively. Claude, David, Cathy and Kevin members of Eurojust B with roles *Prosecutor, Assistant, Judge* and *JAO*, respectively.

We applied the Business Process Modeling Notation (BPMN) to the MLA process (see figure 2.5). BPMN has emerged as a standard notation for capturing business processes, especially at the level of domain analysis and systems design [OMG06].

In our example, we distinguish *Prosecutor* as the main responsible that collaborates with internal and external employees (Assistant, National Correspondent (NC), Judge and Judicial Authority Officer (JAO)) to process the MLA request. First, *Prosecutor A* receives the request and checks it in the MLA information service (tasks 1, 2 and 3). If the provided information are correct, the *Prosecutor* will continue to process the request by asking for the preparation of the request document by his assistant (task 4). Note that depending on the request context, the application process may differ in the data that needs to be considered. In fact, the specific type of legal document requested will have a direct effect on the involved controls. For instance, the "Translate Request Document" task might be required to carry out the request preparation when exchanged documents are issued in the local language; therefore we need to translate documents (task 3). After the preparation of the required legal documents, the *Prosecutor* will send the request to his Eurojust colleagues in country B (task 5). The next steps that need to be taken are the review of the request, the determination of the judicial authority in order to forward the request to the concerned authority in country B (Eurojust B) for the final approval (tasks 6, 7 and 8).

The supporting table 2.1 summarises the required roles, applications, functions and business objects associated to tasks.

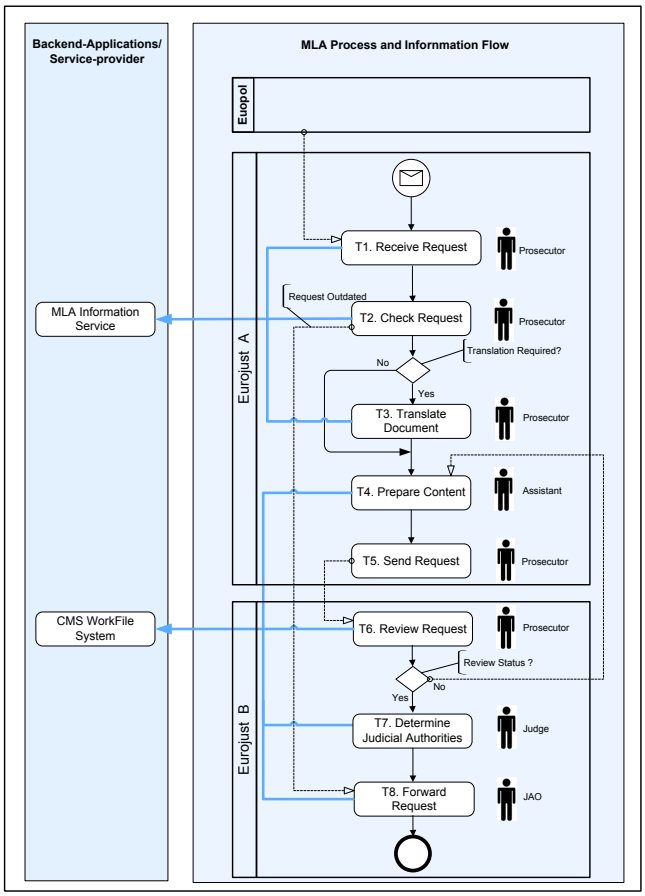

Figure 2.5: MLA scenario

Several of the depicted tasks involve human interactions and are possibly time consuming. Tasks composing the MLA process involve several business actors such as a *Prosecutor*, an *Assistant* or a *Judge*. Depending on the current control-flow sequence, workflow actors can evolve and change from the predefined workflow model. Actually, unexpected events can happen without being modelled beforehand in the workflow. For instance, a *Prosecutor* may delegate a part of his work to a subordinate due to an expected absence. Delegation is a suitable approach to handle such exceptions and to ensure alternative scenarios when deploying a workflow.

We define task delegation as a mechanism for assigning tasks and its access rights from one user to another user. The user who performs a delegation is referred to as a "delegator" and the user who receives a delegation is referred to as a "delegatee". Delegating access rights defines the propagation of authority (privileges) from the delegator to the delegatee. Delegated privileges define the required permissions to access task resources. For instance, functions (*query(), update(), translate()* in table 2.1) define permissions needed to access specific business objects to execute the different tasks of the MLA process. In the following, we present two delegation scenarios (DS1 and DS2) describing local and global delegation :

Task type	Role	Application	Function	Business Object type
T1. Receive Request	Prosecutor	MLA Information Service	read()	Request Document
T2. Check Request	Prosecutor	MLA Information Service	query(), update()	Request Document
T3. Translate Document	Prosecutor	MLA Information Service	translate()	Request Document
T4. Prepare Content	Assistant	CMS WorkFile System	add()	Request Document
T5. Send Request	Prosecutor	CMS WorkFile System	send()	Request File
T6. Review Request	Prosecutor	CMS WorkFile System	read(), update()	Request File
T7. Determine Judicial Authorities	Judge	CMS WorkFile System	add(), modify()	Request File
T8. Forward Request	JAO	CMS WorkFile System	send()	Request File

Table 2.1: Logistic workflow : Relations between tasks, roles, applications and business objects

Local delegation (DS1) : Our interest is related to the Eurojust organisation A. The involved actors are responsible for the reception of the request of assistance and the preparation of the required legal document to be sent to the concerned authority in country B. The role *Prosecutor* is a senior role and has at his disposal subordinates such as *NC* and *Assistant*. In this scenario, the task "Translate Documents" T3 is originally only accessible by *Prosecutor* Alice. Alice is unavailable to execute this task due to illness, and will delegate it to user Bob. Bob is a member of role *Assistant* and is a subordinate to *Prosecutor* in the organisation hierarchy based on the authorisation policy definition.

Global delegation (DS2) : It defines a delegation cross-organisations. We consider an instance of the process where the MLA request exists already in the CMS system of country B. The specialisation of *Prosecutor B* (user Claude member of role *Prosecutor*) will motivate the *Prosecutor* Alice to delegate T2 for his colleague. Task delegation is defined based on a role mapping (RM) cross organisations A and B, where distributed resources with external roles are defined in the policy.

2.3.2 Problem statements

Task delegation is a mechanism supporting organisational flexibility in human-centric workflow systems. Actually, when delegating a task we have to take into account all the

invariants involved in a workflow. Users, tasks and data represent such invariants. Organisational flexibility depends on the rearrangement of the organisation members (users) when reassigning a task to a delegatee. Moreover, we have to decide how to ensure the delegation of authority to a delegatee to access data's resources. The delegation of authority has to be defined in the access control systems. It expresses new delegation policies enforcement defined in a dynamic manner and integrated in the existing policy.

Typically, organisations establish a set of security policies, that regulate how the business process and resources should be managed. While a simple policy may specify which user can be assigned to execute a task, a complex policy may specify additional authorisation constraints supporting delegation. Any mechanism that is used to support task delegation is based on workflow specifications and user authorisations information. Delegation authorisation constraints are defined as events on the control-, data- and task assignment layers of a workflow [GSFC08]. Hence, the problematic is how to bridge the gap between workflow and access control systems to secure task delegation.

Based on the delegation scenarios (DS1 and DS2), we aim to address issues related to delegation requirements in workflow systems. On the *organisational level*, we have to identify the list of potential delegatees having the ability to execute the delegated task. In DS1, the delegatee Bob is a subordinate to the delegator Alice based on the role hierarchy definition of the Eurojust organisation. In DS2, it is a role mapping cross organisations. On the *security level*, we have to compute the required authorisation to execute the delegated task. In DS1, Bob inherits of the rights assigned to the role *Assistant*. In DS2, *Prosecutors* Alice and Claude have the same rights.

However, in DS1 the permissions to translate documents (*translate()*) in T3 are not given to the *Assistant* (see table 2.1). Bob will need an access grant to execute T3. This new access defines additional permissions for the delegatee and has to be updated in the existing policy. This update presents a delegation of authority that will enforce a new access control enforcement in the policy.

Moreover, the involved actors who are members of similar roles have to be distinguished on the *security level*. For instance, *Prosecutor* is a role defined in both organisations A and B. However, permissions given to *Prosecutor* Alice are different from those assigned to *Prosecutor* Claude for privacy reasons. To distinguish between both *Prosecutors* of Eurojust A and B on the security level, we need an additional mapping between workflow's actors and their security roles defined in access control systems to avoid authorisation conflicts..

We observe that delegation is a non-trivial task to model and engineer. Actually, it covers different entities related to the organisation (users), the process (tasks) and the resources access (data). In order to tackle these problems, we need to address two important issues, namely allowing task delegation to complete, and having a secure delegation within a workflow. Allowing task delegation to complete requires a model that forms the basis of what can be analysed during the delegation process. Secure delegation implies the controlled propagation of authority based on the workflow authorisation constraints related to the organisation policy and the resources access.

In addition, the delegation of authority defines an advanced security mechanism supporting authorisation policies. Delegation policies are defined from existing policies and are dynamically specified. However, existing work on access control systems remain state-

less and do not consider this perspective. To that end, we have to come up with an approach supporting automated enforcement for delegation policies. Additionally, the integration of such policies has to be computed and compliant with the existing policy based on delegation constraints. In order to control the delegation behaviour and to specify its authorisation policies in an automated manner, we have to monitor the task execution during delegation and to define specific rules to generate and integrate delegation policies in the predefined policy. Hence, the essence of our thesis is to answer the following interrogations :

1. What are the motivation factors for task delegation in a workflow and how to model it based on the factors requirements ?
2. How to monitor the delegation process with regards to workflow's layers : control (users), task and data (resources) in order to ensure the delegation completion ?
3. How to enrich delegation constraints with organisational and security requirements in the task delegation model based on both the task delegation model and the access control model ?
4. How to propagate the delegation of authority to delegate privileges when accessing task resources ?
5. How to specify and integrate delegation policies in existing access control systems in a secure and a compliant manner ?

2.4 Principles, Approach and Thesis Contributions

2.4.1 Principles

We present a delegation approach to support organisational flexibility in human-centric workflow systems, and to ensure delegation of authority in access control systems. Our objective is to ensure a secure task delegation within workflows. To that end, we have to cope with issues related to the organisational management as well as the security properties in workflow systems. These issues determine the main guiding principles detailed as follows :

- **Organisational flexibility :** The principle is the development of a methodology and a supporting framework for the management of organisational resources in the scope of workflow systems to support delegation. An organisation obtains flexibility by increasing the levels of internal and external flexibility available to it by increasing its ability to manage human resources. Depending on the current control flow sequence, workflow actors (users) can evolve and change from the predefined workflow model. In fact, unexpected situations where a user who is authorised to perform a task is either unavailable or too overloaded may require a new policy assignment. This can lead to a new rearrangement of users in order to optimise the process execution. Delegation intra and inter organisations can be very useful to ensure such flexibility.

- **Human-centric workflow :** The principle is to support heavily human-centric interactions in workflow systems. The requirements for interactions and monitoring can be summarised as transparency and control. Transparency addresses the revelation of the control flow dependencies. This allows to react accordingly to exceptions and compensations during execution. Control fosters the behaviour of organisations according to the defined policies. Existing solutions for workflows often appear to be lacking transparency and control supporting concepts and mechanisms. One type of transparency and control supporting mechanism in human-centric workflows is that of task delegation. We do believe that user-to-user delegation is a suitable approach to handle such interactions.

- **Access control enforcement :** Here, the principle is to give a coherent link from workflow modelling to access control requirements. As motivated previously, we aim to bridge the gap between organisational needs and security functionalities in workflows. Task delegation will enforce new authorisation requirements, thereby inquiring additional assignment and synchronisation for policies. Existing solutions for access control systems do not support such requirements for the delegation of authority. For instance, authorisation systems using role based-access control remain stateless and do not consider dynamic enforcement of policies. At present, responses arising from access control requests are stateless such that a response is given to a particular request which is valid and true only at the time the request is made. If, however, this response changes due to a delegation policy adaptation, no mechanism currently exists that allows the new response to be conveyed to the original requestor proactively. In addition, any policy change due to delegation has to be computed and integrated in the existing policy automatically with regards to the organisational policy compliance. For instance, a conflict of interest defines security constraints for the computing of delegatees and their privileges based on the policy specification. Such a mechanism is vital for supporting dynamic delegation of authority.

2.4.2 Our approach

The scope of our approach is to investigate the potential of delegation events to secure task delegation within a workflow. When delegating a task, often the reasoning behind this is dependent on transient conditions called *events*. Events define transitions ruling a task life cycle, where intermediate events such as *delegate*, *cancel* or *revoke* define a controlled delegation within a workflow. In addition, delegation has to be managed and executed in a secure way. Authorisation constraints are defined based on events with regards to the control-, data- and task assignment layers of a workflow, where delegation events may be a source to a policy change, thereby introducing advanced security requirements in access control systems.

Securing delegation involves the definition of authorisation policies which are compliant with the policy of the workflow. Therefore, these delegation events will imply appropriate authorisations on the delegatee side for further actions as well as contain specific constraints for those actions (e.g. deadline). In order to tackle these problems we need to address two important issues, namely allowing the delegation completion, and having a

secure delegation within a workflow. Allowing task delegation to complete requires the definition a delegation process. Secure delegation implies the controlled propagation of authority during task execution. Based on specific events, we define delegation policies in a dynamic and automatic manner. To that end, we introduce a delegation model that forms the basis of what can be analysed during the delegation process in terms of monitoring and security.

The monitoring of task delegation is an essential step to ensure delegation completion. A delegated task goes through different states to be terminated. States depends on generated events during the delegation life cycle. Events such as *revoke* or *cancel* are an integral part of the delegation behaviour. Revoking a delegated task may be necessary when a task is outdated or an access right is abused. Moreover, additional events such as *validate* may be required when a delegation request is issued under a certain obligation where the delegatee has to perform specific evidence to validate the task execution. Dealing with that, we came up with an event-based task delegation model (TDM) that can fulfill all these requirements. Our model aspires to offer a full defined model supporting all kind of task delegation for human centric-interactions.

Additionally, we define an approach to support dynamic delegation of authority within an access control framework. The novelty consists of reasoning on authorisation based on task delegation events, and specifying them in terms of delegation policies. When one of these events changes, our access policy decision may change implying dynamic delegation of authority and specifying them in terms of delegation policies. Existing work on access control systems remain stateless and do not consider such perspective. We propose a task delegation framework to support dynamic enforcement of delegation policies and discuss its integration into existing access control systems.

Finally, we leverage delegation constraints to automate delegation policies from existing policy specifications. With an automated mechanism, when the policy changes to reflect delegation, the delegation policy will be derived automatically based on specific facts related to the delegation constraints. Accordingly, it is not possible to foresee a deny rule for revocation during the policy definition. Moreover, a manual review of the current access control rights and task executions is costly, labor intensive, and prone to errors. In order to control the delegation behaviour and to specify its authorisation policies in an automated manner, we gather specific events that will define both the task execution path and the generated policies for the delegation of authority. Using *Event Calculus*, we present a technique to monitor the delegation execution. Afterwards, we explain how generated policies change dynamically in response to delegation events.

2.4.3 Contributions

The main contribution of this doctoral thesis is the development of a methodology with a supporting framework for the modelling and specification of human-centric interactions as an internal control in the scope of businesses processes in order to ensure a secure delegation within workflows. We present a generic access control approach for security policies within workflows based on the process modelling specifications. The approach encompasses formal verification of access control requirements and the proper termination of task delegation processes by the means of transient conditions (so-called events). To ease

the monitoring of task delegation and the propagation of delegation authority, we present an event-based task delegation model (TDM) amenable for supporting a framework to model, analyse and generate delegation policies. Therefore, the main contributions of this thesis can be summarised as follows :

- A **delegation model** supporting access control in business processes based on the task lifecycle. After the analysis of organisational management in workflow systems including a model for task-based organisational structures, internal control in terms of security requirements are defined. Using a multi-layered state machine within a workflow, we identify delegation interactions within workflow's layers namely task, control and data based on delegation events. Delegation events define the delegation process and build the task delegation model to be integrated in the business process. This model will guide us to define an access control model supporting delegation.
- Delegation as an **advanced security mechanism** is an enhanced access control model used to capture task requirements and to express it in terms of authorisation policy rules. The access control model is an extension of role-based access control model (RBAC) and is linked to the task assignment requirements. We define it as a Task-oriented Access Control (TAC) model. TAC provides new modelling elements to enable the specification of access control requirements such as organisational roles, resources requirements between users and tasks to restrict access to sensitive data, to delegate privileges among users and to avoid potential conflict of interests at runtime. In addition, we present a technique to optimise delegated privilege computation based on the delegated task instance defined in the TAC model.
- We apply **formal methods** to integrate delegation policies and to detect compliance properties with the authorisation policy. We utilise the Discrete Event Calculus Reasoner (DECReasoner) for performing automated commonsense reasoning using event calculus, a comprehensive and highly usable logic-based formalism. By reasoning on delegation events, we are able to address the policy stateless issue. We can compute delegation policies from triggered events during task execution. Policy automation offers many benefits. Actually, it reduces efforts for users and administrators. Administrator efforts can be related to the process definition and the policy specification. Moreover, it increases control and compliance of all delegation changes. Subsequently, a delegation request is accomplished under constraints which are compliant to the existing policy.
- Finally, we present an **access control framework** to specify delegation policies. Using PERMIS a policy based authorisation system, we develop an extension supporting task delegation requirements during policies specifications. The developed tool is used to specify and integrate delegation polices into existing access control systems.

2.4.4 Published results

The main contributions results have been published in the following papers. Published results are either directly relevant or intermediate based on or in reference to the research

findings presented in this thesis.

- Khaled Gaaloul, Ehtesham Zahoor, François Charoy, and Claude Godart. Dynamic Authorisation Policies for Event-based Task Delegation. *The 22nd International Conference on Advanced Information Systems Engineering (CAiSE'10)*, Hammamet, Tunisia, June 09-11, 2010.

- Khaled Gaaloul, François Charoy. Task Delegation Based Access Control Models for Workflow Systems. *The 9th IFIP WG 6.1 Conference on e-Business, e-Services and e-Society, I3E 2009*, Nancy, France, September 23-25, 2009.

- Walid Gaaloul, Khaled Gaaloul, Sami Bhiri, Armin Haller, Manfred Hauswirth. Log-based transactional workflow mining. *Journal of Distributed and Parallel Databases*, 25(3): 193-240, 2009.

- Khaled Gaaloul, Philip Miseldine, and François Charoy. Towards Proactive Policies supporting Event-based Task Delegation. *The Third International Conference on Emerging Security Information, Systems and Technologies, SECURWARE 2009*, Athens, Greece, June 2009.

- Khaled Gaaloul, François Charoy. Une Approche Dymanique pour la Gestion des Politiques de Délégation dans les Systèmes de Contrôle d´Accès. *27ème Congrès Informatique des Organisations et Systèmes d'Information et de Décision, INFORSID 2009*, Toulouse, France, May 2009.

- Khaled Gaaloul, François Charoy, and Andreas Schaad. Modelling Task Delegation for Human-Centric eGovernment Workflows. *The 10th International Digital Government Research Conference, dg.o 2009*, Puebla, Mexico, May 2009.

- Khaled Gaaloul, Andreas Schaad, Ulrich Flegel and François Charoy. A Secure Task Delegation Model for Workflows. *The 2nd International Conference on Emerging Security Information Systems and Technologies, SECURWARE 2008*, Cap Esterel, France, August 2008.

- Khaled Gaaloul, François Charoy, Andreas Schaad, and Hannah Lee. Collaboration for Human-Centric eGovernment Workflows. *The 8th International Conference on Web Information Systems Engineering, WISE Workshops 2007*, Nancy, France, December 2007.

2.5 Conclusion

In the context of heavily human-centric workflows, business processes are determined by a mix of ad-hoc as well as process-based interactions. This highly dynamic environment must be supported by mechanisms allowing the monitoring, secure and on-the-fly shift of rights with respect to an ongoing human interactions both on a (atomic) task level and on a (global) process level. One specific approach is that of task delegation. In this dissertation, we aim to address **the modelling and mapping of access rights to task**

delegation issue. This will lead us to the design and the implementation of a secure framework of dynamic task delegation within workflow applications.

Task delegation is a mechanism that supports organisational flexibility, and ensures delegation of authority in access control systems. To that end, we present a literature review on delegation concepts as well as concrete supporting technologies in both workflow and access control systems in chapter 3. We discuss their functionalities and limitations with regards to our approach requirements in terms of **organisational flexibility and dynamic access control over task delegation**.

The rest of the dissertation is detailed as follows. We identify delegation requirements motivated by the delegation scenarios presented previously. We classify these alongside a defined taxonomy of motivation factors for delegation with respect to the organisation, the process and the security constraints. Basically, we aim to separate the various aspects of delegation with regards to workflow's actors, task and resources (see chapter 4).

Additionally, we provide an analysis regarding the development of a methodology and a supporting framework for the modelling and specification of human-centric interactions as an internal control in the scope of businesses processes to ensure a secure delegation within workflows. Dealing with that, we leverage constraints properties **to build and integrate delegation authorisation policies dynamically** in existing access control systems to ensure compliancy (see chapter 5).

Chapter 3

State of the Art

Contents

3.1 Introduction

The pace at which business is conducted has increased dramatically in recent years. Much of this can be attributed to the Internet and the emergence of e-Government applications. Most governmental organisations offer electronic services to citizens supporting processes which are determined by a mix of ad-hoc as well as process-based human interactions. Workflows aim to model and control the execution of business processes cross organisations. Thus, it is evident that workflow technology has a great influence on the business operations of an organisation [WFM99].

Moreover, organisations establish a set of security policies, that regulate how the business process and resources should be managed [AW05]. One of the major problems with workflows is that they often use heterogeneous and distributed applications to execute a given workflow. This gives rise to security policies and mechanisms that need to be managed. Since security is an essential and integral part of workflows, workflow systems have to be managed and executed in a secure way. In particular, additional mechanisms are needed to allow controlled access of data objects, secure execution of tasks, and efficient management and administration of security [HK03].

This chapter introduces concepts fundamental to an understanding of workflow management systems, processes modelling as well as the security concepts related to them. Our work is oriented access control requirements in business processes based on the workflow modelling. Dealing with that, we present existing work on workflow, business process and access control systems (sections 3.2 and 3.3) and we highlight their functionalities and limitations to support a secure framework for dynamic task delegation in workflow systems (sections 3.4 and 3.5).

3.2 Business Processes and Workflows

Many of the complex day to day applications in large organisations are conducted using workflow management systems. Workflow systems automate the management and coordination of business processes. Business processes are descriptions of routine activities in an organisational environment which achieve goals that contribute to the overall goal of the organisation. A workflow is "the automation of a business process, in whole or in part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules" [WFM99].

3.2.1 Workflow management systems

The Workflow Management Coalition (WfMC) defines a workflow management system (WFMS) as a "system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with the workflow participants and, where required, invoke the use of IT tools and applications" [WFM99]. The Workflow Management Coalition offers a reference model for workflow management systems. It covers the concepts, terminology, general structure of WfMS, their major functional components and

the interfaces and information exchanges between them. The contents of this section are, to a large extent, based on this document [WFM99].

Some of the terms that are fundamental to an understanding of workflows are the concepts of a business process, activity, process instance and activity instance. Figure 3.1 is adapted from the Workflow Management Coalition [WFM99]. It provides a conceptual overview of how the terms discussed below relate to the concept of workflows.

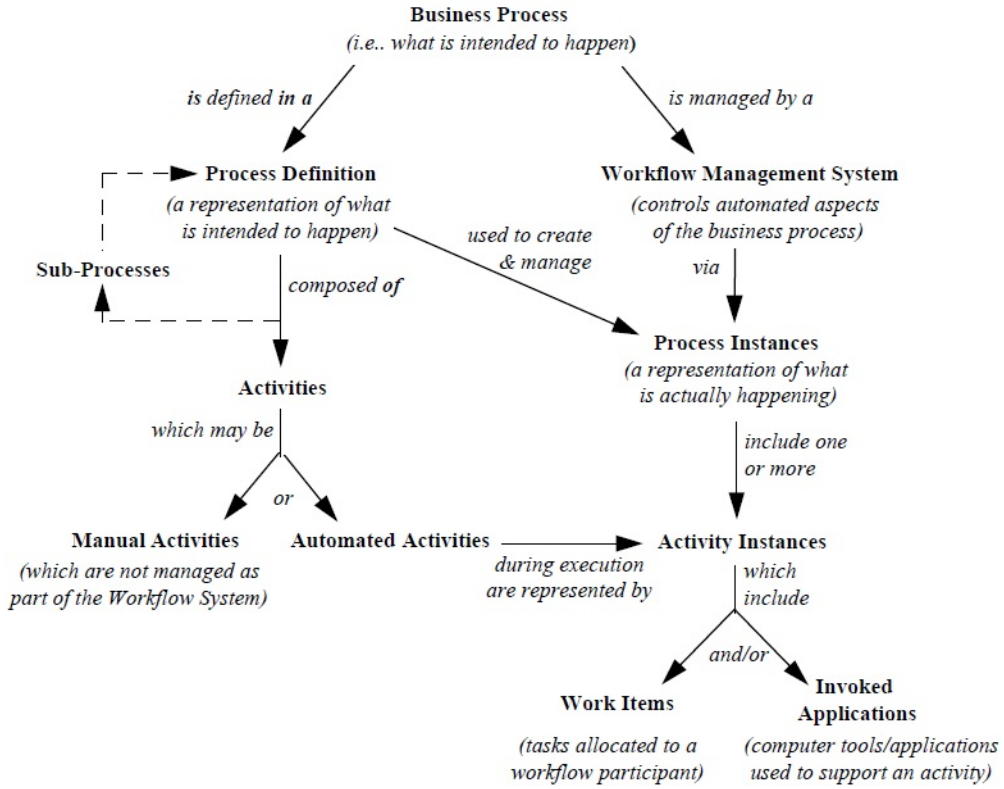

Figure 3.1: Workflow terminology

A business process depicts the association of activities that, when executed in a systematic way, ultimately achieve some goals or objectives. The relationships between these activities and the objectives they achieve are typically defined according to organisational policies and structures.

The Workflow Management Coalition [WFM99] defines an activity as a "description of a piece of work that forms one logical step within a process". Activities generally represent the smallest unit of work in the workflow. Activities are classified as manual or automated. Both types are represented in the process definition.

- Manual activities : They define activities that are not, or cannot be automated. They are tasks that are executed by humans or constitute manual work. Human tasks are included in the process definition but they are not capable of computer automation.

- Automated activities : They can be automated and managed by a WFMS. Automated activities are commonly referred to as "workflow activities". The execution of activities in a workflow process contributes to the achievement of some goal or business objective where the order in which activities are executed is important depending on the process definition.

Each execution of a workflow system creates a workflow instance. A workflow instance represents a certain case of the workflow process. Each process may contain multiple "cases" of an activity, which are termed "activity instances" or "task instances". **The term "task" will be used interchangeably throughout the rest of this dissertation to refer to workflow activities supporting human interactions.** In the motivation example, human activities are defined in the different delegation scenarios DS1 and DS2.

The Workflow Management Coalition (WfMC) has published a widely cited "Generic Workflow Product Structure". It has been referenced in numerous papers. We show that model in figure 3.2 in simplified form and restricted to the runtime view.

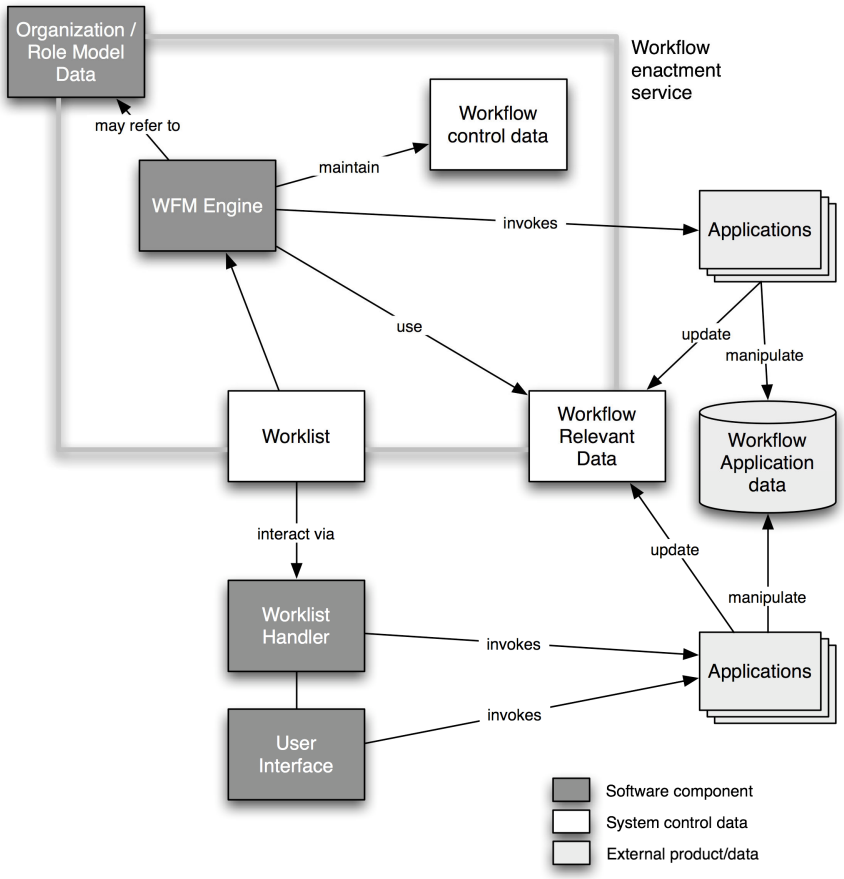

Figure 3.2: The generic architecture of a WfMS

This generic model shows the central component to be the workflow engine, which maintains associations with several data repositories. User interaction goes through a worklist handler and the interface to the engine is the worklist. Applications are accessed directly from the user interface and the worklist handler. They manipulate the same data repository as any processes invoked out of the workflow engine.

3.2.2 Organisational model in WfMS

Organisational model represents organisational entities and their relationships; it may also incorporate a variety of attributes associated with the entities, such as skills or role. The model normally incorporates concepts such as hierarchy, authority, responsibilities and attributes associated with an organisational role. It may be referenced by a workflow management system as part of the mechanism by which process role is established [WFM99].

Organisation model is part of the process definition within a workflow. It allows workflow participants to be specified in terms of attributes (roles) contained within the organisational model. During process execution the WfMS can obtain details of participants matching the attributes from the organisational model (see figure 3.3).

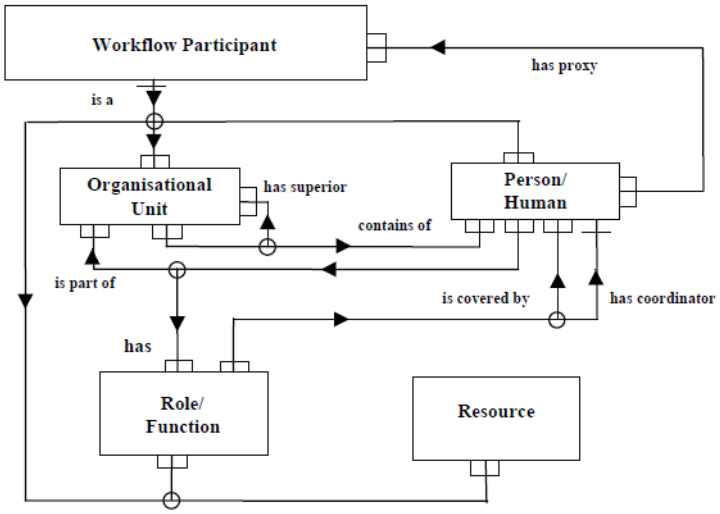

Figure 3.3: Example of an organisational model

A workflow participant assumes a role to access and process work from a workflow management system. The role defines the context in which the user participates in a particular process or activity. The role often embraces organisational concepts such as structure and relationships, responsibility or authority.

The organisation model is important to support human-centric interaction in workflows. Especially during delegation, we need to check the attributes of both the delegator and the delegatee within the organisation. It is a mean to confront them with the organisational policy to access and process work from workflow management systems (see section 3.3).

3.2.3 Business process management vs. Workflows

Business process management (BPM) is a collection of planning, organising, and controlling activities of an organisation's value chain with respect to a specific business goal of a process-oriented organisation [Dav93, ZM04b, GHS95]. A company can reach these business goals only, when people and information systems work well together on these business processes. Therefore, business process management encompasses methods, techniques, and software to design, enact, control, and analyse a companies' operations involving humans, organisations, applications, documents and other sources of information including the continuous and incremental improvement of an organisation's processes [Wes07].

In essence, the management activities related to business processes can be idealistically arranged in a life cycle [AHW03, KLL09]. It represents an ideal situation of sequential stages in engineering a process. As shown in figure 3.4 the life cycle comprises of activities such as survey, design, implementation, enactment, and evaluation [KLL09, Wes07].

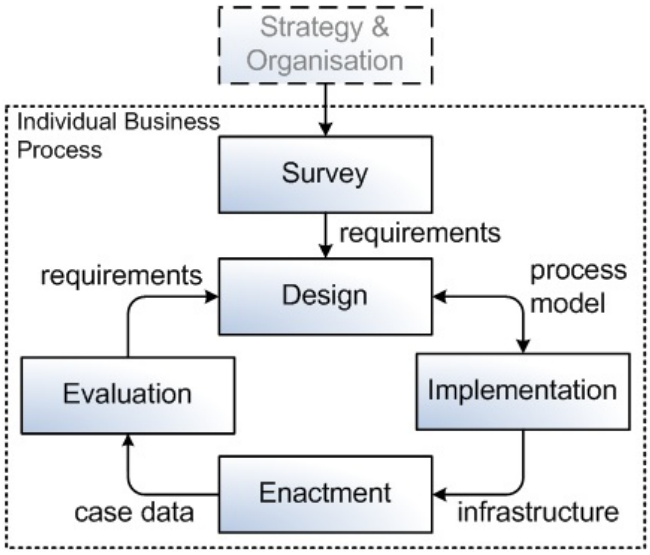

Figure 3.4: Business process life cycle

From the existing literature, the terminologies of BPM and WfMS may have different viewpoints. Business process management (BPM) is a process-oriented management discipline. It is not a technology. Workflow is a flow management technology found in business process management suites (BPMSs) and other product categories [KLL09]. However in reality, to our best knowledge, many BPMs are still very much WfMS [AHW03, KLL09, GHS95]. In addition, the workflow life cycle reflects the business process life cycle. It reflects the desire to continuously improve the performance of business processes by monitoring the present, analysing the past, and planning for the future [ZM04a].

3.2.4 Business process modelling

Process modelling performed as part of the design phase in the business process life cycle is an approach for visually depicting how businesses conduct their operations by defining the entities, activities, enablers and further relationships along control flows [AHW03]. Process visualisation is a core element within business process design, and this is often achieved with a series of as-is and to-be process modelling tasks. It is employed to create an abstraction of an otherwise complex business goal and realising business activities by defining the coordination of such activities [Wes07].

a - Modelling abstraction

To reduce the complexity, business process modelling provides different level of abstractions [Wes07]. Horizontal abstraction allows to start the process design with the definition of high level business goals. These goals are broken down to value chains that are then further refined into business processes and eventually enriched with technical service references understandable by technical experts and information systems for automated execution. This refinement is typically done in several steps involving different stakeholders, ranging from high level business analysts and process experts to IT specialists.

Referring to figure3.5, a process model's granularity depends on the level of horizontal abstraction nevertheless on a single horizontal abstraction level different concerns may be of interest. Vertical abstraction provides specific perspectives on a process model depending on the point of interest of the modeler and potential stakeholders. Figure 3.5 describes an horizontal and vertical modelling abstraction of a gas processing scenario defined in [Wes07] :

- The control-flow perspective (or process) perspective describes activities (tasks) and their execution ordering through different constructors, which permit flow of execution control.
- The data perspective deals with business and processing data. Business documents and other objects which flow between tasks, and local variables of the process, qualify in effect pre- and post-conditions of tasks execution.
- The resource perspective provides an organisational structure anchor to the process model in the form of involved humans and device roles responsible for executing tasks.
- The operational perspective describes the elementary actions executed by tasks, where the actions map into underlying applications. Typically, (references to) business and process data are passed into and out of applications such as Web Services, allowing data manipulation within applications.

The level of abstraction is also needed in the workflow life cycle where the process modelling is performed as part of the design phase. It reflects a conceptual and logical level with regards to the aspects of task, control and data layers within workflows. Workflow layers depends on a set of invariants for workflows from the aspects of users, tasks and

data. Workflow's layers can be portrayed in terms of a multilayered state machine [HK03]. A multi-layered state machine can enable the analysis, simulation and validation of the WfMS under study before proceeding to implementation. In addition, **it can serve as a powerful tool for modelling our task delegation framework at a conceptual and logical level with regards to the aspects of task, control and data** [GSFC08].

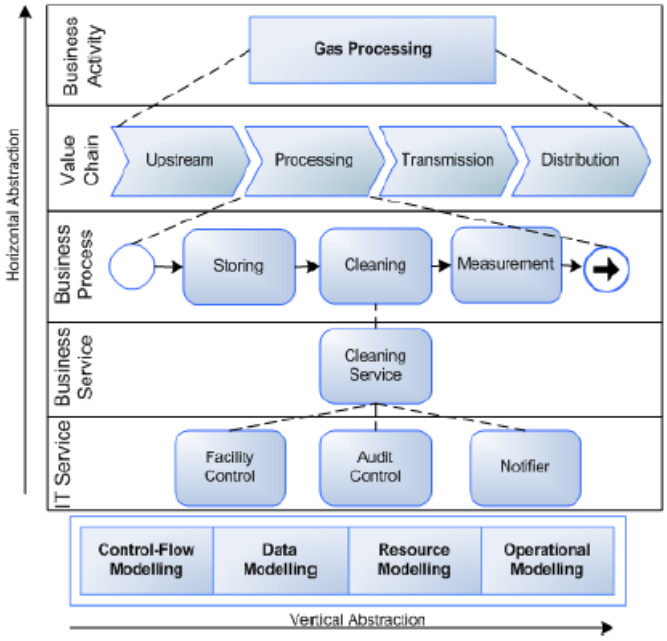

Figure 3.5: Horizontal and vertical modelling abstraction

Different modelling concepts emerged over time, with different strengths and weaknesses, such as the trade-off between expressibility and model analysis complexity, typically due to the variety of their underlying formalisms. Some techniques and notations offer richer syntax sufficient to express most relevant business activities and their relationships in the process model, while some provide more generic modelling constructs which facilitate efficient process model analysis [KLL09].

There are two predominant formalisms on which process modeling is based, namely graph-based formalisms and rule-based formalisms [SS93]. A graph-based modeling language has its root in graph theory, which provides rich mathematical properties for the syntax and semantics and theoretical support, while a rule-based modeling language is based on formal logic and are competitive to the graph-based approach in terms of mathematical soundness, model robustness and verification techniques [SGN].

In the next section, we present an overview of formalisms and notations supporting business process modelling based on graphical standards.

b - Graphical standards

A multitude of graph-based notations emerged in the past decades, such as Flowcharts, Flow-Block diagrams, Petri nets, IDEFs, UML Activity diagrams, EPCs or BPMN, many of them tailored for a specific modeling purpose or target audience [AHW03, Sch92, KLL09].

Unified Modelling Language (UML) Activity Diagrams

The Object Management Group (OMG) presented the Unified Modelling Language (UML) (version 2.0) [OMG04], standardised in 2004, which is the backbone of the object oriented software engineering computing paradigm. Broadly speaking, UML is a suite of 13 object-oriented notations that captures all attributes and behaviour of the objects modelled. A few examples of these notations include the Use Case Diagram (for documenting high-level user requirements), the Sequence Diagram (for documenting program sequence), and the Activity Diagram, etc. Of the two, the Activity Diagram (AD) is most commonly used to model business processes in a graphical way [Hav05]. The UML AD is both a flowcharting technique and a special kind of state machine whose activities are states and interactivity links trigger-less transitions.

The suitability of UML as a business process modelling technique was assessed by Russell *et al.* [RvdAHW06]. He concluded that UML AD offers comprehensive support for the control-flow and data perspectives allowing the majority of the constructs encountered when analysing these perspectives to be directly captured. However, UML ADs are extremely limited in modelling resource-related or organisational aspects of business processes. These limitations are common to many other business process modelling formalisms and reflect the overwhelming emphasis that has been placed on the control-flow and data perspectives in contemporary modelling notations [RvdAHW06].

In our work, we need a formalism that support resource-related and organisational aspects of task delegation in business processes. For instance, time constraints for resources need to be specified when delegating a task (see delegation scenario DS1). In addition, we need specific notations to sperate the involved principals during delegation (e.g. swimlane). For that reason, our choice to model task delegation within workflows is based on the Business Process Modeling Notation (BPMN).

Business Process Modelling Notation (BPMN)

The Business Process Modeling Notation (BPMN) is a standardised business process notation which is defined and specified by the Object Management Group (OMG) [OMG07] and has become the de facto standard for graphical process modeling. The notation inherits and combines elements from a number of other proposed notations for business process modeling, including the XML Process Definition Language (XPDL) [WFM05] and the Activity Diagrams component of the Unified Modeling Notation (UML). Figure 3.6 depicts an excerpt of the BPMN meta-model.

For the sake of simplicity we omitted some elements that are not relevant in the course of this chapter. BPMN process models are composed of flow objects such as routing gateways, events, and activity nodes. Activities, commonly referred to as tasks,

represent items of work performed by software systems or humans, i.e. human activities. Activities are assigned to pools and lanes expressing organisational institutions, roles and role hierarchies.

Routing gateways and events capture the flow of control between activities. Control flow elements connect activity nodes by means of a flow relation in almost arbitrary ways. BPMN supports so called artifacts that enrich the process model by information entities that do not affect the underlying control flow and are a dedicated extension points to add additional information to the model. In our work, we provided an extension for security semantics in BPMN by adding authorisation constraints for task delegation.

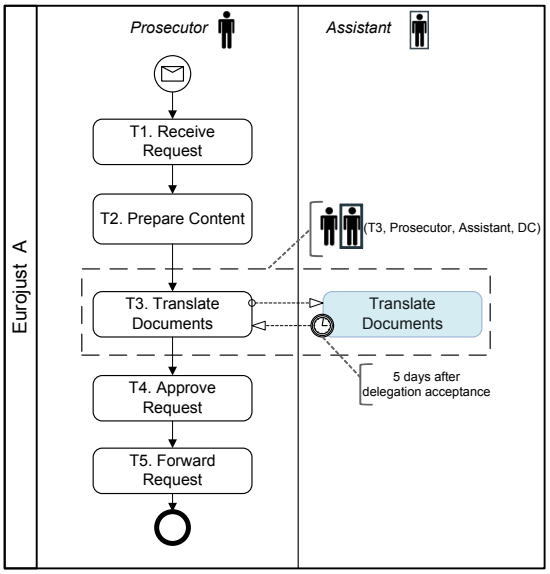

Figure 3.6: BPMN example for DS1

3.2.5 Summary

This section introduced concepts that are fundamental to the understanding of workflow systems. The content was based largely on the definitions and specifications developed by the WfMC in several of its published documents. It is clear that workflow technology has a significant impact on the operations of business processes and business in general.

Securing the information and procedures associated with workflows is an area of increasing concern. It can almost be said that the success of a workflow implementation depends largely on its secure execution. This is particularly true when we consider the need to manage authorisation rights when dealing with sensitive data, information or procedures during delegation. Delegating a task may inquire security requirements to propagate authorisation within workflows. In the following sections, we give an overview of security concepts and we discuss their integration in in workflow systems.

3.3 An Overview of Security Concepts

Nowadays, organisations define control goals with respect to information security to secure and conduct their business activities [Sch03]. Information security became an organisational control goal that may be achieved by confidentiality, integrity or availability providing mechanisms, techniques and processes purposefully designed in order to control the business activities of individuals, groups, and whole organisations.

Information security constitutes the maintenance of five fundamental security properties, which results in the establishment of reliable systems that meet the needs of an organisation. These properties, as defined in [IPS89], are *identification authentication, authorisation, confidentiality, integrity* and *non-repudiation*. Satisfying these properties ensures that management can make informed decisions that are based on accurate information. This enables them to act in accordance with the intentions of the business, and thus gain a competitive edge in the external marketplace [Ven03]. **Authorisation is most relevant to this dissertation, but an overview of all five properties will be given to illustrate the role that access control plays in supporting security properties.**

3.3.1 The five pillars of information security

a - Identification and Authentication

Firstly, systems must promote the principles of identification and authentication. This involves identifying legal users to the system as well as ensuring that they are who they claim to be. Identification concerns the manner in which a user provides his unique identity to the IT system. The identity must be unique so that the system can distinguish among different users. Authentication is the process of associating an individual with his unique identity, that is, the manner in which the individual establishes the validity of his claimed identity [FIS02].

The WfMC defines authentication as "the process by which a computer system or a (human) system user unambiguously identifies themselves to another computer system, normally in the context of gaining access to various services which the authenticated party is authorised to use on that computer system" [WFM01]. It is rendered by defining user identifications as well as implementing authentication mechanisms. Authentication parameters typically come in three forms : something the user knows (PIN or password), something the user possesses (token or memory card) and something genetically unique to the user (fingerprint or retina scan) [IPS89].

b - Access Control (Authorisation)

Controlling access, in one form or another, is considered by most information systems security experts to be a cornerstone to achieve information security. Access control, which may be physical, technical, or administrative, is a mechanism to provide information security. A given information system can implement access control systems in many places and at different levels. Operating systems use access control to protect files and directories while database management systems apply access control to regulate access

to tables and views. Access control policies describe the ways in which information is managed and company assets are protected by mediation of access requests of principals or other information systems [FIS02].

The WfMC makes a distinction between the terms "authorisation" and "access control". Authorisation is defined as "the process of identifying to the computer system the various functions which a user (human and potentially a computer system) may undertake" [WFM01]. Access control is defined as "the mechanism by which users are permitted access to various operations or data within a computer system, according to their identity (established by authentication) and associated privileges (established by authorisation)". However, the terms access control and authorisation will be used interchangeably throughout this dissertation.

c - Confidentiality

Confidentiality models are used to describe what actions must be taken to ensure the confidentiality of information. It is concerned with protecting information from being illegally retrieved by unauthorised people. Confidentiality is implemented using encryption mechanisms which execute mathematical algorithms on plaintext data, thereby rendering it as unintelligible ciphertext to illegal users [PP06].

The most commonly used model for describing the enforcement of confidentiality is the Bell-LaPadula model [LB73]. It defines the relationships between resources and subjects. The relationships are described in terms of the subjects assigned level of access or privilege and the objects level of sensitivity. Formally, the Bell-LaPadula model consists of a set of subjects S, a set of resources R and a set of security levels L. The function $F_{conf} : S \cup R \rightarrow L$ maps each subject $s \in S$ and object $r \in R$ to a security level $l \in L$, where $L, <$ is a lattice. As a preventive model, Bell-LaPadula defines two rules that must be enforced by an access control system at anytime :

- **No-Read-Up :** A subject $s_i \in S$ is granted read access to resource $r_i \in R$ if : $F_{conf}(r_i) \leq F_{conf}(s_i)$
- **No-Write-Down :** A subject $s_i \in S$ is granted write access to resource $r_i \in R$ if : $F_{conf}(s_i) \leq F_{conf}(r_i)$

For example, a subject s_i which has a clearance level *Secret* should not be allowed to read a data resource r_i with a classification of *Top Secret*. Conversely, a subject s_j is holding the clearance level *Secret* should not be allowed to write to a data resource r_j which is rated *Confidential*. The first case represents a *no-read-up* requirement, while the latter is called a *no-write-down* requirement.

In a workflow context, it is important to ensure that the confidentiality of information is sustained. If it is not, it would mean that access control mechanisms have been undermined. An elaboration of these principles is beyond the scope of this dissertation.

d - Integrity

Integrity is the protection of system data from intentional or accidental unauthorised changes. The challenge is to ensure that data is maintained in the state that subjects expect. Although it cannot improve the accuracy of data that is put into the system by subjects, it can help to ensure that any changes are intended and correctly applied. It is imperative, therefore, that no subject be able to modify data in a way that might corrupt or lose assets or render decision-making information unreliable. Several integrity models are discussed in the literature suggesting different approaches to achieve integrity such as Biba [Bib75], Clark-Wilson [CW87] or Brewer-Nash [Har04].

As an example, we have a look at the Biba integrity model, which has been the first model to address integrity within computer systems based on a hierarchical lattice of integrity levels. It comprises a set of subjects S, a set of resources R, and dedicated integrity levels I. The integrity levels form a partial order $\leq \subseteq IxI$. The function $F_{int} : S \cup R \rightarrow I$ returns the integrity level of a subject. In general, an access control system adhering to the Biba model ensures the following two policies :

- **Simple Integrity Policy :** A subject $s_i \in S$ can read a resource $r_i \in R$ if : $F_{in}(s_i) \geq F_{conf}(s_i)$
- **Integrity Confinement Policy :** if a subject $s_i \in S$ has read access to resource $r_i \in R$ if : $F_{conf}(s_i) \leq F_{conf}(r_i)$ with integrity level $F_{in}(r_i)$, then s_i can have write access to resource s_i if and only if $F_{in}(r_i) \geq F_{in}(r_j)$

In the context of a workflow, the maintenance of object integrity is important. Access control contributes to the enforcement of object integrity by limiting user access to certain objects. If, however, integrity is compromised through the unauthorised interception of information, the integrity of the entire workflow will be compromised.

e - Non-repudiation (Non-denial)

Non-repudiation mechanisms require each party involved in any transaction to have in their possession a secret digital signature that uniquely identifies them, thereby serving as the equivalent of an analogue signature. Digital certificates verify the identity of the sender, place a tamper-resistant seal on a message, and provide proof that a transaction has occurred [Ven03]. Digital certificates give the Internet a high level of certainty, much the way a passport or driver's license verifies a person's identity. These digital signatures are incorporated with asymmetric (public key) encryption to enforce non-denial security.

In a workflow context, enforcement of non-denial is particularly important when tasks have financial implications such as e-Government applications. We will discuss this issue and present some techniques supporting non-repudiation mechanisms for authentication purposes in this dissertation.

3.3.2 Access control approaches for security policies

A security policy defines the expected standard of security enforcement using access control within an organisation. Primarily, a security policy addresses who has access to what resources, as well as how this access is to be regulated and managed. Pfleeger *et al.* defines an effective security policy as one that is durable so that it can adapt well to changing circumstances, useful in that it can be understood and followed by everyone to whom it applies and realistic in terms of its specification of realisable security goals [PP06].

A security policy document is significant in that it makes employees aware of the organisation's position, thereby safeguarding the organisation from accidental employee security breaches. Furthermore, it defines a standard according to which retribution of intentional offences may be assessed. The following approaches can be used to define access control in an organisational security policy. They are, discretionary access control (DAC), mandatory access control (MAC), and role-based access control (RBAC) in both static and dynamic organisational contexts.

a - Discretionary Access Control (DAC)

The DAC approach allows the creator of an object, or anyone else that is authorised to control it, to make access control decisions. These rights change dynamically at the discretion of the owner of an object [PP06]. Access control mechanisms that support a DAC policy include directory lists, access control lists and access control matrices.

Directory lists maintain a directory for each user (subject) in the system. Each directory entry contains fields of the object and associated access rights that a user has. A pointer to each object is also included. A disadvantage of this approach is that management complications arise in the event that an object is removed or renamed. Maintenance is difficult when updates are required in the directory of many users [PP06].

b - Mandatory Access Control (MAC)

A MAC approach requires that access control decisions have to be made beyond the control of the owner of an object. These access control decisions are determined by a central authority, and users have no authority to change them [PP06]. The Bell and Lapadula Model supports MAC policies [LB73].

The Bell and Lapadula Model deals with access rights such as read or write. It also labels subjects and objects according to a predefined security ranking. This model is slightly more flexible within organisations [LB73]. The Bell and Lapadula Model allows for a flow of information between multiple levels in an organisational hierarchy. Read access is granted to subjects with higher security classifications than the objects accessed, and subjects with lower security levels have write access to objects of higher security classifications. This is not practical for real-world workflow situations where classifications may need to be changed periodically [Ven03].

Because of the rigidity of both DAC and MAC models, it is difficult to incorporate delegation in them. For this reason, the focus will be on role-based access control (RBAC), which can be seen as a combination of both MAC and DAC approaches.

c - Introduction to the Role-Based Access Control (RBAC) model

In most organisations, a security policy must be applied to hundreds, if not thousands, of employees. To simplify security administration, many organisations define roles with which multiple individuals can be associated. The security policy of the organisation then defines how permissions are to be associated with these roles. Sandhu *et al.* presented the RBAC approach which is particularly effective when changes are made to the organisational security policy. The RBAC model needs only to be made to role assignments, which are significantly fewer than individual assignments [SCFY96].

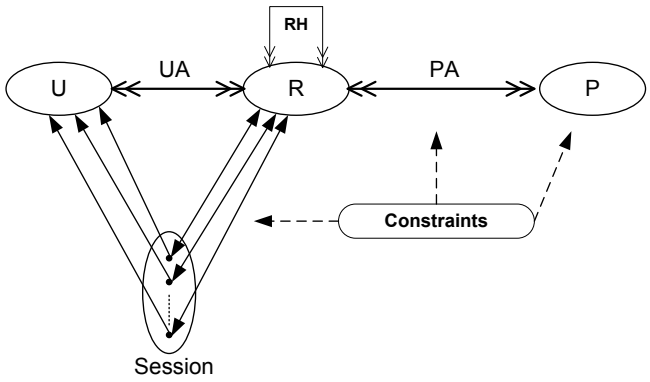

Figure 3.7: RBAC model

RBAC elements include sets of five basic elements : users U, roles R, permissions P, sessions, and constraints C. The fundamental definition is that individual users are assigned to roles and permissions are assigned to roles. A role is a means for naming many-to-many relationships among individual users and permissions.

A user in this model is a human being, a role is a job function or a job title, and permission is an approval of executing an object method (access to one or more objects, or privileges to carry out a particular task). A session is a mapping between a user and possibly many roles. A session is always associated with a single user (so-called a subject) and each user may establish zero or more sessions.

RBAC has two relations : user assignment (UA) and permission assignment (PA). The user assignment is a many-to-many relation between users and roles. The permission assignment is a many-to-many relation between permissions and roles. Users are authorised to use the permissions of roles to which they are assigned. RBAC additionally provides the notion of role activation for the duration of one session. RBAC is also an effective access control approach to use in workflow systems. Hence, due to the effectiveness of this approach and its direct relevance to this dissertation, an in-depth discussion of RBAC will be given in chapter 4, with direct references to Sandhu *et al.* model [SCFY96].

d - Static organisational context

An access control model defines which subjects are permitted to perform a certain operation on a resource. A subject defines a user selecting a role during runtime. Referring

to Figure 3.8 subjects are assigned to one or more roles. Permissions define associations to perform an operation on a resource. They are associated with a role by a permission assignment. Thus, the use of resources is restricted to individuals authorised to assume the associated role.

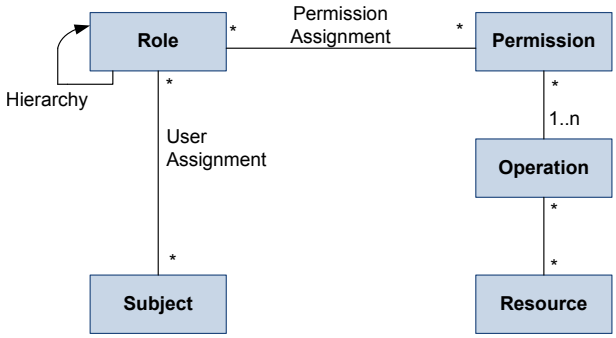

Figure 3.8: Static organisational context for RBAC

In its simplest form the role-based access control model comprises a set of roles R, a set of subjects S, a set of permissions P, a set of permissions assigned to roles $PR \subseteq PxR$, a set of subjects assigned to roles $SR \subseteq SxR$, and a partially ordered role hierarchy $RH \subseteq RxR$. A role-based access control system must ensure that a subject $s_i \in S$ has the permission $p_i \in P$ to perform an operation on an object $o_i \in O$ only if :

- **Role-Permission Assignment :**
  $Rp_i = \{r_i \in R : \exists p_i, r_i \in PRp_i\}$
- **Subject-Role Assignment :**
  $Sp_i = \{s_i \in S : \exists s_i, r_i \in SR, r_i \in R\}$

In other words $R(p_i)$ is the set of roles holding the permission p_i. Thus, $R(p_i)$ is the set of users authorised to perform p_i.

e - Dynamic organisational context

The conflicting entities paradigm introduced by Botha *et al.* [BE01] postulates that within an organisation the risk of security incidents increases if associations with specific entities are not carefully controlled and monitored by an access control management system. Botha differentiates between four different types of conflicts, such as conflicting subjects, conflicting roles, conflicting objects and conflicting permissions. Conflicting subjects refer to individuals in an organisation that are likely to conspire due to a personal relationship and together may share information or gain permissions that imply too much executive power.

Similarly, conflicting roles and conflicting permissions are defined that, if assigned to a single individual, may increase the likelihood of fraud. Conflicting objects relate to the Chinese Wall principle [BN89], which is based on information or knowledge that is mutual exclusive to be acquired by an individual.

This notion of conflicting entities is rather strict and applied to an organisational context would require a very fine-grained distribution of access control permissions within an organisation, hardly reflecting the original organisational structures and hierarchies. For instance, we consider our motivating example in which a *Prosecutor* must act as a pre processor or a post processor for the MLA request. In the organisations (Eurojust A and B) exists only one kind of a *Prosecutor*, while the access control management system must differentiate between two kinds of mutual exclusive *Prosecutors* types which may not be assigned to the same performer (see delegation scenario DS2).

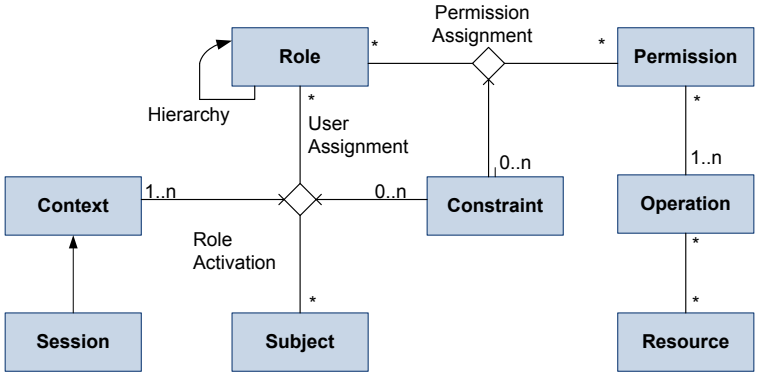

Figure 3.9: RBAC Sessions

As discussed by Sandhu *et al.*, the role-based access control model may be further extended by sessions, which may be considered as a context as well to translate static role-based separation of duty (SoD) constraints into a role activation constraint, which is commonly referred to as a dynamic separation of duty constraint (see figure 3.9).

Crampton *et al.* first introduced the notion of entailment constraints, which restrict permissions depending on contextual information, such as a subject's recent activities performed, data accessed or roles activated [Cra05]. Dynamic organisational context defines additional constraints that have to be considered during delegation. We leverage specifics constraints for tasks such as SoD to see whether a delegatee can claim a task based on his role specifications in the organisational context (see chapter 5).

f - Others access control models

Task-based access control (TBAC) aims to provide task context to permission assignments. A workflow system consisting of tasks is assumed. Each of these tasks is then assigned a "protection state", providing information as to who gets to have which permission on a task basis. According to the current state of the workflow system moving through the process instance, different permission assignments are activated or deactivated as ordered by the protection state. The TBAC design is process oriented, offering time-related features such as expiration and validity as well as instance-based characteristics [TS98].

Team based access control (TMAC) is an access control scheme similar to RBAC, but it provides user context and object context. Thus, it allows for more fine-grained specification of permissions on individual users in a team context, as well as isolated

addressing of specific object instances relevant to tasks. The main contribution of TMAC is the assignment of both users and (object access) permissions to teams. Each team (instance) then is bound to the task it was created for. At runtime, more than one team can be created out of the same template, but each team will be working on a different task instance and accordingly will need access to different object instances [Tho97].

TBAC and TMAC were defined to support specific contexts which are out of the scope of this dissertation. In our work, we use and extend the RBAC model since it is more suitable for our approach. RBAC offers the possibility to support human interactions in terms of user-to-user delegation. In addition, we leverage RBAC features (role-permission relation) to extend it with task-permission relation in order to support task delegation requirements (see chapter 4).

3.3.3 XACML : a policy language

We understand policies as persistent declarative specifications, derived from management goals, defining choices in the behaviour of a system. Policy-based approaches to system management allow the separation of the rules that govern the behaviour of a system from the functionality provided by that system [Sch03].

There are currently numerous choices of policy languages available. Most of them can be categorised as one of the languages based on logic, XML, or credential. Here, we will look into the eXtensible Access Control Markup Language (XACML) policy specification language. XACML is a declarative language based on XML for expressing security policies. It is an open standard, ratified by the Organisation for the Advancement of Structured Information Standards (OASIS) in 2005[1].

Beyond specifying the representation format of security policies, this standard also defines the interpretation of those in a generic processing model. It also addresses the issue of today's distributed system architectures which are comprised of many system entities, each of which has specific security requirements associated with multiple points of enforcement, by proposing a modular and distributed system setup for its enforcement architecture.

Currently XACML can be found in many publications in the area of research[2]. The different research facilities paying attention to XACML can be classified as the big players in the field of information technology, like IBM, Sun Microsystems, Oracle Corporation, Cisco Systems, and SAP amongst others. Thus it is evident, that XACML is gaining increasing attention and importance in the area of information security.

The major components making up the proposed architecture for XACML enforcement environment, and its data flows are shown in Figure 3.10. The following sequence introduces those components by tracking the flow of information between them.

1. At first the policies managed though the Policy Administration Point (PAP) are made available to the Policy Decision Point (PDP). The PDP is the component

[1]For more information on the work of OASIS : http://www.oasis-open.org.

[2]A list of publications, standards, products, and specifications that contain substantial information about XACML or make use of XACML in a substantial way can be found under http://docs.oasis-open.org/xacml/xacmlRefs.html.

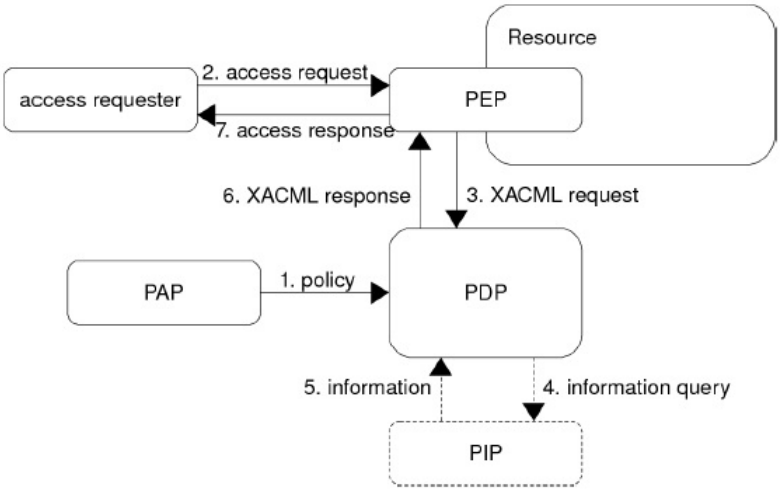

Figure 3.10: XACML enforcement environment architecture

which decides access requests by validating them against its policies.

2. An access requester wants to access a resource, therefore sends an access request to it, which gets intercepted by a Policy Enforcement Point (PEP). The PEP component ensures, that each access request to resources it protects is decided by a PDP.

3. To get an access decision, the PEP creates an according XACML request consisting of information about the access attempt, and sends it to the PDP.

4. If the decision making process requires additional information which is not provided with the XACML request, but is accessible through a Policy Information Point (PIP), the PDP queries this information. A PIP could be any kind of repository or data source.

5. The PIP returns the requested information to the PDP.

6. The PDP returns an XACML response consisting of the decision result.

7. After handling the results from the PDP, the PEP returns the access response to the access requester. Depending on the PDP decision result, this could be the desired access to the resource or an access denial.

The XACML meta-model is shown in figure 3.11. The root of all XACML policies is a policy or a policyset element. Policysets may hold one or more policies or other policysets. Each policy contains rule elements which are evaluated by the PDP. Target elements specify the context a rule applies to or not. A target is composed of subject, resource, and action elements. A subject element defines a human interacting with the system. A resource element defines a protected entity, such as a Web Service, file or process task in the context of a workflow management system. An action element defines

the operation that is performed on the protected resource element. If more than one rule applies a rule combining algorithm defines the outcome of the overall decision request [Tim05].

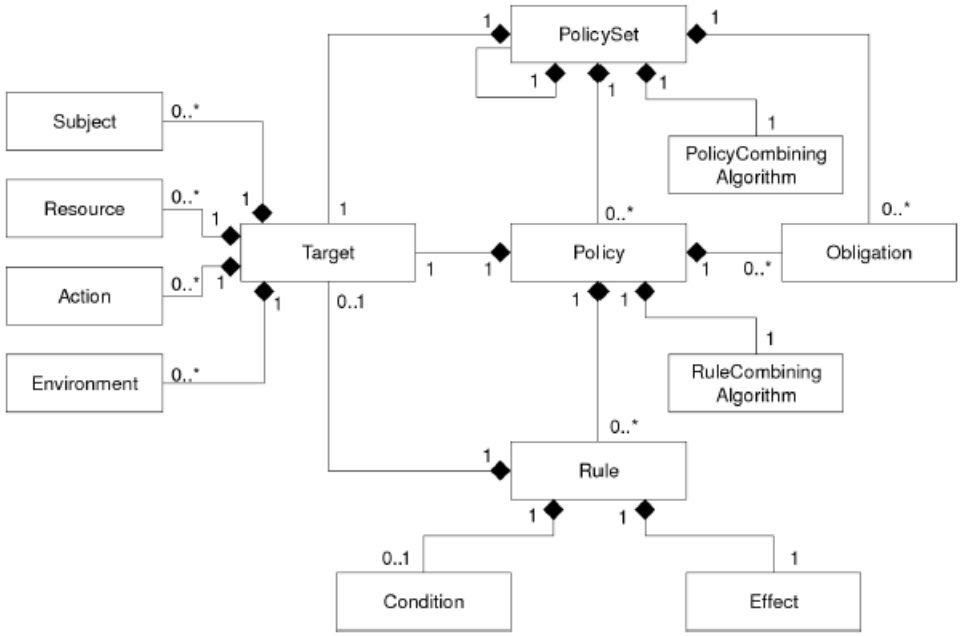

Figure 3.11: XACML policy language model

Condition elements further restrict the overall decision based on the contextual attributes of the access request. A condition contains a predicate which evaluates to either true or false, e.g. a subject's role attribute must be clerk. If the condition returns true the rule's effect is returned. The effect may result in a permit, deny, or not applicable statement. An obligation is an additional activity (e.g. sending a notification email) that must be performed by the policy enforcement point in case a policy applies to a specific request and rule effect [Tim05].

Delegation requests are part of specifics requests including conditions and obligations for delegatees. It is a mean to specify task delegation constraints (e.g. deadline, delegatee's attributes). In our work, we leverage XACML specifications to define delegation policies in our approach (see chapter 5).

3.3.4 Summary

In this section, we presented an overview of security concepts which are relevant to workflow systems. The main security properties was introduced to explain the role that it plays in total system protection. Moreover, we presented access control approaches (MAC, DAC and RBAC) to support security properties and to specify authorisation policies. It is widely accepted that RBAC models are most effective at enforcing workflow security.

In the next section, we explain how such access control models will secure workflows in general and task delegation in particular, and discuss their functionalities and limitations.

3.4 Level of Access Control within Workflows

Workflows traditionally have been used by business organisations for modeling and controlling the execution of business processes to achieve a business objective [WFM99]. In the classical software engineering approach the organisational context and related security requirements and threats are often considered during the design phase of a workflow, but internal controls are later defined and implemented in a manual fashion disjoint from predefined models due to a semantic gap and poor understanding of integration dependencies [CSBE08]. To tackle this issue, various frameworks, procedures and guides emerged to achieve information security within an organisation.

3.4.1 Organisational goals

Organisations adopted and defined control goals with respect to information security to secure and conduct their business activities [Sch03]. Information security became an organisational control goal that may be achieved by confidentiality, integrity and availability providing mechanisms, techniques and processes purposefully designed in order to control the business activities of individuals, groups, and whole organisations (see figure 3.12).

The Committee of Sponsoring Organisations of the Treadway Commission (COSO) defines processes and measures by which an organisation should be coordinated and directed to achieve information security [Moe07]. The COSO framework focuses on organisational measures and the definition of those processes is usually kept abstract to emphasise the independence of any technical, architectural and application level context. Other frameworks, such as COBIT, try to bridge the gap between the organisational processes and information systems. It is emphasis on achieving the linkage between the organisation, measures, and information systems [ITG07], while frameworks, such as the NIST (National Institute of Standards and Technology) Security Handbook [NoC95] primarily addressed the measurement and assertion of information system security properties on a technical level.

3.4.2 Secure workflow approaches

Much of the work on role-based access control systems within WfMS is based on the context of the RBAC96 access control model family [SCFY96], which since then has evolved to a NIST standard allowing for a standardised integration into security products [FSG+01]. In the RBAC96 model family, the central notion is that permissions are associated with roles, and users are made members of appropriate roles, thereby acquiring the roles permissions.

Atluri *et al.* presented the Workflow Authorisation Model (WAM) which concentrates on the enforcement of authorisation flow in task dependency and transaction processing by using petri nets. Though WAM only concentrates on the authorisation in a task's

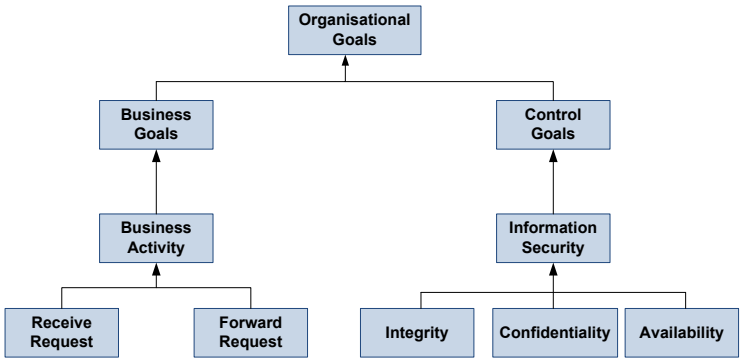

Figure 3.12: Business and control goals in an organisation

state and primitives, not the authorisation during resources access. WAM discussed the synchronisation of authorisation flow and the specification of temporal constraints in a static approach, however this is not sufficient to support workflow security requirements. This is because workflows need a more dynamic approach to synchronise the flow of authorisations during execution [HK03]. This is the case when we propagate authority from a *Prosecutor* to his *Assistant* to support delegation (see the motivating example : DS1).

Oliver *et al.* considered the specification of application-level security in workflow systems. The approach consisted of defining a workflow on three distinct levels. The level 3 describes workflow notions such as activities. Security requirements such as the fact that users should only be granted access to objects, while they require the access to perform some activity aspects are viewed as level 2. Level 1 consists of (one or more) databases where the information is stored [OvdRG98]. However, the specification of application-level security does not consider constraints that may appear frequently in WfMS, such as separation of duties based on the time at which they can be evaluated within a workflow [BFA99]. This is the case when we consider the same role (Prosecutor) for different organisations (Eurojust), where authorisation assignment have to be separated (SoD) from users and tasks perspectives (see the motivating example : DS2).

Another approach presented by Wainer *et al.* is an extension of the RBAC based on the RBAC96 model [WKB07]. It defines a pair of role-based access control models for workflow systems, collectively known as the W-RBAC models. The first of these models, W0-RBAC is based on a framework that couples a powerful RBAC-based permission service and a workflow component with clear separation of concerns for the administration of authorisations. The approach provides an expressive logic-based language for the selection of users authorised to perform workflow tasks, with preference ranking. Further, authors extended the initial model to W1-RBAC model. It allows some constraints to be overridden for the case that the workflow could not reach a defined end due to an unpredictable behaviour [WKB07]. However, W-RBAC models remain stateless and lacks of means to support dynamic policy enforcement when considering dynamic authorisations. This is the case, when an authorisation policy is updated with new delegation rules that grant authorisation for delegatees within the existing policy [GMC09].

3.4.3 Summary

As security is an essential part of workflows, a set of access control mechanisms should be introduced into workflow management systems to allow controlled access of data objects, secure execution of tasks, and efficient security management. In this section, we have presented organisational control goal in order to control and to secure the business activities of individuals, groups, and whole organisations. We have discussed different approaches supporting security in workflows and presented their scopes and limits regarding our thesis motivations for a secure and dynamic task delegation in workflows.

3.5 Analysis of Delegation in Secure Workflows

In a secure workflow system, it is imperative to specify which users (or roles) can be authorised to execute which tasks. Users may also be able to delegate their rights of executing a task to others. Much of research work in the area of delegation have been carried out. In this section, we discuss different approaches that tackle delegation in secure workflow systems. To do so, we firstly analyse delegation in workflows. We then discuss access control systems to ensure delegation of authority within workflow systems and present their limitations for our approach.

3.5.1 Delegation in workflows

The delegation of a task can be very useful for real-world situations where a user who is authorised to perform a task is either unavailable or too overloaded with work to successfully complete it. This can occur, for example, when certain users are sick or on leave. It is frequently the case that delaying these task executions will violate time constraints on the workflow impairing the entire workflow execution. Delegation is a suitable approach to handle such exceptions and to ensure alternative scenarios by making WfMS more flexible and efficient [Sch07].

Most of the work done in the area of security constraints and requirement modelling are focused on the infrastructure of WfMS and secure transaction management in workflow execution [AW05, Ven03]. There exists little related work in the domain of specifying task-based delegation. This observation is supported by research done by Russel *et al.* [RvdAHE05] and Hung *et al.* [HK03]. They outlined that existing solutions, such as the Workflow Authorisation Model (WAM) [AW05], are static and do not support security constraints in general and task delegation in particular.

Crampton *et al.* discussed delegation in the context of workflow systems using three different workflow execution models [CK08b]. The work offers a greater understanding of the effects of various delegation operations on the authorisation data structures in the context of role-based workflows [CK06]. In addition, the authors were able to determine when the delegator must be considered to be an authorised user and when the delegator will be able to retain existing task assignments. However, they did not consider task delegation constraints when delegating authorisations. The integration of delegation policies into existing policy is not treated and the problems of security conflicts with the policy compliancy are not yet addressed.

Atluri et al. have extended the notion of delegation to allow conditional delegation, where the delegation conditions can be based on time, workload and task attributes. When workflow systems entertain conditional delegation, different types of constraints come into play, which include authorisation constraints, role activation constraints and workflow dependency requirements [AW05]. Authors addressed the problem of assigning users to tasks in a consistent manner such that none of these constraints are violated. However, the problem of user-task assignment under delegation is not considered. The assignment relation is very important during delegation and needs to be done under specific constraints to ensure the delegation of authority.

Russel *et al.* proposed an approach supporting delegation [RvdAHE05]. They described the life cycle of a work item in the form of a state transition diagram with a particular focus on the resource allocation perspective. One of the main drawbacks of this approach is that it defines a static binding of all work items associated with a task to a single resource. This approach ignores additional events (transitions) during delegation execution and does not support secure and dynamic interactions within a workflow with regards to aspects of users, tasks, events, and data.

3.5.2 Delegation in access control models

Role-based access control (RBAC) is recognised as an efficient access control model for large organisations. In [ZOS03, ZAC03], authors extend the RBAC96 model by defining some delegation's rules. Zhang *et al.* proposed a flexible delegation model named Permission-based Delegation Model (PBDM) [ZAC03]. PBDM supports user-to-user and role-to-role delegations with features of multi-step delegation and multi-option revocation which are out of the scope of this dissertation.

Barka and Sandhu proposed a role-based delegation model based on RBAC96 model. The unit of delegation in them is a role. In addition, authors focused on role-based models supporting role hierarchies when studying delegation in the context of both RBAC0 model (flat roles) and RBAC1 model (hierarchical roles) of the RBAC96 family [BS00]. However, users may want to delegate a piece of permission from a role, which is not supported in such models. This will be an immediate priority in our approach to enrich our delegation access control model to enforce authorisation mechanisms based on event-based delegation policies.

The eXtensible Access Control Markup Language is an XML-based, declarative access control policy language that lets policy editors to specify the rules about who can do what and when. Unlike other application-specific, proprietary access-control mechanisms, this standard can be specified once and deployed beyond the boundaries of organisations and countries. Seitz *et al.* investigated how an authorisation management system based on XACML can be extended to use flexible delegation mechanisms. They developed a separate policy administration point component called "Delegent" that specifies allowed modifications on different elements of an XACML policy for different users [SRS+05]. Administrating delegation policies is, however, stateless and lacks of reactivity to support policy change when delegating a task. We need a reactive approach to reflect events change and to support a dynamic enforcement of policies (see DS1 example when Alice revoke Bob's work).

Chadwick *et al.* developed a middleware authorisation framework, which focuses mainly on role based access control (RBAC) model. It supports additional conditions such as role assignment validity, delegation depth and target access clauses [CO02]. The authorisation framework does not provide direct support for bilateral exchange of policies and credentials to address privacy issue and trust. This is not enough to manage security for systems in which organisations are dynamically built with the collaboration of multiple independent organisations sharing their resources. This is especially the case when we consider authorisation decision making supporting delegation policies for process-based human interactions cross-organisations. For instance, two *Prosecutors* from different organisations need to collaborate together by sharing their resources in the global delegation scenario DS2.

3.5.3 Summary

We have presented a literature review related to the delegation requirements in workflow and access control models. We have discussed approaches, models, and technologies which fit with delegation and highlighted their functionalities and limitations.

To the best of our knowledge, most of the work done in the area of workflow and access control systems does not treat delegation in sufficient details and deserve more investigations. Secure delegation implies the controlled propagation of authority dynamically with regards to workflow's invariants (tasks, users and data). Consequently, an extension of the RBAC model will be defined in the next chapter to support such requirements within workflows.

3.6 Conclusion

This chapter outlined the research foundations of this thesis. We presented concepts fundamental to the understanding of workflow management systems, processes modelling as well as the security in information systems. We have shown that in the classical software engineering approaches, the organisational context and related security requirements are often considered during the design phase, but internal controls supporting human interactions such as task delegation are badly defined and disjoint from system specifications due to a semantic gap and poor understanding of delegation requirements.

Based on this solid knowledge foundation, the next chapter will introduce our delegation approach, its fundamental concepts, design decisions and supporting framework requirements. We aim to come up with an approach to secure task delegation within workflows. The motivation is **to bridge the gap between organisational needs and security functionalities**. Our research directions is oriented access control requirements in business processes. From a process perspective, we will present a generic model for task delegation **to support organisational flexibility when modelling a workflow**. From a resource perspective, we will define an access control model **to ensure delegation authorisation in access control systems**.

Moreover, the delegation of authority defines policies which are defined from existing policies and are specified dynamically. However, existing work on access control systems

remain stateless and do not consider this perspective. In chapter 5, we come up with an approach **supporting dynamic enforcement of delegation policies**. Additionally, **the integration of such policies have to be computed and compliant with the existing policy** based on the delegation constraints. In order to control the delegation behaviour and to specify its authorisation policies in an automated manner, we will to monitor the task execution during delegation and define specific rules to generate and integrate delegation policies in the global authorisation policy.

Chapter 4

Modelling Task Delegation in Workflows

Contents

4.1 Introduction

In this chapter, we present a novel approach for modelling task delegation in workflow systems. By leveraging the foundations described in the chapter "State of the Art", we define a task delegation model that not only meets the requirements at the organisational level, but also enables performing delegation in a secure and flexible manner at the security level.

We answer the interrogations defined in the chapter "Context and Problematic" related to the motivation factors, the delegation model and the access control model. In order to capture the requirements regarding the definition of our delegation approach, we present, first, a detailed **taxonomy of factors for delegation with respect to workflow aspects**. Basically, we aim to separate the various aspects of delegation with regards to workflow's actors, tasks and resources. Alongside this taxonomy, we motivate that the **task delegation model will support organisational flexibility in the human-centric workflow systems, and ensure delegation of authority in access control systems**. On one hand, the organisational flexibility deals with the organisations adaptation capacity to support user-to-user delegation. On the other hand, the delegation of authority defines **constrained access control over workflow systems**. Moreover, we show how this model will support security requirements when considering our task-based delegation approach.

This chapter is organised as follows. Section 4.2 presents the motivation factors for delegation. It defines a classification based on both control and resource layers in a workflow. Delegation factors depend on the organisation requirements, the process definition, and the resources constraints. In section 4.3, we introduce the notion of organisational flexibility for task delegation. It motivates delegation within an organisation to ensure flexible execution and to support exceptions when moving away from the predefined policy. Concretely, delegation works on defining alternatives with unavailable functional roles (workflow's roles) to cope with the organisation rigidity. Section 4.4 analyses delegation in business processes. Workflows model and control the execution of business processes in an organisation. It presents a novelty to study a delegation process as a multilayered state machine to allow delegation to complete. Section 4.5 focuses on modelling task delegation for human-centric workflows. We aim to ensure a user-to-user delegation with heavily human interactions to perform a delegation request. This request needs to support additional requirements related to the organisation flexibility and the authorisation

policy definition such as negotiation. Negotiation is defined when initiating delegation. It aims to offer a wide choice of delegation ensuring flexibility and controlling access rights. Finally, we address security requirements for delegation in section 4.6. We aim to reason about task delegation from a resource perspectives to analyse and specify task delegation constraints while accessing workflow's resources. To that end, we define a task oriented access control model (TAC) based on the RBAC model.

4.2 Motivation Factors for Delegation

The delegation process within workflow management systems is well studied at the conceptual and technical levels [Sch03]. A non exhaustive identification of delegation factors have been presented in [Gay05]. We motivate and enrich this classification based on both the control and the resource layers in a workflow [GSFC08]. The delegation process will inquire different needs to address two important issues, namely allowing task delegation to complete, and having a secure delegation within a workflow. Allowing task delegation to complete requires a model that forms the basis of what can be analysed during delegation process from the negotiation phase to the completion phase [GSFC08]. Secure delegation implies the controlled propagation of authority, ensuring confidentiality at the control and data flow layers as well as availability at the task assignment layer and integrity at the data layer, thereby specifying security requirements on both design and run time. Figure 4.1 is the graphical representation of this taxonomy.

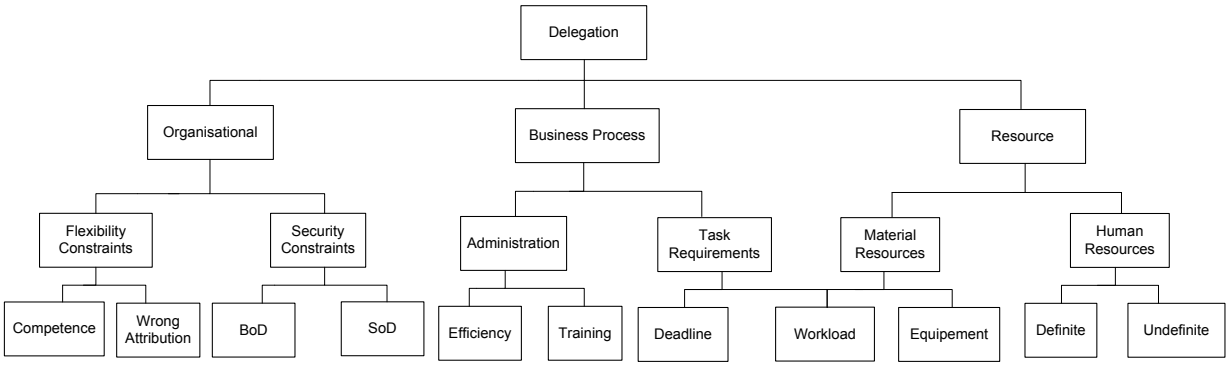

Figure 4.1: Taxonomy of the motivation factors for delegation

4.2.1 Organisational

The normal execution of the workflow reflects the real organisation. If a delegation process occurs, it means that the normal execution does not model the real process anymore. The following points describe the identified motivation factors regarding the organisational factors of the delegation taxonomy :

- **Flexibility constraints :** Such constraints can be a causing factor for delegation if they are well identified in the organisation and users are accepting the notion of

collaborative work. It is often the case that implicit knowledge cannot be shared among users and that only some specialists possess the competence on a specific field.

- **Competence :** The required competencies to achieve a task can evolve. The level of needed competencies is lowered and the actual user could delegate the task to someone with fewer technical skills, giving him time to concentrate on other tasks (see DS1 scenario described in the motivating example). The inverse is also possible, with a task evolving toward a greater complexity and the actual user is not able to deal with it anymore. Delegating to a better qualified user will reorganise the workflow.

- **Wrong attribution :** Considering an administrative distribution of tasks, a user assigned with a task not objectively fitting with his competence profile may need delegation. This category of delegation solves the problem of wrong attribution without involving another administrative process.

- **Security constraints :** Security is an essential and integral part of workflow management systems. Protecting application data in workflow systems are defined through access control policies. Organisational policies may conflict and require a user to delegate. Specific organisational constraints are identified below :

  - **Separation of duty (SoD) :** The separation of duty is normally modeled as a gathering of rules that reflects the organisational policies. It is an important structuring point of the architecture of a workflow in its normal execution. The separation of duty ensures that a user delegates a task not to enter in conflict with one of the organisational rules. SoD can be motivated by many reasons such as Conflicts of interest (CoI) or frauds avoidance.
  - **Binding of Duty (BoD) :** The principle of BoD is used to express that if a certain user executed a task, this particular user must also execute specific other tasks to complete a workflow. Therefore the other tasks are restricted such, that only this user can execute them in the pertained workflow instance. It can be seen as the inverse model to SoD, which demands to disperse the responsibility for the execution of a set of tasks, while BoD demands to bind it. Like for SoD, many delegation examples exist in the business world, in cases where a delegator delegate his task, but due to entailment restrictions such as the BoD, the delegatee must also execute additional tasks.

4.2.2 Business process

Workflow management systems automates the management and coordination of organisational or business processes. A workflow typically comprises a set of coordinated activities,

known as tasks. The following points describe the identified motivation factors regarding business process management in the delegation taxonomy :

- **Task Requirements :** In this category, the dynamic of the task is taken into account. Constraints or rules structuring the task as well as multiple external factors can evolve. This evolution can be either predictable or not but it may inflict on the execution of the task.

  - ☐ **Deadline :** A deadline defines a time constraint linked to the task. For instance, when the delegator is running out of time and he is not able to respect a time limit. To not cancel the executed task, a delegation will be supported.

  - ☐ **Workload :** Considering the current filling of his worklist, the delegation process could be initiated if the worklist is overloaded and the addition of a new task would result in time constraints not being met.

- **Administration :** A less technical but interesting motivation for delegation is the interest of the managers in the performances of their processes and in the involvement of the team management.

  - ☐ **Efficiency :** A user may have all the required authorisations, resources and competencies to achieve a task, but it may be a positive thing to delegate this task to another more specialised principle in order to make the workflow more efficient (see DS2 scenario in our motivating example).

  - ☐ **Training :** By delegating a specific task, often, to a non specialised user it will help him to acquire new skills and with the new responsibilities associated, to be better involved in his work. Depending on the result of work done, the delegatee, through successive delegations can earn the delegator confidence.

4.2.3 Resource

A task cannot be achieved due to a lack of resources. In the literature, a task defines a unit of work that at each invocation performs the binding between different resources needed to complete a specific part of the workflow [RvdAHE05]. The resources that may be involved are different. We distinguish material and human resources for business objects and workflow actors, respectively. The following points describe the identified motivating factors :

- **Human resources :** The user who is authorised to perform a task misses one or several necessary resources. We distinguish here, two types of motivation for delegation related to human resources. This category gathers the situations where the user will not be available during the execution of a task. Then he can delegate it to another user to act on his behalf.

  - ☐ **Definite :** Here the user is aware of the length of his absence and moreover it can be a repetitive fact. This definite unavailability can be expected or unexpected.

  - ☐ **Indefinite :** Here the user cannot tell how long he will not be able to perform his allocated task. A representative case would be an employee leaving a company. This case can be both expected or unexpected.

- **Material resources :** Generally, the manipulation of material resources is interfaced by one or several entities called applications or services. These applications consist of functions that manipulate business objects. We distinguish two motivation factors :

  - ☐ **Equipment :** In this case the motivation to delegate is obvious as the execution of the task will be stopped by the absence of one or several material goods.

  - ☐ **Workload :** See section 4.2.2.

4.2.4 Link with the case study

In the table below, we summarise the delegation scenarios described in the motivating example. It contains the different observations based on the MLA case study and the the motivations factors for each scenario.

Scenario	Organisational	Business Process	Resource
DS1	Flexibility constraint in the organisation hierarchy	Workload : Prosecutor Alice is overloaded	Human resource (e.g. Alice requests a leave of absence)
DS2	Flexibility constraint cross organisations	Efficiency : Prosecutor Claude is more specialised than Prosecutor Alice	Material resource (e.g. Alice is lacking information about the MLA request)

Table 4.1: Summary of motivation factors for delegation scenarios in MLA

4.2.5 Summary

The detailed classification of the delegation motivations factors presents the essence of this chapter. Actually, organisational flexibility, task requirements and security issues will be discussed in the rest of this chapter. Our delegation approach is a mechanism offering flexibility during workflow execution by ensuring alternatives depending on the organisation adaptability. Organisation adaptability depends on organisational roles specified in the existing policy where workflow's actors execute a process based on their access permissions. In addition, a task delegation behaviour is related to the process and has to be aligned with the process definition and responds to its objectives. Moreover, delegating a task inquires a propagation of authority that must be controlled and does not abuse the access control requirements. Delegation of authority aims to bridge the gap between both workflow and access control systems.

4.3 Organisational Flexibility in Workflows

In the context of heavily human-centric workflow systems, the requirements for human interactions and monitoring can be summarised as transparency and control. Transparency addresses the revelation of the control flow dependencies. This allows to react accordingly to exceptions and compensations during execution. Control fosters the behaviour of organisations according to the existing policies. One type of transparency and control supporting mechanism in human-centric workflows is that of task delegation [GCSL07]. Delegating a task may need additional requirements from the organisation itself. Actually, the process of delegation depends on different factors related to the organisational flexibility in addition to its security constraints (see figure 4.1).

Flexibility is becoming more important for organisations. Information technology (IT) has been proposed as a tool which can aid the attainment of flexibility. IT can be an enabler of organisational flexibility. First, specific types of information technology provide flexibility by providing more flexible ways of doing things. Second, the information systems (IS) infrastructure can be designed so as to provide flexibility by allowing the organisation to adapt the information systems to new competitive environments [GP00].

4.3.1 Flexibility constraints

Flexibility constraints can be a causing factor of delegation. It is often the case that additional competence or a new task assignments may be required to achieve a task. In this way, flexibility is achieved through diverse specialisation. This occurs where each organisation focuses on what it does best and leverages the capabilities of other entities for complementary activities.

The organisational flexibility is used to describe situations where individual organisations concentrate on their core competence and use internal or external human resources where required to enable a task to be achieved. Such requirements express the need for human resources management. An organisation obtains flexibility by increasing the levels of internal and external flexibility available to it by increasing its ability to manage human resources. The importance of the human element in creating flexibility is shown

by Suarez *et al.* [FFSF95] who admits that high worker involvement and flexible wage schemes provide organisations with more flexibility than the flexible IT they use.

Golden *et al.* [GP00] identifies four dimensions of flexibility; temporal, range, intention and focus. The first is temporal; how long it takes an organisation to adapt. The second is range; the number of options that an organisation has open to it for change that was foreseen and the number of options it has available to react to unforeseen change. The third is intention; whether the organisation is being proactive or reactive. The fourth is focus; whether the flexibility is gained internally or externally. These dimensions are important for delegation where time constraints, delegation scope [ZAC03], dynamic delegation and user-to-user delegation define our approach to support task delegation requirements.

4.3.2 Organisational flexibility in practice

We introduced a workflow scenario related to the European administrations collaboration (see chapter "Context and Problematic"). In our example, we described the MLA process cross Eurojust organisations A and B, and detailed the different business actors and resources models involved in the process. We distinguish *Prosecutor* as the main responsible that collaborates with internal and external employees (Assistant, National Correspondent (NC), Judge and Judicial Authority Officer (JAO)) to process the MLA request (see figure 4.2).

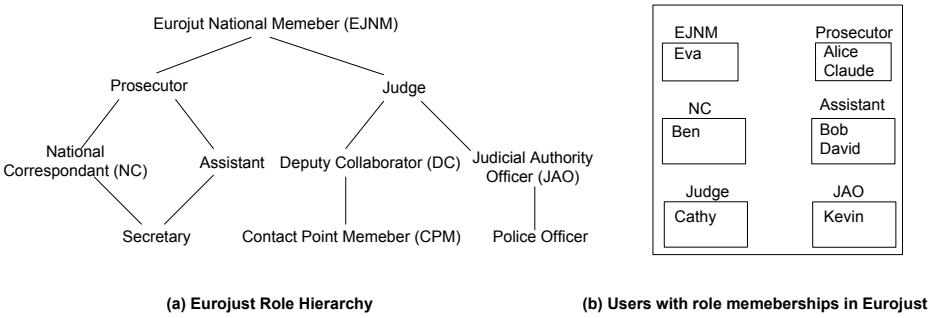

Figure 4.2: An example of organisational role hierarchy and users in Eurojust

Depending on the current control-flow sequence, workflow actors can evolve and change from the predefined workflow model. This can lead to a new rearrangement of actors in order to optimise the process execution. In addition, unexpected events can happen without being modelled beforehand. For example, a *Prosecutor* delegates a part of his work to a subordinate due to emergency situations. For instance, Alice needs to send the MLA request to authority B. Alice is overloaded (lack of resources) and needs to delegate his assigned task to one of his assistant. Delegation criteria is based on the role hierarchy (RH) of Eurojust, where the *Assistant* Bob is a subordinate to the *Prosecutor* Alice based on the authorisation policy definition.

Role hierarchies are a natural means for structuring roles to reflect an organisation's lines of authority and responsibility, and are organised in partial order $\geq$, so that if r_1

$\geq r_2$ then role r_1 inherits the permissions of r_2. A member of r_1 is also implicitly a member of r_2. In such case, r_1 is said to be *senior* to r_2. Note that such hierarchy can be mapped to a different organisations in a collaborative context based on role mapping (RM). A new reassignment of a task can occur cross organisations while keeping the same role but assigning it to a different user. This can be the case when a specific specialisation is required and an external role is called from a different organisation to perform a task. In our case study, Alice member of role *Prosecutor* sends a delegation request to his colleague Claude located in the Eurojust organisation B (see DS2 scenario in the motivating example).

4.3.3 Requirements for organisational roles

Within organisations, and thus in workflow applications, the concept of a hierarchy of users/roles is prevalent. Users are placed in one or more units such as departments, divisions, or groups, and they have different bosses, at different hierarchical levels. It is important to discuss the differences between the proposed organisational unit hierarchy and the one that is based on their role-based access control models hierarchy to manage human resources [WKB07].

Organisational flexibility depends on the rearrangement of the organisation members (users) when reassigning a task to a delegatee. The involved user's members of similar roles have to be distinguished on the security level. For instance, *Prosecutor* is a role defined in both organisations A and B (see figure 4.2). However, permissions given to *Prosecutor A* are different from *Prosecutor B* for privacy reasons. To distinguish between prosecutors of Eurojust A and B on the security level, we need an additional mapping between workflow's actors and their security roles defined in the access control system.

Moreover, we observed that delegation depends on entities related to the organisation, the process and the resource's access in a workflow. In order to support such requirements, we identify three entities : user, functional role and security role within an organisation. The three entities behave like the class diagram described in figure 4.3.

A functional role defines the workflow's actors. It depends on task's assignment. A security role is defined in the access control system and depends on rights. A user may be assigned from the "can execute" association to functional roles for which he is entitled and qualified. This assignment can take place at any time and does not describe the current system state. Therefore it remains static. Only the association "executed" provides the instantiation of the execution and the dynamic assignment of the functional role. This association is described by an association class : the attributes "from" and "to" describe, for what period a user assignment to a functional role is active. The active attribute describes whether this user is performing effectively this role in the current system state (e.g. the user is logged in).

Moreover, a user is member of security roles. Security roles is part of the access control systems and defines the organisation policy in terms of users, rights and resources access. A user can be assigned to one or more security roles that are explicitly defined in the policy. In addition, we define memberships in security roles that are tied to the exercise of a functional role. Permissions that are dependent upon whether a functional role is being actively carried out, are represented on the "assigned when active" association : A

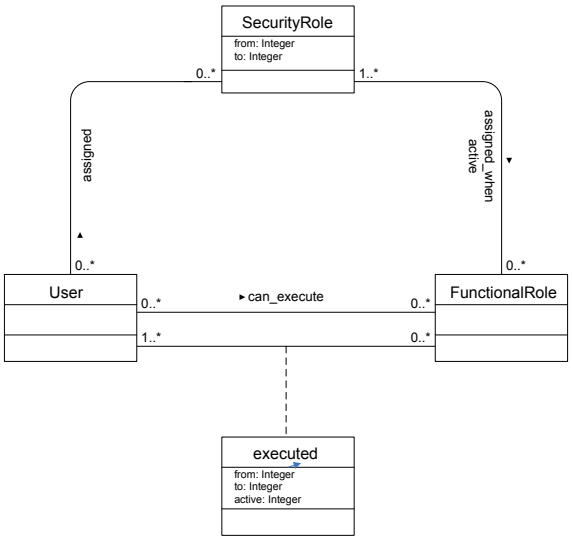

Figure 4.3: Organisational roles mapping

user executing a functional role will inherits of the specific permissions mapped from the security role that is assigned to the functional role when active (see figure 4.3).

The mapping of functional and security roles holds when a user logs in the system with the appropriate security role. By logging the user's "active" field is set to true : the functional role is busy and active. In addition to the user, the validity of the transferred rights is mapped to the security roles required for the execution as long as he is assigned as active to execute a functional role. The mapping is important for task delegation. It ensures that a delegatee is assigned to the functional role to execute the task and having the required permissions to access its resources. Functional roles computation is based on the RH (role hierarchy) or RM (role mapping) relations depending on the context of delegation which may be local or global. In addition, security roles is specified in the authorisation policy and transferred to the specific functional role when task assignment is active.

In this dissertation, we assume that delegated roles define activated functional roles inheriting security permissions. They are computed based on the access control model specifications (see section 4.6.2).

4.3.4 Summary

The organisational flexibility in workflows is an important part for our delegation approach. It defines the capacity of an organisation to evolve during runtime and offers alternatives to support exceptions within organisational processes. Organisational roles presents a way to model, specify and organise policies. Policies are law regulations within a workflow in order to support functional and security roles. Those roles are tightly related and have to be mapped in order to control roles and rights during delegation. Mapping roles is crucial to ensure the delegation of authority. It offers a solution to specify workflow actors into access control systems in order to compute their permissions during task

delegation. Delegated permissions defines privileges that have to be computed based on the process and resources requirements. This analysis will be discussed in the upcoming sections.

4.4 An Extended Analysis of Delegation in Business Processes

In this section, we define task delegation in business processes. Workflows model and control the execution of business processes in an organisation. A workflow is defined as a set of coordinated activities (tasks) that achieves a common business objective. Tasks in a workflow are related and dependent upon each other reflecting the coordinated activities in the business process. Task delegation needs to take into account this ordering with regards to users, tasks and resources. It defines the way to monitor and allow delegation completion within a business process.

4.4.1 A workflow model

The workflow management coalition (WfMC) developed a model using states transitions that illustrate the basic underlying concepts which are necessary to scope the effects of the workflow applications [WFM99]. The workflow enactment service may be considered as a state transition machine, where individual process and tasks instances change states in response to external events (e.g. completion of a task) or to specific control decisions taken by a workflow engine (see definition 5).

Definition 5 (A Workflow Model) *A workflow model $\mathcal{W}$ is a partially ordered set of tasks (*T,$\leq$*) that is coordinated by a set of events* E*. The order of task execution is orchestrated by matching the input and output event(s)* E *of each task.*

In addition, we consider a task as a single unit of work. Each executing task is termed a work item [RvdAHE05]. In an elementary form, a task is an atomic unit of work. In a compound form, it modularises an execution order of a set of subtasks. It can define a sub-process or a block of tasks. In this dissertation, we consider the elementary form. The basic states for a task life cycle are *Initial, Assigned, Executed, Cancelled Failed*, and *Completed* [WFM99].

4.4.2 Basic task delegation model

A task, once created, is generally assigned to a user. The assigned user can choose to start it immediately or to delegate it. Delegation depends on the assignment transition, where the assigned user has the authority to delegate the task to a delegatee in order to act on his behalf.

Delegation can be introduced to a task model through an extension that supports additional states and transitions. The transition *Delegate* is closely related to the *Assign*

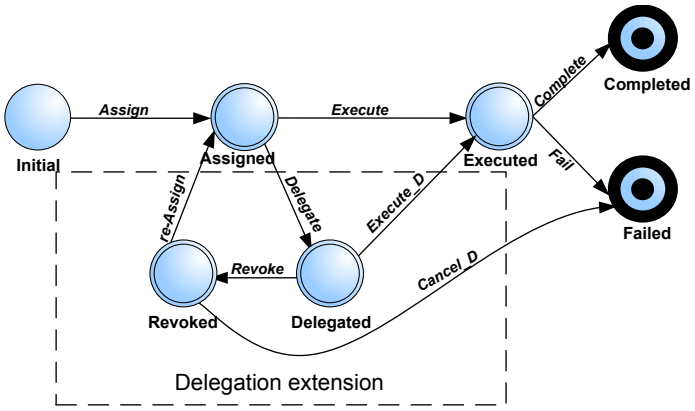

Figure 4.4: Basic task delegation model

transition, where the assigned user has the authority to *Execute* or *Delegate* the task. The *Revoke* transition is derived from *Delegate* transition, such that it can be considered as the cancellation of the task delegation. The internal delegation states are *Executed, Revoked* and *re-Assigned*. The delegation behaviour remains internal according to the task model, where *Completed* and *Failed* are the final states (see figure 4.4).

Note that revocation may be defined as an alternative to reassign the task again. In addition, the delegation of a task from one user to another has to be managed and executed in a secure way, in this context implying the presence of a fixed set of delegation events (transitions) with regards to the workflow invariants (e.g. user, task and data).

4.4.3 Securing task delegation within a workflow

Security is an essential and integral part of workflows, addressing the properties of integrity, confidentiality and availability. In a secure workflow, integrity prevents the unauthorised modification of information, while confidentiality implies that no data or resource is accessed by unauthorised users at anytime. Availability moreover, implies that a resource should be available when it is needed (see definition 6).

Definition 6 (A Secure Workflow Model) *A secure workflow is a computer supported business process with security requirements defined in the authorisation policies. Formally, a process is composed of tasks that need to be done by defined users* U*. Further, each task is given an authorisation, which describes the possible data permissions. A user needs access permissions* P *to a set of data* D *during task execution.*

Our concern is to come up with a secure delegation model based on a set of invariants for workflows from the aspects of users, tasks, and data. The novel part of this model is separating the various aspects of control in a workflow and portraying it as a multi-layered architecture. This model defines an initial draft for our final delegation framework. Basically, it aims to give a clear and coherent link between control and authorisation flows

during delegation. Additional delegation requirements and constraints will be analysed as one goes along with our delegation approach.

Olivier *et al.* state that a workflow system should be considered at three levels in terms of its components (task's assignment, control, data) [OvdRG98]. Securing a workflow involves enforcing security principles at all three levels. Hung *et al.* developed a secure workflow model using a multi-layered state machine to manage the flow of authorisations at different layers for a secure workflow execution [HK03]. A multi-layered state machine can enable the analysis, simulation and validation of the workflow under study before proceeding to implementation. In addition, it can serve as a powerful tool for modelling a secure delegation framework at a conceptual and logical level with regards to the aspects of task, control and data.

We present a basic and secure task delegation model using a multi-layered state machine within a workflow. We define three layers : *Task, Control* and *Data* (see figure 4.5).

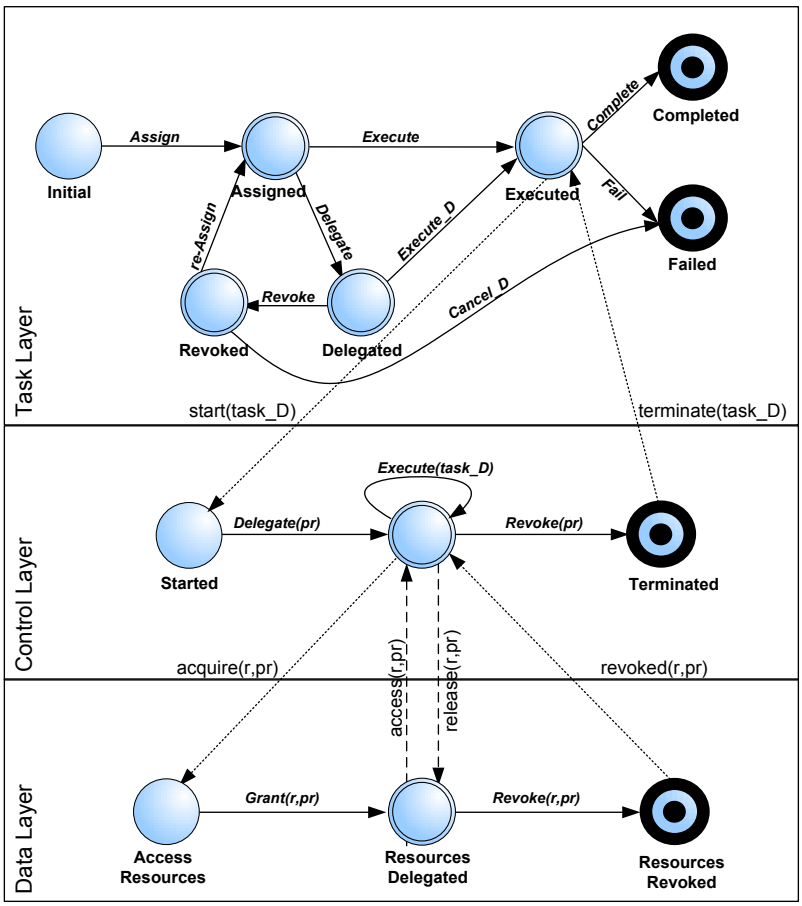

Figure 4.5: Multi-layered state machine for secure task delegation

The novelty of this model relies on the separation of the various aspects of delegation, and in its portrayal as a multi-layered architecture. The major motivations for using a multi-layered state machine are the modelling of different aspects of authorisations in

a single framework, and the ability to address different security services to handle the security properties in different layers. For instance, access control mechanisms can be applied to *Task Layer* and *Data Layer* to handle the security property of authorisation for delegating or revoking tasks and resources to and from users, respectively.

We impose security requirements on events (transitions) to ensure the security properties of integrity, confidentiality and availability. During delegation, the interactions between the different layers are triggered by delegation events. These delegation events imply appropriate authorisation on the delegatee side for further actions (e.g. starting the delegated task) as well as contain required context for those actions (e.g. accessing delegated task resources).

In the *Task Layer*, we require that *Assign* defines availability : *"For every task there must be at least one user (delegator) who is able to execute (delegate) the task"*. In addition, the assignment of the task means that the user has the authority to execute it, thereby controlling confidentiality and integrity of the assigned task.

In the *Control Layer*, *Delegate* defines the authority of delegating a task. We define a privilege *(pr)* (so called permissions) as a role assignment given authorisation to access resources. We require that *"A delegatee can only perform the delegated task if and only if the task is delegated and delegated privilege is granted by the delegator"*. The control layer monitors the behaviour of the task delegation. It involves the events generated from the task assignment layer and will generate events to trigger the data layer to be executed (see *acquire (r,pr)* in figure 4.5).

In the *Data Layer*, data are stored as resources. We define *(r,pr)* as a delegated resource to the delegatee. We require that *"A delegatee can only access delegated resources if and only if the delegated privilege is granted to access the delegated resources"*. Granting and revoking resources will ensure the integrity and confidentiality of resources.

Moreover, we define additional events supporting concurrent states. This is a practical property for a workflow model because there may be more than one delegated task running concurrently and also a given resources can be accessed by a set of concurrently running tasks. In order to avoid an over-privileged delegatee at anytime during the execution of a task, a delegatee is asked to release the resource based on the agreement with the delegator (see *release(r,pr)* in figure 4.5).

4.4.4 Summary

Securing task delegation within workflows demands requirements with regards to workflow's layers : task, control and data. We identified the main requirements allowing a delegator to assign the delegated task, to monitor delegation by computing privileges, and to control access resources. Monitoring delegation defines human interactions within an organisation. The process is modeled in an ad-hoc manner where delegation principals (the delegator and the delegatee) interact dynamically with respect to the organisation policy. In addition, the defined policy specifies how this interactions will ensures resources sharing in a secure way. Human interactions and access control over task delegation will be discussed in the rest of this chapter.

4.5 Modelling Task Delegation for Human-centric Workflows

In this section, we aim to model task delegation for human-centric workflows. In the context of delegation, interactions describe a user-to-user delegation to issue and perform delegation. Delegation needs to support requirements related to the organisation flexibility and the authorisation policy definition. To that end, we enrich our task delegation model presented previously with additional requirements that may be inquired depending on human interactions during workflow execution. Moreover, task delegation may involve security requirements regarding resources access (e.g. time constraints, permissions management). These requirements can be defined during execution in order to offer a wide choice of delegation ensuring flexibility and controlling access rights.

4.5.1 Delegation kind

Delegation defines a process describing the interaction between users, authorisation and resources under specified constraints. Constraints are related to the way the delegation is defined and will be executed. This varies from the type to the steps of delegation. A delegation can be defined in two different ways. Actually, we distinguish between two kinds of delegation that need to be clarified : *Administrative* delegation and *Ad-hoc* delegation.

When task delegation occurs frequently, has a regular pattern, then this defines an administrative delegation [Sch03]. Administrative delegation is the basic form of delegation in which an administrator or system authority assigns attributes and privileges to enable users to conduct certain tasks. This process typically happens when a user joins a security domain. The delegator, in this case, represents the authority of the system. This administration is predefined and a transfer of responsibilities from the delegator to the delegatee.

However, new requirements may indicate that the current organisational structure and procedures do not reflect the goals of the involved principals such as the evolution or change of tasks. An initially temporary and ad-hoc delegation must now become part of the regular administrative delegation supporting dynamic constraints for delegation.

4.5.2 Delegation of privileges

Delegation of privileges may be classified into at least two kinds : *Transfer* delegation and *Grant* delegation [CK06]. In transfer delegation models, the ability to use a delegated access right is transferred to the delegatee; in particular, the delegated access right is no longer available to the delegator. It is often desirable that sensitive access rights may not be available to a large number of users at any given time. Such requirements are usually expressed in the context of administrative delegation, where the transfer delegation policies prove to be more useful when an access control policy specifies cardinality limits on the availability of access rights between users [Sch].

A grant delegation model, following a successful delegation operation, allows a delegated access right to be available to both the delegator and delegatee. Grant delegation models are, primarily, concerned with allowing the delegatee to use the delegated access right. Such requirements are usually expressed as cardinality constraints in the context of ad-hoc delegation [Sch, CK06].

4.5.3 Task delegation model

Previously, we defined a basic task delegation model (TDM) based on the task life cycle in the workflow management coalition. We fully complete the delegation process based on additional events. Figure 4.6 depicts a UML state diagram that illustrates the life cycle of our TDM in the form of a state transition diagram from the time that a task is created through to final completion, cancellation or failure. It can be seen that there are series of potential states that comprise this process. A task, once created, is generally assigned to a user. The assigned user can choose to start it immediately or to delegate it. Delegation depends on the assignment transition, where the assigned user has the authority to delegate the task to a delegatee in order to act on his behalf.

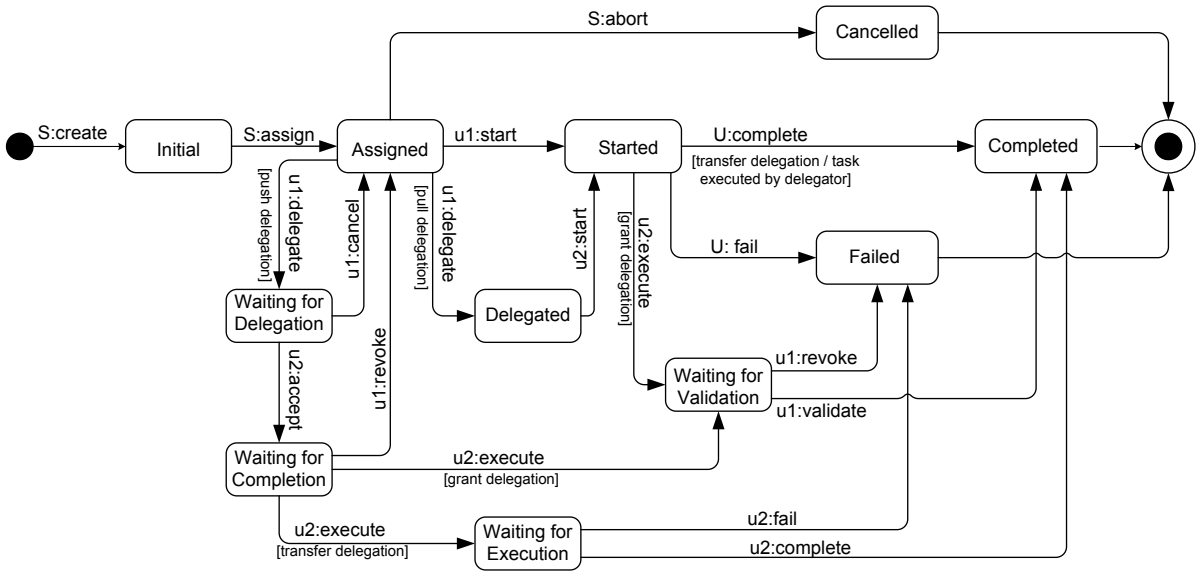

Figure 4.6: Task delegation model

Intermediate events define controlled delegation within a workflow (e.g. *delegate, cancel, revoke*). For instance, the delegator might want to cancel. Our TDM would then go back to the previous state (*Assigned* state). The delegation control flow behaviour remains internal according to the task model, where *Completed*, *Cancelled* and *Failed* are the final states. Moreover, we enriched the model with intermediate states supporting delegation features such as :

- **Delegation mode** : It defines how delegation request is issued. The Pull mode assumes that a delegator has at his disposal a pool of delegatees to be selected to

work on his behalf. The Push mode assumes that a delegator is waiting for an acceptance from a potential delegatee. Derived transitions from push mode are *accept*, *cancel* and *revoke*.

- **Delegation type** : It refers to the delegation of privileges management. It may be classified into grant or transfer. A grant delegation model allows an instantiated task to be available for both delegator and delegatee. As such, the delegator is still having the control to *validate* or *revoke* the task, and the delegatee to execute it. However, in transfer delegation models, the assigned task is transferred to the delegatee. There is no validation required and the task is terminated by the delegatee. Transfer delegation is used to support administrative delegation.

Note that each edge within the TDM is prefixed with either an S or an U indicating that the transition is initiated by the workflow system or the human resource respectively. We define u_1 and u_2 belonging to the set of users U, where u_1 is the delegator and u_2 the delegatee.

4.5.4 Negotiation in user-to-user delegation

From our research perspective, we argue that task delegation may be seen as distinct from administration. Three characteristics can be used to support this distinction. These are the representation of the authority to delegate (grant); the specific relation of a principal to an object (resources access); the duration of this relation (delegation constraints); and the validation requirements (reviewing and evidence). In addition, we argued that an ad-hoc delegation supporting human interactions can be done in two modes : either the pull or the push mode. Pull mode assumes that a delegator has at his disposal a pool of delegatees to be selected to work on his behalf. Push mode assumes that a delegator is waiting for an acceptance from a potential delegatee.

In the ad-hoc delegation protocol, we can identify three main steps : negotiation, declination and acceptance. The two last steps depend on negotiation. We consider negotiation as the trigger point for the main operations. We identify several factors related to user-to-user delegation during negotiation :

- **F1. Scope** : This factor describes the scope of delegation. Basically, the delegator proposes the degree of delegation regarding the context of delegation propagation in an organisation such as cascaded or multistep delegation [ZOS03].

- **F2. Time** : This factor defines one of the delegation constraints: the deadline. Delegation may be actually temporary. This involves a time constraint specifying a time window for the delegatee. This constraint utility can avoid also a long period of inactivity of the delegatee.

- **F3. Workload** : The negotiation here deals with the delegated amount of work. In fact, the delegatee may be overwhelmed by the number of tasks assigned to him, a situation which can be sorted by reducing the workload.

- **F4. Evidence** : This factor is a specific type of business object that can be manipulated by a task. Task execution may generate evidence for review by the delegator. The negotiation here deals with the reviewing specifications issued by the delegator to validate the completion of delegation.

- **F5. Privileges** : This factor can be a role assignment or an action on a resource. Privileges will be granted to the delegatee later on to execute tasks or access specified resources. Privileges can be permanent or temporal depending on time constraints.

Moreover, we identify the relationship between delegated users and negotiation factors (see table 4.2). Factors can be defined by the the delegators. In fact, they are the users initially assigned to execute tasks and authorised to delegate, and so expresses their delegation specifications. As part of the negotiation some of the factors can be modified by the delegatee. In fact, both principals can negotiate time and workload. This consists of giving the delegatee the ability to extend the deadline or to reduce the workload.

Factors	Delegator	Delegatee
F1	✓	
F2	✓	✓
F3	✓	✓
F4	✓	
F5	✓	

Table 4.2: Negotiation factors specified by delegation principals

The declination step occurs when the negotiation failed and the proposed specifications (negotiated factors) are rejected. We consider this step as a precondition to restart delegation. The acceptance step consists of granting delegated privileges, performing the task, and reviewing it in order to complete the request (see next section).

4.5.5 Delegation protocol supporting negotiation

We introduce a delegation protocol to support the dialogue between a delegator and a delegatee during a secure task delegation. The delegation protocol will support the defined *Control Layer* with regards to the aspects of users, tasks, events and resources. A delegation protocol describes request and response message pairs from a delegator to a delegatee. The core operations described in figure 4.7 are defined in the following :

The delegator issues a delegation request and sends it to the delegatee (*InitDelegReq()*). The first step consists of negotiating the request based on the request specifications such as deadline, evidence and workload (*NegotiationReqIssue()* and *NegotiationReqResponse()*). The delegatee will then decide whether to perform the requested operation and will send the response to the delegator (*InitDelegResponse()*). If the request is declined, the delegator will check whether another delegatee exists and then will renew his request (*RedefineDelegReq()*).

If the request is accepted, the delegatee will acquire delegated privileges issued by the delegator (*DPReqIssue()* and *DPReqResponse()*). Once delegated privileges are acquired,

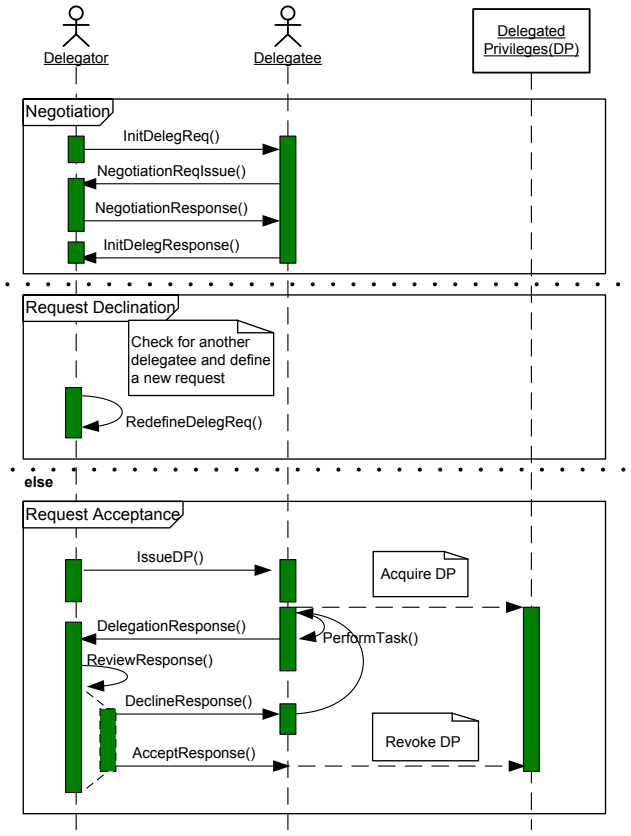

Figure 4.7: Delegation protocol negotiation-based

the delegatee starts performing the delegated task and then sends as a response the execution outcome to the delegator to review it based on evidence specifications defined in the negotiation step (*DelegationResponse()*).

The reviewing step will lead to the acceptance or the declination of the delegation response, and so the re-assignment or the acceptance of the delegated execution task (*DeclineResponse()* and *AcceptResponse()*). Finally, the acceptance step will complete the task and revoke the delegated privileges.

4.5.6 Summary

Human interactions define the involved users (principals) taking part in the delegation process. Interactions is a mean to support heavily human-centric workflows. We observed that such interactions is defined at both organisational and policy level inquiring flexibility and access control requirements. In this section, we described how such requirements have to be combined together to secure a delegation request. Securing delegation depends on the delegated privileges. It manages how permissions can be granted to the delegatee and when a revocation will happen to avoid access abuse or conflicts. Privileges managements is based on access control systems and will be detailed in the next section.

4.6 Access Control Over Task Delegation in Workflows

In this section, we aim to reason about task delegation from resources perspectives (material resources) to analyse and specify task delegation constraints while accessing workflow's data (e.g. permissions on business objects). Any mechanism that is used to support task delegation is based on workflow specifications and user authorisations information. In current workflow management systems, the RBAC model is widely adopted, where system administrators assign roles to users. Actually, it is more convenient for administrators to manage roles than to manage users directly [BS00, XLfC].

Delegation is a mechanism that permits a user to assign a subset of his assigned authorisations to other users who currently do not possess the authorisation. In our work, we define a task oriented access control model based on the RBAC model. We aim to support security delegation constraints with regards to potential delegatees and their required privileges.

4.6.1 Task execution model

We define a task execution model using a UML activity diagram composed of three main activities : *Initialisation*, *Processing* and *Finalisation*. During the initialisation of the task, a task instance is created and then assigned to a user. During task processing, the assigned user can start or delegate the task which gathers all operations and rights over the business objects related to task's resources. Finally, the task finalisation would notice the workflow management system that the task is terminated, where termination defines completeness, failure or cancellation.

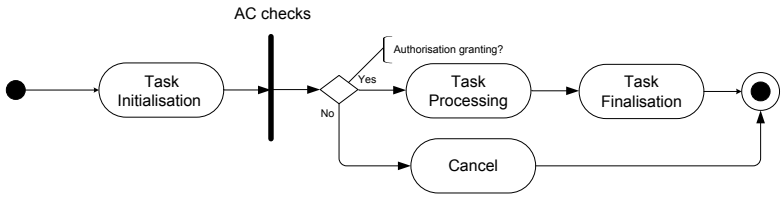

Figure 4.8: Task execution model

Seeing the task as a block that needs protection against undesired accesses, the activity diagram includes an access control (AC) transition that is in charge with granting or not the access to the task. AC checks defines the transition from the creation of a task to its assignment to a user. This assignment will lead to the processing or the cancellation of the task. Cancellation can be triggered when the assigned user does not fulfill the required authorisation to execute the task instance.

The AC transition defines the on-time authorisation supporting task execution. It defines a relationship between users, task instances and authorisation instances. An authorisation instance defines the permission P needed to execute operations on business objects to carry out a task (see definition 7).

Definition 7 (Permissions) P *is a set of permissions.* P *defines the right to execute an operation on a resource type. A permission* p *is a pair* (f,o) *where* f *is a function and* o *is a business object. We note :* $P \in F \times O$ *where* F *is a set of functions and* O *is a set of business objects.*

For instance, the task T7 "Determine Judicial Authorities" requires a permission that defines functions *add()* and *modify()* to access the MLA business object (see table 2.1). Therefore, the assigned user Cathy member of role Judge needs to be authorised to access T7 task resources.

We aim by this AC specifications to motivate to two sides of task requirements, namely material and human resources. Once the task's resources requirements are identified, an access control has to be defined to check the authorisation of the initiated user. An authorisation makes the explicit binding between a user, a task resource (object) and his rights over it (action). Such authorisation information may be specified using a simple access control list where users may perform tasks for which they are authorised.

4.6.2 Task-oriented access control model

We propose a Task-oriented Access Control (TAC) model to support authorisation requirements in workflow systems (see figure 4.9). Authorisation information will be inferred from access control data structures, such as user-role assignment and task-role assignment relations. We model permission assignment relations for tasks and roles in order to support both human and material resources. The tuple (P,T,R) specifies TRA, TPA and RPA many-to-many relationships which are specifics to the task execution context. The remaining relations are generic relations based on the RBAC model [SCFY96].

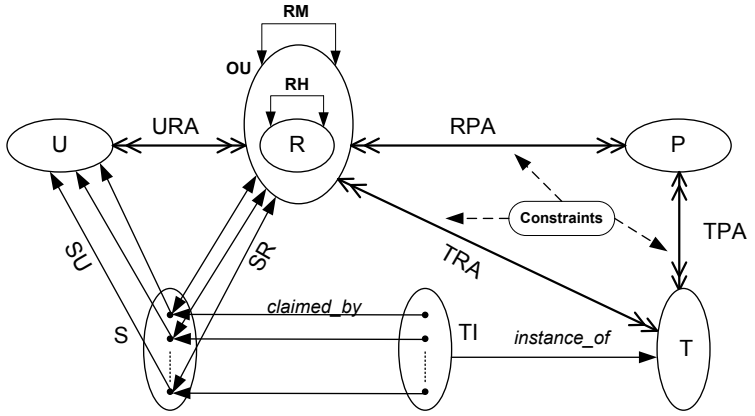

Figure 4.9: Task-oriented access control (TAC) model

Formally, we define sets *U, R, OU, T, P, S* and *TI* as a set of users, roles, organisations units, tasks, permissions, subjects and task instances, respectively. Note that we use a subject to define the time a user selects roles for a session. It manages the task

instantiation assignment ($claimed_{by}$) in order to create a user's current active role set.
We define RH (Role Hierarchy), where RH is a partial order on R. r_i and $r_j \in R$, RH denotes that r_i is a role superior to r_j, as a result, r_i automatically inherits the permissions of r_j.

We define RM (Role Mapping), where $RM \subseteq OU_i \times OU_j$ with OU_i and OU_j two organisations units. RM defines external roles accessing distributed resources cross-organisations [FPP+02]. It provide a decentralised access control mechanism where externally known roles are publicly available :
$r_k \in OU_i$ and $r_l \in OU_j$, RM denotes that r_l is a role mapped to r_k, as a result, r_l automatically inherits the permissions of r_k.

a - Definitions of map relations

- $URA \subseteq U \times R$, the user role assignment relation mapping users to roles they are member of.
- $RPA \subseteq R \times P$, the permission role assignment relation mapping roles to permissions they are authorised to.
- $TPA \subseteq T \times P$, the task permission assignment relation mapping tasks to permissions. This defines the set of permission required to execute a task.
- $TRA \subseteq T \times R$ the task role assignment relation mapping roles to tasks they are assigned to.

b - Definitions of functions

- $SU: S \rightarrow U$ a function mapping a subject to the corresponding user.
- $SR: S \rightarrow R$, a function mapping each subject to a role, where $SR(s) = r, (SU(s), r) \in URA\}$ and s having as permission $p|(r, p) \in RPA\}$.
- $instance_{of}: T \rightarrow TI$, a function mapping a task type to its task type instances.
- $claimed_{by}: TI \rightarrow S$, a function mapping a task instance to a subject to execute it : s = $claimed_{by}$(t_i) where :
  $\{t_i = instance_{of}(t), (r, u) \in URA|(SR(s) = r \bigwedge SU(s) = u), (t, r) \in TRA\}$.
  ,

c - Definitions of constraints

Here we discuss Separation of duty (SoD) and Binding of duty (BoD) constraints. We define an exclusive relation between tasks for SoD, and binding relation between tasks for BoD as follows :

TT_{SOD} : $\{(t_i, t_j) \in T \times T \mid t_i$ *is exclusive with* $t_j\}$

$TT_{BOD} : \{(t_i, t_j) \in T \times T \mid t_i \text{ is binding with } t_j\}$

If $(t_i, t_j) \in TT_{SOD}$, then t_i and t_j cannot be assigned to the same user. For instance, (T4, T6) $\in TT_{SOD}$, where users with the role *Prosecutor* must be different (see figure 2.5).
If $(t_i, t_j) \in TT_{BOD}$, then t_i and t_j must be assigned to the same user.

d - Contributions and motivations

The main contribution is to specify the task assignment conditions based on the RPA and TPA requirements (see definition 8). Actually, two conditions have to be verified to satisfy TRA relation. The first condition is related to task resources requirements. The user's permissions defined in RPA needs to satisfy the permissions defined in TPA. If this condition is satisfied, the task is executed if and only if the user is assigned to it. Basically, having permissions to execute a task but not being assigned to it will not satisfy those conditions and, therefore, will deny the access to task resources.

Definition 8 (Task assignment conditions) *A task can only be assigned to a role if and only if :* $(t, r) \in TRA \Rightarrow \{p \in P | (t, p) \in TPA\} \subset \{p | (r, p) \in RPA\}$.

Returning to the motivating example, T2 "Check Request" is assigned a set of permissions (*query(), update()*) via TPA in order to carry out this task. The user Alice with the role *Prosecutor* is assigned to T2 since Alice is authorised to execute it based on the authorisation policy definition. However, if we consider another *Prosecutor* from Eurojust B such as user Claude, he is not allowed to execute T2. Claude's permissions do not fulfill T2 requirements since the user-task assignments is not defined in the policy (see figure 2.5, table 2.1), which is a condition for the TRA relation.

4.6.3 Access control over task delegation using TAC

Delegation is a mechanism that permits a user to assign a subset of his assigned authorisations (privileges) to other users who currently do not possess it. We remind that the user who performs a delegation is referred to as a "delegator" and the user who receives a delegation is referred to as a "delegatee".

Definition 9 (Delegation Relation) *We define a delegation relation* $DR \subseteq T \times U \times U \times 2^{DC}$ *where* T *a set of tasks,* U *a set of users and* DC *a set of delegation constraints. A task delegation relation is defined as* DR = (t,u_1,u_2,{DC}), t *is the delegated task and* t $\in$ T, u_1 *the delegator and* u_2 *the delegatee* $\in$ U.

For instance, delegation constraints can be related to time or evidence specifications. Moreover, a role hierarchy (RH) defines the delegation relation condition in a user-to-user

delegation. Returning to the example, we have :
(T3,u_1:Prosecutor,u_2:Assistant,{RH,5days}) $\in$ *DR*.

The TAC model defines the list of potential delegatees (RPA) that may satisfy the delegated task requirements (TPA). We define a method for access control over task delegation using TAC. We present an algorithm to describe how valid delegatees are checked and whether they need delegated privileges grant (see algorithm 3).

Algorithm 3: Optimised computation for delegatees and privileges

Require : DR // Delegation relation
u_1, u_2 // users $\in U$
r_1, r_2 // roles $\in R$
t_i, t_j // tasks $\in T$
$DR = \emptyset$ /*Initialisation of DR*/
$\{(u_1, r_1), (u_2, r_2)\} \subseteq URA$ /*User role assignment */
$\{(r_1, p_{r1}), (r_2, p_{r2})\} \subseteq RPA$ /*Role permission assignment*/
$(t_i, p_{ti}) \in TPA$ /*Task permission assignment with $p_{ti} \subset p_{r1}$ */
$(t_i, r_1) \in TRA$ /*Task role assignment*/
$SU(s_1) = u_1$ /*Mapping a subject to the corresponding user*/
$SR(s_1) = r_1$ /*Mapping a subject to the corresponding role*/
$SU(s_2) = u_2$
$SR(s_2) = r_2$
$t_{i1} = instance_{of}(t_i)$
$t_{j1} = instance_{of}(t_j)$
$s_1 = claimed_{by}(t_{i1})$ /*t_{i1} assigned to s_1*/
Precondition :
($\nexists\ t_j1$ | (TT_{SOD}(t_i,t_j), $s_2 = claimed_{by}(t_{j1})$) AND ($TT_{BOD}$($t_i$,$t_j$), $s_1 = claimed_{by}(t_{j1})$))
Postcondition :
$p_{ti} \subset p_{r2} \Rightarrow s_2 = claimed_{by}(t_{i1})$ /*t_{i1} delegated to s_2*/
$p_{ti} \not\subset p_{r2} \Rightarrow p'_{r2} \leftarrow p_{r2} \cup p_{ti}$ /*p'_{r2} set of permissions associated with s_2 during delegation*/
Result : DR = (t_i,u_1,u_2, {*DC*})
$DR_1 \leftarrow$ instanceOf(DR)
DR_1 = (t_{i1},s_1,s_2, {*DC*})/*Instanciation of the delegation relation*/

For instance, u_1 and u_2 are members of roles r_1 and r_2 respectively; (t_i,u_1,u_2,{DC}) $\in$ *DR iff* the delegatee u_2 has no conflict with the delegated task t_i and his permissions (granted or transferred) allows him to execute t_i. Note that we assume that the mapping between functional and security roles of users u_1 and u_2 is already verified where r_1 and r_2 are the security roles (see section 4.3).

The main contribution of this a method is to specify the delegated task assignment conditions based on the RPA and TPA requirements. Actually, two conditions have to be verified to satisfy delegation. The first condition is related to task resources requirements. The delegatee's permissions defined in RPA need to satisfy the permissions defined in TPA (see definition 8). If this condition is satisfied, then the task t_i is delegated to the delegatee u_2. However, if u_2 do not have the permission required and there is no conflicts (BoD or SoD) to execute t_i. This condition will grant the required privileges for him. The main steps in the algorithms consist of :

1. Defining the role and permission assignment for each users.
2. Instantiating the task t_{i1} and assigning it to the delegator s_1 who is the current user.
3. Checking security constraints before delegation (SoD and BoD).
4. Computing the delegatee s_2 based on his permissions assignment ($(t_i, p_{r2}) \in TPA$) or;
5. Granting privileges for s_2 based on the task instance permissions assignment ($p'_{r2} \leftarrow p_{r2} \cup p_{ti}$)
6. Defining the delegation relation instance : $DR_1 = (t_{i1},s_1,s_2,\{DC\})$.

The computation of the privileges is based on the TRA relation defined in our TAC model. Basically, we provide an optimised method to compute the least privileges to delegate based on the current requirements of the task instances and not the full requirement of the task type. The task instance is generated from the delegated task. We aim to optimise the delegated privileges based on what the delegated task instance defines. For instance, we consider the local delegation DS1 presented in the case study. The *Prosecutor* Alice delegates T3 to his *Assistant* Bob. We assume that Bob do not have the permissions to access T3 resources. Alice will grant just the permission *translate()* on the business object type "Request Document" while keeping additional privileges related to the MLA request treatment that have to be protected due to privacy reason. By computing the task instance requirements for delegation, we ensure the grant of the least privileges for the delegatee : (Assistant,translate()) $\in$ RPA.

4.6.4 Revocation

Revocation is an important process that must accompany delegation. It is the subsequent withdrawal of previously delegated objects such as a role or a task. A vast amount of different views on the topic can be found in literature [WKB07, ZAC03, HJPPW01]. For simplification, our model of revocation is closely related to the delegation model (TDM) based on the user-to-user interactions. Actually, the decision of revocation is issued from the delegator in order to take away the delegated privileges, or the desire to go back to the state before privileges were delegated. The privileges consist of the delegating of access rights provided to the delegatee. Basically, delegating access rights issued from the delegator describes the permissions given to access to the task's resources such as legal request documents in the MLA scenario.

Analogous to the motivation factors for delegation, the revocation factors can be identified depending on the delegation context. For example, a delegator can take back his tasks if his workload is being decreased or if the delegatee is not efficient anymore for performing the delegated task. In short, we identify five factors that we assume supporting our delegation approach defined in the previous section :

- **Workload** : If the task is not performed yet and the delegator workload is lightened, the delegator will take back his task.

- **Deadline** : Some delegating access rights are temporary for some security or workflow policy reasons. This involves a time constraint giving a validity time for the delegatee and revoking it afterwards. This constraint utility can avoid also a long period of inactivity of the delegatee (e.g. 5 days as a deadline to execute T3 by the *Assistant*).

- **Efficiency** : It may be advantageous to revoke a task when a new situation arises that makes performance by the earlier delegator more efficient than performance by the delegatee.

- **Organisational policies** : Goals may conflict and specific organisational policies may change generating new organisational requirements and pushing eventual revocations.

- **Task evolution** : A task may have to be redefined due to some goals redefinition or an evolution of some required competences by assigning new users or adding new roles and thus revoking unnecessary delegatees.

The operation of revocation can be defined manually when the delegator decides to revoke the selected task by removing the delegatee's privileges or automatically by bending a time constraint to delegation. This constraint will affect the delegating access rights by deriving additional requirements for the access control enforcement. For instance, a user can assume a temporary role during a delegation and so a temporary privilege. The delegating access rights will not renew it once the delegation period is over.

On a technical level, we developed an approach that support dynamic revocation [GMC09]. We leverage certification techniques when generating authorisation and subsequently embed credentials attributes for authentication and authorisation purposes. Currently, techniques like temporary certificates or certificate revocation list (CRL) are time-based, and, therefore, do fulfill time-based delegation. Additionally, we came up with a new solution to enrich our revocation approach using our event-based TDM model. This solution will be discussed in the next chapter dealing with dynamic policies supporting event-based task delegation.

4.6.5 Summary

We presented workflow authorisation constraints to support security requirements for delegation. These requirements identify the main authorisation relationships regarding users, tasks and resources in order to extend the RBAC model to support task delegation. To do so, we presented a Task-oriented Access Control (TAC) model to support access control over workflows. The novelty of this model is to reason about task delegation from human and material resources. Our TAC model is enriched RBAC specifications with additional task and resource requirements, thereby controlling delegatees assignment and task execution resources. This model will help us to specify the delegation polices within workflows afterwards.

4.7 Conclusion

In this chapter, we have presented the first part of our approach for task delegation in workflow systems. Modelling task delegation goes through different steps related to the organisation hierarchy and its policy definition. Our approach is based on our **delegation taxonomy**. It identified the main delegation constraints within workflows. Task delegation has to consider the process definition and the security properties in a workflow. This defines additional constraints for delegation in order to cope with the organisation flexibility and the authorisation policy. Basically, it involves requirements with regards to workflow and access control systems.

Moreover, the novelty of this approach relies on the separation of the various aspects of delegation, and in its portrayal as a multi-layered state machine. The interaction between different layers is triggered by delegation events. **Delegation events define the delegation process and build the task delegation model (TDM) to be integrated in the business process**. This model will guide us to define an access control model supporting delegation. In fact, delegation events ensure the appropriate authorisation to delegate or revoke a task, thereby supporting security properties. Additionally, we detailed a delegation protocol with a specific focus on the **initial negotiation** step between the involved principals where we envisaged a wide-ranging request that gives flexibility for delegation.

In addition to the organisation flexibility, we have shown that such a delegation process inquires additional requirements for security. Enormous amount of data flow cross-organisations along processes and are shared by many different users. Their security must be assured. To that end, we analysed the relevant authorisation requirements in workflow management systems. Then, we proposed the **task-oriented access control (TAC) model** based on the RBAC model. This model will grant authorisations based on workflow specifications and user authorisation information from access control systems.

In chapter 5, we consider task delegation as an advanced security mechanism supporting policy decision. We will define an approach to support dynamic delegation of authority within an access control framework. The novelty consists of reasoning on authorisation dependently on task delegation events, and specifies them in terms of delegation policies. Moreover, we will propose a technique that automates delegation policies using *event calculus* to control the delegation execution and increase the compliance of delegation changes in the global authorisation policy.

Chapter 5

Securing Task Delegation in Access Control Systems

Contents

5.1 Introduction

The contribution of the chapter is the definition of a dynamic delegation of authority approach to ensure authorisation policies in access control systems. We propose to reason on authorisation dependently on task delegation events, and specifies them in terms of delegation policies. When one of these events occurs, our access policy decision may change proactively implying dynamic delegation of authority.

We answer the interrogations defined in the chapter "Context and Problematic" related to the delegation of authority and the integration of delegation policies in existing access control systems. We observed that the delegation completion and its authorisation enforcement are specified under specific constraints. Constraints are defined from the delegation context implying the presence of a fixed set of **delegation events to control the delegation execution**. Our objective will be to define and specify delegation policies in an automatic manner. To do so, we have to investigate the potential of delegation events to ensure a secure task delegation within a workflow. **Securing delegation involves the definition of authorisation policies which are compliant with the policy of the workflow**. Therefore, these delegation events will imply appropriate authorisations on the delegatee side for further actions as well as contain specific constraints for those actions. To that end, we leverage our task delegation model that forms the basis of what can be analysed during the delegation process in terms of monitoring and security. Dealing with that, we will propose a technique that automates delegation policies using event calculus to control the delegation execution and **to increase the compliance of all delegation changes in the authorisation policy**.

The remainder of this chapter is organised as follows. Section 5.2 introduces a motivation scenario based on the MLA example to secure task delegation in access control systems. Section 5.3 analyses delegation events impact on authorisation policies, it presents a framework ensuring dynamic delegation of authority and it shows how proactive policy decisions will be implemented on existing access control framework. Finally, we present an approach for the integration of delegation policies in section 5.4. Using *Event Calculus*, we propose a technique that gathers specific events and integrates delegation policies to control the delegation execution and to ensure the compliance of delegation changes in the global authorisation policy.

5.2 Modelling Task Delegation Using Access Control Systems

In this section, we introduce a delegation scenario based on the MLA example to secure task delegation in access control systems. We motivate the use of the Task-oriented Access Control (TAC) model for an access control framework and detail its deployment within a workflow. We mainly focus on the involved users to ensure the delegation of authority in access control systems based on workflow specifications. To that end, we present a modular architecture for access control systems and show how a delegation request will interact between both workflow and access control components.

5.2.1 Context for dynamic delegation policies

We present an instance of the MLA process supporting task delegation locally. User Alice member of role *Prosecutor* is assigned to execute the MLA request in Eurojust A.

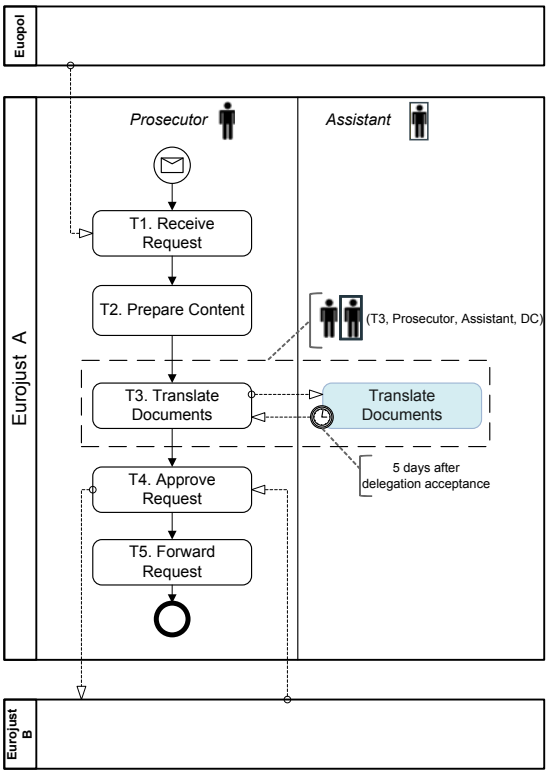

Figure 5.1: Local delegation scenario from the MLA example

In this scenario, the task "Translate Documents" T3 is originally only accessible by *Prosecutor* Alice, a fact defined in the workflow policy. This task is a long-running task and is expected to take 5 working days to complete. Alice is overloaded (lack of resources), and will delegate it to User Bob. Bob is a member of role *Assistant* and is a subordinate to *Prosecutor* in the organisation hierarchy. Delegation criteria is based on the role hierarchy (RH) of Eurojust, where the *Assistant* Bob is a subordinate to the *prosecutor* Alice. The delegation relation $DR = (T3, Alice, Bob, RH)$.

Task delegation inquires security requirements to ensure delegation of authority in access control systems. A policy can be defined as a level of defining access to task resources. We define an authorisation policy P for the MLA process. During delegation, the policy P is updated so that User Bob is now allowed to complete task T3. As such, Alice and Bob are here the delegator and the delegatee, respectively. Bob claims the task and issues an access control request to execute it. Authorisation policy enforcement mechanism is vital for supporting delegation in long-running tasks. For that reason, we need to support specific interactions based on delegation events that would be automatically captured, and conveyed back to the requestor for appropriate actions within the access control system.

At present, we can enforce delegation access rights via policy adaptation (i.e. permitting the delegatee to perform the delegated tasks). Subsequently, we need to update the delegation relation *DR* in the workflow policy once a delegation event is triggered. It consists of adding a new policy authorisation constraint for the delegated user. If this constraint changes due to a policy adaptation (e.g. a task revocation event), a new response needs to be conveyed to the delegatee dynamically. Moreover, we have to compute automatically the new delegation within the existing policy *P*. This computation will ensure compliancy by adding valid rules for delegation in the workflow's policy using access control systems.

5.2.2 Access control framework

An access control has to be defined to check the authorisation of the initiated user (the *subject*). An authorisation makes the explicit binding between a subject, a task resource (object) and his rights over it (action). This binding is defined based on the TAC specifications. It includes entities defined in the four main relations of TAC namely *URA, TRA, RPA* and *TPA*. Subsequently, an authorisation expresses a user's permissions on a task's resources, where a permission is the right to execute an action on a resource. The definition of the authorisation has to be specified in a policy (see definition 10).

Definition 10 (Policy) *We define a policy* $P \subseteq target \times rule \times 2^C$*, where* target *defines where a policy is applicable,* rule *is a set of rules that defines the policy decision result, and* C *the policy constraints set that validates the policy rule.*

The pseudo formal expression of a target is :

```
<Authorisation>
<Subject>[role]
<Resource>[object]
<Action>[operation]
<Task>[task type]
< /Authorisation>
```

The rule effect is the authorisation decision. It can return as a result a permit, a deny or an indeterminate request. Constraints are related to the workflow authorisation specifications. For instance, the separation of duty (SoD) is a constraint for users assignment and resources access.

An example of a policy where the decision returns "Permit" for a subject member of role *Prosecutor* on the task T1 "receive request" (see figure 5.1) is :

```
<Poliy>
<target>[Prosecutor1,MLA1,read,T1]
<rule>[Permit]
<C>[none]
< /Poliy>
```

Existing access control systems in the domain of role-based access control (RBAC) define an approach for restricting system access to authorised users. Here, we restrict task's resources to undesirable users. To do so, we present an access control framework (ACF) to support authorisation requests based on our access control model (TAC) specifications. The main components are described as follows :

- **Policy Manager** : It allows an administrator to define policies. Through a graphical user interface, the administrator can navigate through the policy document, select document elements (e.g. targets, authorisation rules, obligations), and specify values for selected elements. For instance, an administrator defines an authorisation policy P with decision *permit* on target task T3 for subject Alice with role *Prosecutor*. In the context of delegation, delegation policies will embed delegatee's attributes for authentication and authorisation purposes.

- **ACF** : An access control framework (ACF) is defined as a set of software components which accept requests to access resources, analyse these against policies representing actual access rights to resources, and return a response based on this analysis. To illustrate the original architecture of an ACF, a request is issued by the requestor, which is received by the *Receiver* component in ACF. This is then sent to the *Analyser* component that queries policies stored in a policy database. A response is generated by the *Responder* component, which defines a decision (permit, deny, or notapplicable) that is sent back to the requestor. It should be noted, that the above appears asynchronous for the requestor; they provide the request, and a response is produced.

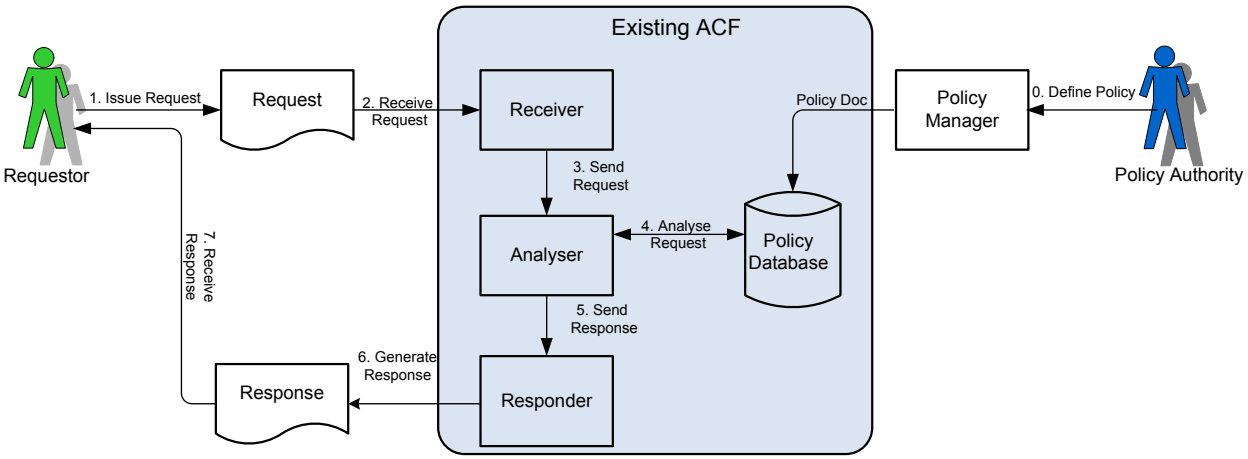

Figure 5.2: Access control framework

5.2.3 General control process

The general control process illustrates the message flow between access control and workflow components. It defines what workflow operations a user can perform. We present a UML sequence diagram that illustrates whenever a subject, as an instance of a user, claims a task instance. Basically, a task assignment can be defined in the worklist by the workflow system [WFM99]. In addition, a subject can claim a task access request without being initially assigned, thereby involving dynamic checks of his authorisation credentials (see figure 5.3).

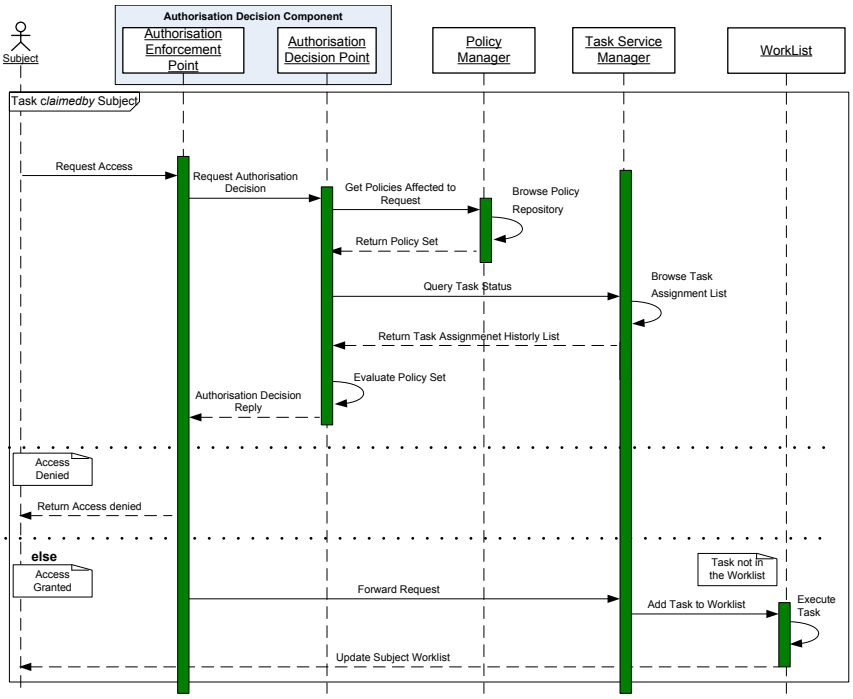

Figure 5.3: Task assignment sequence diagram

Whenever a subject issues a claim request to perform a task that is protected by the control components, all requests will be intercepted by an authorisation enforcement point. The authorisation point is not capable of making an access decision on its own. Therefore, the authorisation enforcement point will request a decision from the authorisation decision point.

To make a decision, the decision point queries the policy manager for all policies that are affected to this request. Thus, all policies that apply to the identity of the subject, his role, and policies related to the requested task instance in the corresponding workflow are prompted from the policy manager. The policy manager will browse through its policy repository and returns the set of affected policies to the decision point.

Some of these policies may contain dynamic constraints depending on the current task state and the process history. To get this information the decision point will ask the Task Service Manager (TSM) for the current process state. The TSM retrieves all relevant state information and returns it to the authorisation decision point. The state

information tell us about the availability and the state of progress of the task life cycle based on its triggered events (e.g. create, complete). If the task is already assigned and being executed, no further assignment will be allowed.

Now the decision point is able to evaluate the static and dynamic policies. Depending on the rule effect the binary authorisation decision is returned, i.e. access denied or access granted. In the case the access request is denied, the enforcement point will mediate this decision to the subject, for instance as some error message. In the case access was granted, the intercepted request is forwarded to the TSM hosting all task instances. The TSM will initialise the task and passes the original request to it. In case that the claimed task is not in the subject worklist, the TSM will update the worklist and the task will be performed and will send back potential results to the subject.

We made some assumptions and simplifications in the task assignment process. For instance, we assume the subject's identity and his role attribute are already known in the control architecture. Otherwise the subject has to identify himself against some trusted authority or authentication mechanisms. The first one could issue some kind of identity certificate that can be verified by the control architecture, the later one would request the subject's password before the access request will be accepted by the authorisation enforcement component. Note that authentication issues will be discussed and implemented in the "Deployment Environment" chapter.

5.2.4 Delegation protocols

We introduce delegation protocols to support both *push* and *pull* modes. Delegation protocols define two different models that depict the dialogue between a delegator and a delegatee during a secure task delegation. We model the different protocols using UML sequence diagrams. Delegation protocols will ensure delegation of authority in access control systems.

a - Pull Delegation (TDM1)

The pull delegation model (TDM1) is based on a direct allocation of the task through a delegation without any notion of role. This model associates implicitly an authorisation to a subject.

When a subject holding a task initiates a delegation process, then the following procedure manages it :

1. First the delegator is sending a request for delegation to the Delegation Component for a specific task (delegated task) and a specific subject (the delegatee).

2. The Delegation Component checks with the help of the Authorisation Component (AC) if the delegator can actually delegate and the delegatee can receive the request.

    - a) The AC first retrieves the attributes affecting the policy and conducts an initial evaluation regarding the delegator's right to delegate. This is due to the fact that certain task assignments are exclusive and are not allowed to

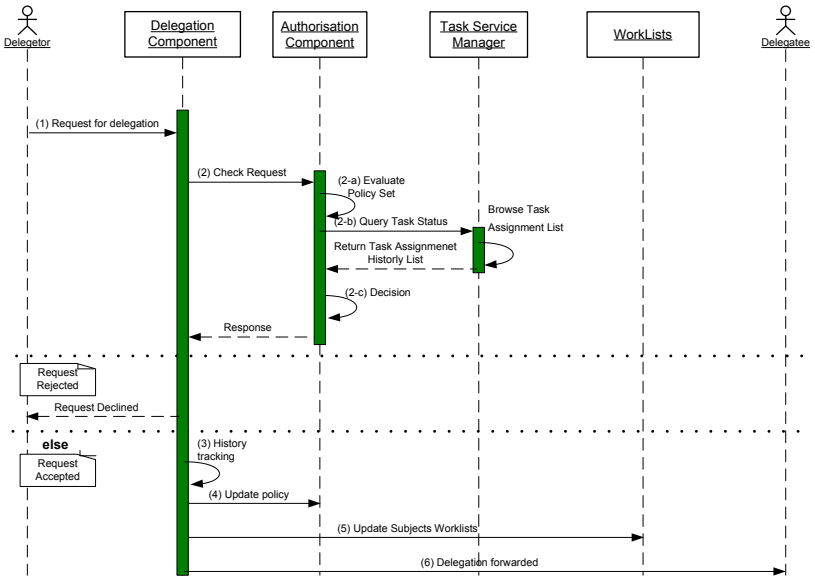

Figure 5.4: Task delegation pull model

be delegated. In the context of an access control policy, it is defined as an obligation to a rule effect (see section 5.4.4).

- b) The AC checks then the task status with the Task Service Manager (TSM) component which browses the current task assignment list to check the availability of the task (i.e. executed, aborted)
- c) The AC receives the history list from TSM. Finally, the AC sends a response to the delegation component based on the intermediate results received.

3. The Delegation Component then keeps track of the current delegation within internal history records.

4. The delegation component updates the appropriate policy in the policy repository.

5. The delegation component updates the appropriate worklists (delegator and delegatee) if the delegation is related to a task instance.

6. The delegation request is forwarded to the designated delegatee.

In case of transfer delegation, the given authorisation from the delegator's set are removed from the policy repository.

b - Push Delegation (TDM2)

The model TDM2 is based on an allocation of the task through a delegation to a role and not directly to a subject.

When a subject holding a task initiates a delegation process, then the following procedure manages it :

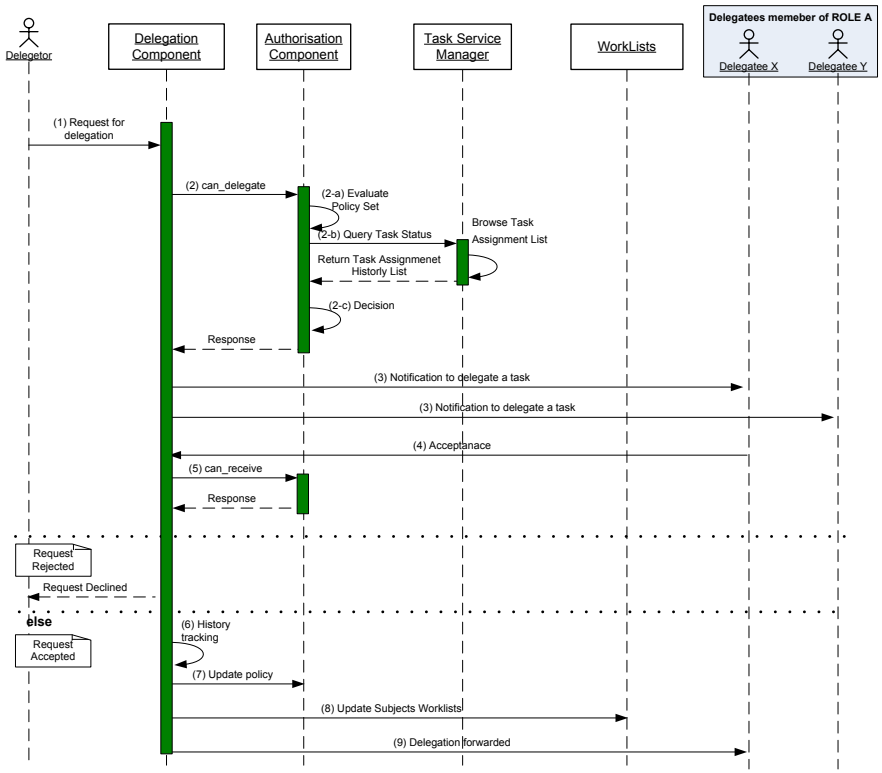

Figure 5.5: Task delegation push model

1. First the delegator is sending a request for delegation to the delegation component for a specific task and a specific role (Role A).

2. The delegation component checks with the help of the Authorisation Component (AC) if the delegator can actually delegate based on his policy attributes, then with the task service manager regarding the delegated task status.

3. The delegation component notifies all the subjects belonging to the role (Role A) of the availability of the task.

4. The first one to respond is allocated with the task.

5. The delegation component checks with the help of the AC if the delegatee can actually receive the task.

6. The delegation component then keeps track of the current delegation within internal history records.

7. The delegation component updates the appropriate policy in the policy repository.

8. The delegation component updates the appropriate worklists (delegator and delegatee) if the delegation is related to a task instance.

9. The delegation is forwarded to the designated delegatee.

5.2.5 Access control enforcement

The access control model handles normally a delegated task and does not need to be modified. The expression of the associated authorisation for delegation is updated in the existing policy. This way provides several advantages :

- In our TDM1 model, the delegatee is not granted with a new role. The delegation process does not need further control of the permission; the access control model handles normally a delegated task and does not need to be modified. The delegatee inherits his own permissions.
- In our TDM2 model, the link of the authorisation with the role is kept. It allows us to reuse the established access control model role-based.

Based on the case study, we exemplify here our work. The TDM1 model delegation answers the requirements defined in the local delegation scenario (DS1). User Bob exists in the delegatees list of the delegator Alice and is directly assigned to the task T3. The formal expression of the authorisation in TDM1 is :

```
<Delegation>
<Issuer>[Subject:Alice]
<Receiver>[Subject:Bob]
<Task>[T3]x[TaskId]
<Authorisation>[Authorisation Instance]
< /Delegation>
```

The TDM2 model answers the requirements defined in the global delegation scenario (DS2). User Alice needs to delegate based on the role equivalence (role mapping). In Eurojust B, user Claude with role *Prosecutor* is one of the potential delegatees that may accept this delegation request. User Claude is the first to accept the request. The formal expression of the authorisation in TDM2 is :

```
<Delegation>
<Issuer>[Subject:Alice]
<Receiver>[Subject:Claude]x[Role:Prosecutor]
<Task>[T2]x[TaskId]
<Authorisation>[Authorisation Instance]
< /Delegation>
```

If the request is granted by the delegation component then this delegation message will be forwarded to the delegatee that will include the given authorisation to its own set. The delegatee can claim achieving the task as he is provided with the same access rights functions (permissions) as the previous owner of the task (the delegator). The authorisation instance is explicitly related to the role of the first owner of the authorisation to execute an instance of the task. The attributes defined to express authorisation in TDM1 and TDM2 are defined based on the TAC model (see chapter 4).

5.2.6 Summary

In this section, we have presented how delegation will occur during the execution of an MLA scenario. We detailed two different protocols for delegation to support human interactions (push and pull modes). Additionally, we noticed that the access control enforcement has to take into account new changes due to delegation in order to update the existing policy. The computation of policy changes will be discussed in the next section.

5.3 Event-based Task Delegation Policies

In this section, our concern is to identify events that enforce policy changes. We define an approach to support dynamic delegation of authority within an access control framework. The novelty consists of reasoning on authorisation dependently on task delegation events, and specifies them in terms of delegation policies. When one of these events changes, our access policy decision may change proactively implying dynamic delegation of authority. Existing work on access control systems remain stateless and do not consider this perspective. We highlight such limitations, and propose a task delegation framework to support proactive enforcement of delegation policies.

5.3.1 Problem statement (part I)

A policy can be defined as a level of defining access to task resources. We define an authorisation policy P for the MLA process (see figure 5.1). During delegation, the policy P is updated so that User Bob is now allowed to complete task T3. User Bob claims the task and an access control request is granted to execute the delegated task. After two days, Alice interrupts her sick leave and returns to work. Once again, Alice is able to claim the task. Due to qualification considerations, it is decided that Alice should complete the task, and that Bob should revoke his actions, and free the task. The policy P needs to be updated to reflect that only Alice has access to the task. As such, the original request made by Bob would now evaluate to a deny decision for access.

In traditional access control frameworks no mechanism exists that would alert User Bob of this fact automatically [BCFM00, SRS+05]. Accordingly, it is not possible to revoke a previous response given to an access control request. Moreover, a manual review of the current access control rights and task executions is costly, labor intensive, and prone to errors. With a proactive mechanism, when the policy changes to reflect delegation events, the delegatee will be informed proactively. For that reason, we need to support specific interactions on the access control architecture that they run on. Specific interactions are meant to be task delegation events that would be automatically captured, and conveyed back to the requestor for appropriate actions. Delegation policies will reflect those actions dynamically based on specific events that will be discussed in the next section.

5.3.2 Security requirement for delegation

We define delegation transitions as events ruling delegation behaviour. We have enriched our TDM with additional events supporting delegation requirements such as pull/push mode and grant/transfer kind. The internal behaviour based on events may be a source to a policy change, thereby introducing advanced security requirements in access control enforcement systems. For instance, events like *delegate* or *revoke* inquire access control enforcement supporting new delegatee privileges. From the TDM, we analyse security requirements that need to be taken into account to define event-based delegation policies (see figure 4.6).

The requirement statements that need to be taken into account to enforce delegation policies are defined as follows :

- **Delegation mode** : It defines how delegation request is issued : the pull and the push mode (see section 4.5.3).
- **Delegation type** : It may be classified into grant or transfer (see section 4.5.3).
- **Delegation of authority** : It permits to a delegator to assign a subset of his assigned privileges to a delegatee who currently does not possess the required authorisation to execute the task. For instance, *delegate* is an event that will trigger task delegation, thereby updating a policy to enforce access control for a delegatee.
- **Access control enforcement** : It permits dynamic policy enforcement. For instance, *revoke* implies the revocation of delegated privileges where the delegator will take the control back on his assigned task and, therefore, cancel the previous policy decision.

The specified *events* define the condition to validate the policy decision effect. An event change may inquire a policy decision change. In the following, we classify delegation events and identify the relationship between delegation events, delegation criteria and policies decision change (see table 5.1).

Delegation Events	Push Delegation		Pull Delegation		Policy Decision Change
	Grant	Transfer	Grant	Transfer	
u1:delegate	✓	✓	✓	✓	✓
u2:accept	✓	✓			✓
u1:cancel	✓	✓			
u2:execute	✓		✓		
u1:validate	✓		✓		
u1:revoke	✓		✓		✓
U:fail		✓		✓	
U:complete		✓		✓	

Table 5.1: Delegation policies changes based on events

We do believe that events such as *accept* or *revoke* are a part of delegation policies and have a direct impact on delegated authority, thereby inquiring dynamic enforcement

of policies. In table 5.1, a grant delegation using a push mode is based on events : *u1:delegate*, *u2:accept*, *u1:cancel*, *u2:execute*, *u1:validate* and *u1:revoke*. In this case, delegation policies changes when *u1:delegate*, *u2:accept* and *u1:revoke*. From our motivation scenario, we present two examples to explain the table 5.1 :

Example 1 : *revoke* event is defined in both push and pull modes. It supports grant delegation, where a delegatee needs to wait for the validation from the delegator. This event will enforce a policy change and terminates authorisation policy for the delegatee. The revocation leads to the completion of the task, and the revocation of the delegated privileges. For instance, Bob work is cancelled by Alice and then his delegated authority is no more valid in the policy.

Example 2 : *fail* event is defined in both push and pull modes. It supports transfer delegation, where a delegatee terminates the task by himself without validation. Defined policy will take effect until the termination of the task during transfer delegation, where no new updates are required since all the task privileges are transferred to the delegatee.

Returning to the example 1, we can observe a dynamic policy enforcement during delegation. We define a delegation policy as following :

Definition 11 (Delegation Policy) *Let P a global authorisation policy :* P = (target,rule,C)*, we define a delegation policy* P_D = ($target_D$,$rule_D$,C_D)*, where* $target_D$ = DR *: the delegation relation,* $rule_D \subset$ rule*,* $C_D \subset$ C *and* C_D = DC $\bigcup$ events *where DC the set of delegation constraints.*

Initially, T3 is delegated to Bob and the delegation policy for T3 : P_D *= (DR,permit,{RH, 5 days,delegate}).* In the meanwhile, User Alice is back to work before task is done and would like to cancel what was performed by User Bob so far. Alice is once again able to claim the task and will cancel the policy effect (permit) for Bob. The event *revoke* will be updated in the policy, and a notification (deny) is then conveyed back to Bob for appropriate actions. Thus, the delegation policy for T3 needs to be updated and P_D = *(DR,deny,revoke).*

5.3.3 A secure framework for task delegation

In this section, we develop a framework to support secure task delegation. We present a modular architecture ensuring dynamic delegation of authority and show how proactive policy decisions will be implemented on existing access control frameworks (ACF). In the context of delegation, when a request is issued, it is stored along with details of how to inform the requestor (the delegatee) if the policy decision to the request changes. When a policy is changed, previous requests are re-evaluated, and the requestor is informed that his access rights have changed. To support this approach, we propose an extension to the ACF architecture (see figure 5.2) that permits proactive enforcement of policies, based on delegation events that would alter previous policy decisions (see table 5.1).

a - Architecture overview

We describe the main components of the task delegation framework supporting proactive policy enforcement. We detail what parts were changed and what the new extended architecture looks like (see figure 5.6).

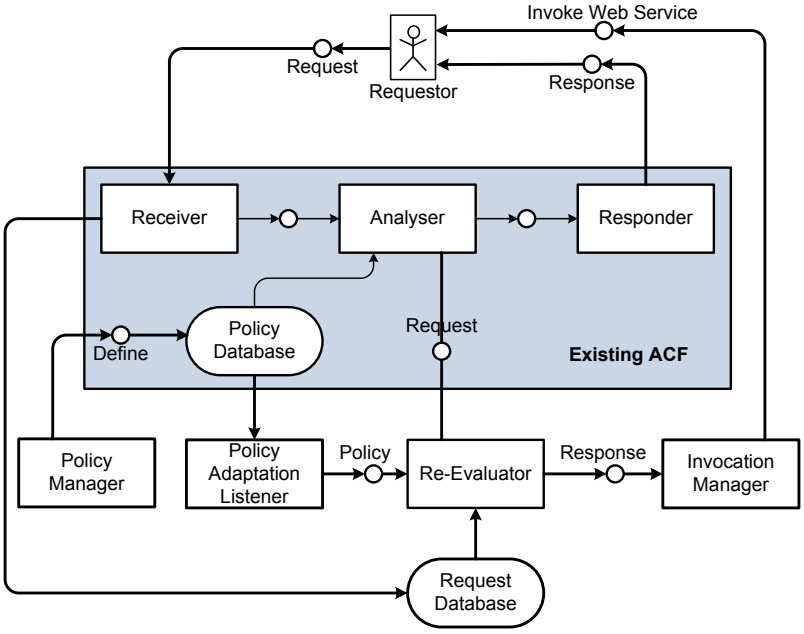

Figure 5.6: Architectural extensions supporting delegation policies

- **Policy Manager** : see section 5.2.2.
- **ACF** : see section 5.2.2.
- **Dynamic Policy Enforcement Component** : It implements our approach to support proactive policy decisions. We extend the ACF architecture with additional components related to the policy database. When the *Receiver* receives a request, it sends this to a Request Database that stores this request. A *Policy Adaptation Listener* component polls the *Policy Database* and sends an event to a *Re-evaluator* component when a policy has changed (see delegation events that change policy in table 5.1). This queries the *Request Database* to retrieve the previous requests made, and sends this to the *Analyser* component for re-evaluation. The *Analyser* then sends back a new response to the *Re-evaluator*, which queries the *Request Database* to see if this is different to the response given to the request being analysed. If this is a different response, the *Invocation Manager* component invokes the "contact point" provided by the requestor (and stored in the Request object) with the new response.

b - Architecture requirements

On an **architectural level**, as requests are required to be re-evaluated upon a policy change, a storage mechanism of previous requests and the given response are needed. If a previous request is re-evaluated and a different response to the one stored is produced, the ACF must inform the requestor of the new result. Thus, a mechanism must exist that triggers a re-evaluation when it detects a policy change. These effects of the policy change would be automatically captured, and conveyed back to the requestor for appropriate actions. In addition, an invocation component is needed that actually marshals this information to the requestor.

On a **language level**, the approach would require new constructs to describe the invocation method that the ACF can use to contact the requestor. As such, this acts as a "contact point" for the requestor. In a service-oriented architecture (SOA), this contact point could consist of the endpoint of a service that could be invoked (see the Invoke Web Service in figure 5.6). Subsequently, all access policies must be centralised and referenced by the SOA architecture which is protected. We give an SOA a single point of access and we let the services register with our ACF. Since services are essentially black boxes, we define how to contact them and to sort out what it means when policy changes.

On a **technical level**, the *Policy Manager* generates policies and subsequently embed credentials attributes for authentication and authorisation purposes. Credentials providers such as certification of authorities issue digital certificates to the requestor in order to compute his request by the ACF. At this stage, the *Receiver* component acts as a policy enforcement point to perform access control by making decision request and enforcing decisions. For instance, an Attribute Certificate (AC) is issued to the delegatee for authentication and authorisation purposes [CO02]. AC will ensure integrity, protection and non-repudiation through a digital signature. The *Receiver* gets his attributes certificate and checks his permissions afterwards. The retrieved attributes are validated against the policy (e.g. subject attributes, validity time). Once the delegatee has been successfully authenticated, he will attempt to perform specified actions on task resources. At each attempt, the *Receiver* passes the access request to the *Analyser* to decide. Decisions results (permit, deny, or not applicable) are then sent via the *Responder*.

A new re-evaluation of a policy defines new attributes for further request with regards to policy changes. For instance, a revocation implies the cancellation of the issued attribute certificate for the delegatee previously. Currently, techniques like temporary certificates or certificate revocation list (CRL) are time-based, and, therefore, do not fulfill event-based requirements. As a first solution, a service is invoked to contact the delegatee and based on the mutual agreement between delegation principals, appropriate actions will take place (e.g. the cancellation of his work and the log off from the system).

5.3.4 Summary

In this section, we have architected a solution based on existing access control frameworks to support delegation policies. We have also presented at a conceptual level the different components that are necessary to support dynamic policy enforcement. Delegation policies may change according to specific events. We defined the nature of events based

on task, and described their interactions with the policy decisions. When relevant events occur, we defined how they are detected and how to interact with them. In this context, we proposed an extension to an access control framework (ACF) architecture that ensures delegation policies enforcement, thereby supporting dynamic delegation of authority.

The next stage of our work is the computation of delegation policies that will be integrated in the existing policy. We will propose a technique that automates delegation policies using event calculus.

5.4 Integrating Event-based Delegation Policies

In this section, we develop a solution for the integration of delegation policies into existing policies. We aim to reason about delegation events to specify delegation policies dynamically. To that end, we use our event-based task delegation model (TDM) to monitor the delegation process. In order to control the delegation behaviour and to specify its authorisation policies dynamically, we gather relevant events that will define both the task execution path and the generated policies for the delegation of authority. Using *Event Calculus* (EC), we propose a technique that automates delegation policies to control the delegation execution and to increase the compliance of all delegation changes in the existing policy.

5.4.1 Problem statement (part II)

In the motivation example (see figure 5.1), we observe that the authorisation policy P needs to reflect the new requirements for delegation. In order to derive a delegation policy from the existing policy, we have to specify additional authorisation rules to support delegation, where a rule defines the policy decision effect (e.g. permit, deny). Considering a user-to-user delegation, we motivated that such delegation is done in ad-hoc manner, thereby supporting a negotiation protocol. We consider negotiation as a fundamental step for delegation. It involves all the principals (delegator and delegatee) and negotiation specifications (e.g. time, evidence). Our intention is to envisage a wide-ranging request that gives flexibility for the delegation request. Subsequently, such specifications have to be included in the delegation policy to define specific conditions to validate the policy decision effect.

We consider a different situation for the delegation scenario DS1 where the delegator *Prosecutor* sends a delegation request for all users members of role *Assistant*. This defines a *push* delegation mode, where a delegatee is chosen dynamically based on the negotiation step. An acceptance of delegation inquires a new access control enforcement in the existing policy, thereby adding a new authorisation rule for the delegatee under defined conditions (i.e. time) and/or obligations (i.e. evidence) agreed between the delegation principals. The *Prosecutor* may need to review all the translations done by his *Assistant* for validation. Validation is done based on evidence defined during negotiation. Evidence can be related to the language of translated documents or the number of translated documents within 5 day. To that end, an authorisation rule permitting the access (e.g *read, write*) to the legal document, is constrained by an obligation allowing to investigate

whether evidence were satisfactorily met. If however, evidence are not satisfied, a revoke action may be triggered including a deny result for the previous policy effect.

In traditional access control frameworks no mechanism exists that would support such delegation constraints. Delegation constraints are meant to automate delegation policies from existing policy specifications. Accordingly, it is not possible to foresee a deny rule for revocation during the policy definition. Moreover, a manual review of the current access control rights and task executions is costly, labor intensive, and prone to errors. With an automated mechanism, when the policy changes to reflect delegation, the delegation policy will be derived automatically based on specific facts related to the delegation process. A delegation process defines a task delegation life cycle within the existing process. It is enriched with additional constraints to be compliant with the existing policies. Delegation constraints will have to support specific interactions that would be automatically captured, and specified in the delegation policies for appropriate actions. We do believe that such interactions are intermediate states in the delegation process driven by specified events to control the delegation behaviour.

5.4.2 Monitoring and securing task delegation

Securing delegation involves the definition of authorisation policies which have to compliant with the policy of the workflow. To do so, we need to address two important issues, namely allowing task delegation to complete, and having a secure delegation within a workflow. Allowing task delegation to complete requires a model that forms the basis of what can be analysed during the delegation process. Secure delegation implies the controlled propagation of authority during task execution. To that end, we use the TDM model that forms the basis of what can be analysed during the delegation process in terms of monitoring and security.

The monitoring of task delegation is an essential step to ensure delegation completion. A delegated task goes through different states to be terminated. States depends on generated events during the delegation life cycle. Events such as *validate* may be required when a delegation request is issued under a certain obligation where the delegatee has to perform specific evidence to validate the task execution. For instance, the delegation of T3 can generate evidence related to the number of translated documents within a period of 5 day. Subsequently, evidence validation will be an important step in the delegation process.

Additionally, we consider task delegation as an advanced security mechanism supporting the policy decision. We define an approach to support dynamic delegation of authority within an access control framework. The novelty consists of reasoning on authorisation based on task delegation events, and specifying them in terms of delegation policies. When one of these events changes, our access policy decision may change implying dynamic delegation of authority. Delegation policies are defined from existing policy and are specified from triggered events. For instance, T3 evidence are not satisfied and the validation will trigger the event *revoke* for the delegatee. T3 is not anymore authorised to be executed by the delegatee. In this case, another rule has to be integrated in the policy with an effect of deny for the authorisation.

In order to control the delegation behaviour and to specify its authorisation policies in

an automated manner, we gather specific events that will define both the task execution path and the generated policies for the delegation of authority. In the rest of this chapter, we will present a technique to monitor the delegation execution using *Event Calculus* and we will explain how generated policies change dynamically in response to delegation events.

5.4.3 Modelling task delegation in event calculus

We use our task delegation model to monitor the delegation execution. It defines how delegation request is issued and then executed depending on delegation constraints. The idea is to offer a technique to monitor the delegation execution based on specific events. Using *Event Calculus*, we can foresee the delegation behaviour within a workflow.

a - Background and motivations

The proposed approach for the representation of task delegation process relies on the *Event Calculus* (EC) formalism [KS89]. The choice of EC is motivated by several reasons for delegation. Actually, given the representation of the task delegation model and the corresponding events that enforce policy changes, an event calculus reasoner can be used to reason about such events.

Event Calculus is a logic programming formalism for representing events and is being widely used for modeling different aspects such as flexible process design, monitoring and verification [ZPG10]. It comprises the following elements : $\mathcal{A}$ is the set of *events* (or actions), $\mathcal{F}$ is the set of *fluents* (fluents are *reified*[3]), $\mathcal{T}$ is the set of *time points*, and $\mathcal{X}$ is a set of objects related to the particular context. In EC, events are the core concept that triggers changes to the world. A fluent is anything whose value is subject to change over time. EC uses predicates to specify actions and their effects. Basic event calculus predicates used for modelling the proposed framework are defined in the table 5.2 :

Predicate	**Interpretation**
$Initiates(e, f, t)$	fluent f holds after timepoint t if event e happens at t
$Terminates(e, f, t)$	fluent f does not hold after timepoint t if event e happens at t
$Happens(e, t)$	is true iff event e happens at timepoint t
$HoldsAt(f, t)$	is true iff fluent f holds at timepoint t
$Initially(f)$	fluent f holds from time 0
$Clipped(t_1, f, t_2)$	fluent f was terminated during time interval $[t1, t2]$
$Declipped(t1, f, t2)$	fluent f was initiated during time interval $[t1, t2]$

Table 5.2: Event calculus predicates

The reasoning modes provided by event calculus can be broadly categorised into ab-

[3]Fluents are first-class objects which can be quantified over and can appear as the arguments to predicates.

ductive, deductive and inductive tasks. In reference to our proposal, given a TDM and authorisation policies, it will be interested to find a plan for task delegation, that allows to identify what possible actions (policy changes) will result from the task delegation and may opt to choose the optimal plan in terms of minimal policy changes, this leads to the "abduction reasoning". Then, it will be also interested to find out the possible effects (including policy changes) for a given set of actions (a set of events that will allow task delegation), this leads to the choice of "deduction reasoning" and using the event calculus is thus twofold.

The event calculus models are presented using the discrete event calculus language [Mue06] and we will only present the simplified models that represent the core aspects. All the variables such as task and time are universally quantified and in case of existential quantification, it is represented with variable name within curly brackets, {variablename}.

b - Event calculus based on TDM

The basic entities in the proposed model are tasks. In terms of discrete event calculus terminology, they can be considered as *sorts*, from which instances can be created. Then, each task can be in different states during the delegation execution. In reference to the task delegation model presented earlier (see figure 4.6), the possible task states include Initial, Assigned, Delegated, Completed and others. As task states change over time, they can thus be regarded as *fluents* in event calculus terminology. Further, the state change is governed by a set of actions/events and in relation to task delegation model, the task state changes from Initial to Assigned as a result of *assign* event occurring. Finally the task delegation model introduces a set of orderings, such as the state of a task cannot be assigned, if it is not created earlier. In reference to the event calculus model, we will introduce a set of axioms to handle these dependencies. The event calculus model below introduces the fluents, basic events and dependency axioms :

sort task
fluent Initial(task), Assigned(task), Delegated(task), Started(task)...

event Create(task)
[task, time] Initiates(Create(task), Initial(task) ,time).
event Assign(task)
[task, time] Initiates(Assign(task), Assigned(task) ,time).
[task, time1] Happens(Assign(task), time1) $\rightarrow$ {*time2*} *HoldsAt(Initial(task), time2)*
& $time1 > time2$

Figure 5.7: Event calculus based task delegation model

The event calculus model presented in figure 5.7, first defines *sort* and *fluents* that mark the different task states. Then we define an event *Create*(task), and an Initiates axiom that specifies that the fluent *Initial*(task) continues to hold after the event happens at some time. Similarly, we define the event/axiom for the assignment event and fluent. We further introduce an axiom that specifies that in order to assign some task at time1,

that a task must already be created and thus in Initial state at time2, and time1 is greater than time2. In a similar fashion, we can define events and associated Initial axioms for the complete TDM model.

For the basic event calculus model above, the solutions (plans) returned by the reasoner may also include the trivial plans which does not enforce the delegation and directly *start* or *abort* the task once assigned. In order to give the user ability to choose the delegation mode once the task is assigned, we enrich the model to include the following axioms (see figure 5.8) :

```
[task, time] !Happens(Abort(task), time).
[task, time] !Happens(Start(task), time).
[task, time] !Happens(PullDelegate(task), time).
```

Figure 5.8: Delegation mode choice

The event calculus model above, specifies that the task does not either *Start, Abort* or requires *PullDelegation* once assigned, and thus the only option for the reasoner is to conclude that the model requires a *PushDelegation* mode . We can similarly restrict the delegation permission (Grant/Transfer), once the task is in the *WaitingforCompletion* state (see TDM states in figure 4.6).

5.4.4 Building policies for delegation

Here we analyse the security requirements that need to be taken into account to define event-based delegation policies. Additional requirements such as *pull/push* mode and *grant/transfer* type may be a source to a policy change during delegation. Using *Event Calculus*, we present a technique capable of computing and generating new policy rules automatically.

We define delegation transitions as events ruling delegation behaviour. The internal behaviour is based on events defined in our TDM, and may be a source to a policy change, thereby requiring the integration of additional authorisation rules.

A policy rule may include conditions and obligations which are used to identify various conditions or cases under which a policy may become applicable. Conditions and obligations are related to delegation security constraints when defining delegation policies. Based on the result of these rules different policies can become applicable in the context of delegation (see definition 12).

Definition 12 (Delegation Rule) *We define a delegation rule* $\text{rule}_D \subseteq \text{P}_D$*, where* rule_D = (effect,condition,obligation)*, where* effect *returns the policy decision result (permit, deny),* condition *defines the delegation mode (push, pull) and* obligation *checks evidence.*

We analyse security requirements that need to be taken into account in a *push* mode to define delegation policy rules based on TDM (see figure 4.6). We present a table

that gathers specific events for *push* delegation and analyse them in terms of policy rules. Adding rules in the workflow policy will ensure the delegation of authority, thereby adding the required effect (permit, deny) to the delegation policy rules (see table 5.3).

Delegation Events	Push Delegation		Adding Policy Rule
	Grant	Transfer	
u1:delegate	✓	✓	Add rules if accepted
u2:accept	✓	✓	Add rules based on execution type
u1:cancel	✓	✓	No add
u2:execute/Grant	✓		(Permit,Push,Grant:Evidence)
u2:execute/Transfer		✓	(Permit,Push,Transfer:NoEvidence)
u1:validate	✓		No add
u1:revoke	✓		(Deny,Push,Grant)
u2:fail		✓	No add
u2:complete		✓	No add

Table 5.3: Push delegation policy rules-based events

Returning to the example, we can observe a dynamic policy enforcement during delegation. Initially, T3 is delegated to the *Assistant* u_2 (the delegatee) and the delegation policy for T3 : P_D = *(RD,Permit,{Push,5 days})* (see table 5.3/*u2:execute/Grant*). In the meanwhile, the *Prosecutor* u_1 (the delegator) is back to work before delegation is done and is not satisfied with the work progress and would revoke what was performed by his assistant so far. The *Prosecutor* is once again able to claim the task and will revoke the policy effect (permit) for the *Assistant*. The event *revoke* will be updated in the policy, and a deny rule is then updated in the policy. Thus, the delegation policy for T3 needs to generate a new rule and the delegation policy is updated to :
P_D = *(RD,Deny,{Push,Grant})* (see table 5.3/*u1:revoke*).

Note that the generated rule depends on the *RD* relation to check access rights conflicts. We determined access rights based on the current task status and its resources requirements using a task-based access control model (TAC) for delegation presented in the previous chapter. Our access control model ensures task delegation assignments and resources access to the delegatees corresponding to the authorisation policy constraints.

5.4.5 Modelling delegation policies in event calculus

In order to model the delegation policy, we introduce new sorts called *effect*, *condition*, and *obligation* to the event calculus model for the table 5.3 and specify instances of each sort to be the possible effects, conditions and obligations. Possible effects include *Deny* and *Permit* results, and conditions define the *Push* and *Pull* modes. The possible instances for obligations include *Grant, Transfer, Evidence* and *NoEvidence* which are constraints related to delegation type and mode. We further add an action *AddRule(effect, condition, obligation)* and enrich the model to specify the policy changes as a result of events. We define an algorithm to describe how to compute delegation policy rules. For a sake of simplicity, we will just present the grant type (see algorithm 4).

Algorithm 4: Computing grant delegation policies for a task

```
Require : DPolicy // Delegation policy
sort // task, effect, condition, obligation
effect // Permit, Deny
fluent // AddRule(effect, condition, obligation)
event //AddPolicyRule(effect, condition, obligation)
time //time1 ≥ time2
while (HoldsAt(AddRule(effect, condition, obligation)) AND obligation = grant) do
    if (Happens(PushDelegateAcceptExecuteGrant(task), time1) = true then
        AddPolicyRule(Permit, Push, Evidence) /* Grant authorisation with a permit
        rule */
    else
        if (Happens(PushDelegateAcceptRevokeGrant(task), time1) = true then
        AddPolicyRule(Deny, Push, Grant) /* Cancel authorisation with a deny rule */
Result : DPolicy = (AddPolicyRule(effect, condition, obligation), time1)
```

The policy change specifies that once certain actions happen (execute, revoke), they cause policy change and thus we add a new rule to the existing policy. The name of actions/events depicts their invocation hierarchy, *PushDelegateAcceptExecuteGrant* is the *execute* event with a grant permission once the *PushDelegation* request has been accepted by a delegatee and has to be validated before completion (see table 5.3).

5.4.6 Delegation automation

Automation is necessary for both the task completion and the policy specification during delegation. Reasoning on delegation events using *Event Calculus* offers a solution to foresee the delegation execution and increase the control and compliance of all delegation changes.

a - Benefits

By reasoning on specific events, we are able to control the order of delegation execution which is computed automatically based on events. Events can distinguish between the order of execution by checking the delegation mode and type. For instance, an execution expects a validation transition if and only if we are in a *grant* delegation type. In addition, we are able to address the policy stateless issue. We can compute delegation policies from triggered events during task execution [GZCG].

Delegation automation offers many benefits. Actually, it reduces efforts for users and administrators. Administrator efforts can be related to the process definition and policies specification. Moreover, it increases control and compliance of all delegation changes. Subsequently, task delegation is accomplished under constraints which are compliant with the global policy of a workflow. For instance, time constraint has to be taken into account when granting a temporal access for delegation (i.e. T3 deadline in figure 5.1).

b - Reasoning

In our study, we utilise the Discrete Event Calculus Reasoner (DECReasoner) for performing automated commonsense reasoning using the event calculus. It solves problems efficiently by converting them into satisfiability (SAT) problems. Actually, the DECReasoner attempts to find a solution by transforming the EC model into a SAT problem and invoking the SAT solver for the solution [GZCG]..

The event calculus based on the task delegation model can be used to reason about the delegation process. As we discussed earlier, the reasoning can either be abductive or deductive. For the abductive reasoning, a plan is sought for the specified goal. A plan is a sequence of *EC Happens clauses* that specify the temporal ordering of different actions event-based, whose execution leads to the goal. In reference to our proposal, the goal is to have either the task in completed, cancelled or failed states, so we add the goal *[task] HoldsAt(Completed(task),15) | HoldsAt(Failed(task),15) | HoldsAt(Cancelled(task),15)* to the event calculus model and add an instance of the delegated task T3, where 15 is the width of the *timepoint* interval. For instance, a value less than 15 may stop our execution before the defined goal (e.g. completed, cancelled or failed).

```
1389 variables and 7290 clauses
relsat solver
1 model
—
model 1:
0 Happens(Create(T3), 0).
1 +Initial(T3).
2 Happens(Assign(T3), 2).
3 +Assigned(T3).
4 Happens(PushDelegate(T3), 4).
5 +WaitingDelegation(T3).
6 Happens(PushDelegateAccept(T3), 6).
7 +WaitingCompletion(T3).
8 Happens(PushDelegateAcceptExecuteGrant(T3), 8).
Happens(AddPolicyRule(Permit, Push, Evidence), 8).
9 +RuleAdded(Permit, Push, Evidence).
+WaitingValidation(T3).
10 Happens(PushDelegateAcceptExecuteGrantValidate(T3), 10).
11 +Completed(T3).
—
;DECReasoner execution details
0 predicates, 0 functions, 12 fluents, 20 events, 90 axioms
encoding 0.5s - solution 0.2s - total 0.9s
```

Figure 5.9: Delegation plan

The invocation of the event calculus reasoner will then give us a set of possible solutions (called plans) for achieving the goal. Let us first consider the case, when the chosen

delegation mode is *PushDelegation* with the grant of permissions to the delegatee, the event calculus reasoner returns the plan detailed in figure 5.9.

The execution plan follows the delegation of T3 described in the delegation scenario DS1. It shows the actions that need to be considered for delegation and most importantly, it shows the possible policy changes as a result of delegation. Steps from 1 to 11 depicts the delegation process execution. Having a *push* mode as a condition, we derive the relevant rules to be added in the policy. For instance, step 8 and 9 show when a delegatee request the task T3 for execution. Delegatee acceptance went through the "WaitingDelegation", "WaitingAcceptance" and "WaitingCompletion" states (see figure 4.6). Based on those events, we deduce that an authorisation rule is added at this stage under a certain obligation (evidence for task validation), and finally a task validation completes the delegation execution (see steps 10 and 11 in figure 5.9).

All the defined axioms using the DECReasoner language can be given to the reasoner for finding a solution (if exists) to support policy changes, which automatically orients these axioms into delegation rules. Then, given as inputs the specification of the conditions and obligations expressed when adding rules, the generated plan by the reasoner shows that either the authorisation rules result in a permit or a deny decision (Verification results and encoding details can be found in the Appendix B).

In concrete policy changes, there are two possible scenarios. The first scenario is the adding of a new policy rule because the conjectures (conditions or obligations) are valid. The second scenario concerns cases corresponding to an overriding of this rule to a deny result.

c - Discussion

If there are some conflicts in the event calculus model this leads to empty solution set and requires the **verification of the model** to identify any conflicts with delegation constraints. Our approach for verification relies on the SAT solver to provide a set of *near-miss models* and/or *unsatisfied clauses* [ZPG10]. As an example, we consider the delegation type constraint *transfer* added to the event calculus model, saying that the delegated task t_i should not be validated once executed by the delegatee. In case of planning, the reasoner will only generate the solutions that will respect this constraint :

```
Happens(PushDelegateAcceptExecuteTransfer(t_i), 18)
Completed(t_i)
```

However, if no such solution exists and the only solution is to validate t_i before completion :

```
Happens(PushDelegateAcceptExecuteGrantRevoke(t_i), 19)
Happens(AddPolicyRule(Deny, Push, Grant),19)
```

This can be misleading when adding a new rule if the validation is not satisfied (revoking a rule). Thus, the planner will return a near-miss model highlighting the type of constraint (transfer) where no validation is required for the transfer of delegation.

d - Evaluation

We aim to check the performance of our reasoning algorithm to **evaluate the performance** of the proposed framework. The example presented in figure 5.9 computed the time of encoding for a single delegated task. Here, we have tested the event-calculus model for sequential and parallel tasks (about 50 tasks) using the DEC reasoner. The tests were conducted on a MacBook Pro Core 2 Duo 2.53 Ghz and 4GB RAM running Mac OS-X 10.6. The DEC reasoner version 1.0 and the SAT reasoner, relsat-2.0 were used for reasoning.

Figure 5.10 illustrates two curves for sequential and parallel tasks. Time is computed based on the total of time encoding (DEC reasoner) and solution computation (SAT). We can observe that the sequential task curve reflects a good performance of our model. For instance, 40 delegated tasks in sequence will take less than 25 seconds to be encoded and solved. However, parallel (concurrent) tasks are more time consuming. For the same number of tasks, we will need about 2 minutes. This can be explained by the algorithm of encoding for parallel tasks. The algorithm complexity is $\Theta(n^2)$, which presents an exponential growth in the graph (see appendix B).

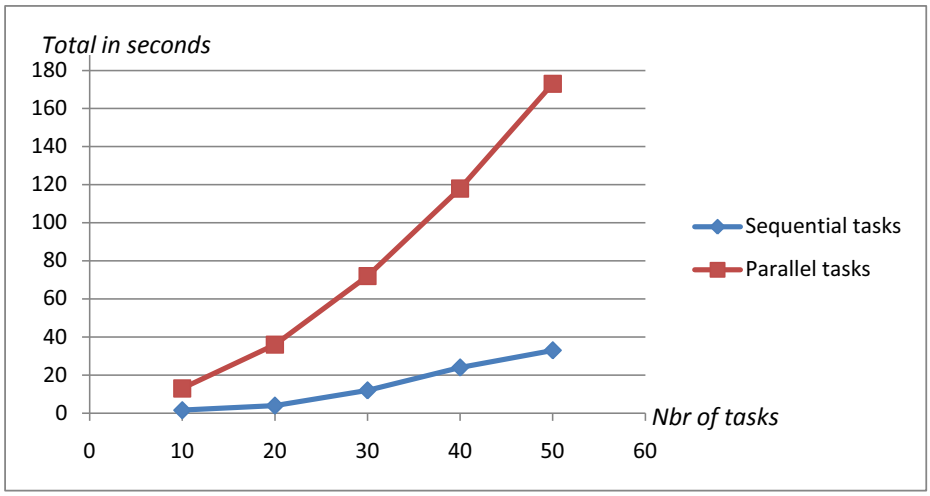

Figure 5.10: Performance testing for sequential and parallel tasks

In general the encoding process does not scale well especially with the increase in *timepoints*. and as a future work, we aim to modify the proposed DECreasoner encoding to make the process faster. For the delegation process re-instantiation, a key observation is that it always takes less time than the initial solution as we do have a partial plan and that reduces the problem. Subsequently, we can leverage the trace of encoding in DECReasoner to give all necessary information (events, fluents and timepoints) to help designer (policy administrator) to detect policies problems. The event calculus model can further be enriched with additional axioms to ensure minimal policy changes. To do so, we can use an **auditing technique** that allow us to choose the best candidate based on his performance using a potential list of delegatees that can accept delegation. Performance

may be related to the successful completion of a task and so the reduction of rules update in the policy.

5.4.7 Summary

In this section, we have presented problems and requirements to integrate dynamic delegation of authority in workflow systems. We proposed a technique for delegation to specify delegation policies automatically. Delegation policies may change according to specific events. We defined the nature of events based on task delegation constraints, and described their interactions with policies decisions. When relevant events occur, we presented how delegation will behave and how policy rules will change dynamically in response to this change. Using *Event Calculus* formalism, we implemented our technique and deployed an example for task delegation ensuring authorisation policy changes in a compliant manner.

5.5 Conclusion

In this chapter, we have presented the second part of our approach to secure task delegation in access control systems. Actually, we have observed that the delegation completion and its authorisation enforcement are specified under specific constraints defined in the task delegation model. On one hand, control constraints define a fixed set of delegation events **to monitor the delegation execution**. On the other hand, delegation policies may change according to security constraints. We defined the nature of events based on such constraints, and described their interactions with policy authorisation decisions. **When relevant events occur, we defined how delegation will behave and how policy rules changed dynamically in response to this change**.

Moreover, we addressed the issue related to the integration of delegation policies. To do so, we have investigated **the potential of delegation events to ensure authorisation policies which are compliant with the authorisation policy of the workflow**. We have proposed a technique that automates delegation policies using event calculus to control the delegation execution and increase the compliance of all delegation changes in the existing policy.

Besides that, we detailed delegation protocols to support human interactions. We argued that the access control enforcement has to take into account new changes due to delegation events in order to update the existing policy. Dealing with that, we have architected a solution based on existing access control systems and extended it with a dynamic policy enforcement component for delegation policies. The implementation of access control framework supporting dynamic task delegation will be discussed in the next chapter.

Chapter 6

Deployment Environment

Contents

6.1 Introduction

This chapter presents the development environment for the delegation framework. We have implemented our framework within the R4eGov (Research for e-Government) project[4]. We present the collaborative infrastructure for e-government applications and we focus on interactive processes supporting human interactions. To do so, we have developed a delegation plugin that we integrated within the workflow system in R4eGov. By this means, we aim to **model and monitor task delegation within workflows**. In addition, it

[4]For more details : http://www.r4egov.eu/

is necessary to provide access to delegated resources intra and inter organisations. To do so, we have extended our delegation plugin with security requirements for authorisation policies. Using access control mechanisms in R4eGov, we have developed a secure delegation application to **enforce dynamic delegation policies** within the global policy of the workflow system.

6.2 The Development Environment

In this section, we present our conceptual architecture based on a decentralised collaboration approach tailored to provide transparency and control features to support human-centric interactions (i.e. user-to-user delegation) within e-Governmental workflows.

6.2.1 Project overview

The R4eGov project consists of inter-organisational collaboration between European administrations. It describes an interagency collaboration within the areas of law enforcement and justice. One of the objectives is to establish a collaboration, including information exchange cross organisations based on a global policy, to which they have to comply. Those objectives can be achieved using collaborative workflow systems.

Moreover, the project follows a decentralised approach, combining the local workflows to form a collaborative workflow, integrating the existing systems of the involved partners, and adding a decentralised collaborative administration architecture. Partners are members of the different organisations involved in the e-Government project. The whole infrastructure operates on global collaboration policies and makes local workflow data and events visible to all partners according to these policies.

6.2.2 Collaboration infrastructure

We identify key components to leverage an existing workflow system onto decentralised collaboration. As indicated by figure 6.1 the proposed architecture is divided into a control-flow layer and an administration layer which are consisting of the following components :

Abstraction Layer : The abstraction layer wraps a collaborative partner's underlying workflow management system (WfMS). It provides a unified access to the local WfMS by publishing services of the local WfMS that are necessary for the collaborative workflow coordination.

Interoperability Gateway : The interoperability gateway (IOP) intercepts internal and external control-flow messages, such as local application calls and remote web service invocation. The IOP intercepts the messages, reports them to the collaborative messaging framework and forwards the messages to their destination.

Collaborative Messaging Framework : The collaborative messaging framework (CMF) provides a layer for exchanging coordination messages between distributed partners

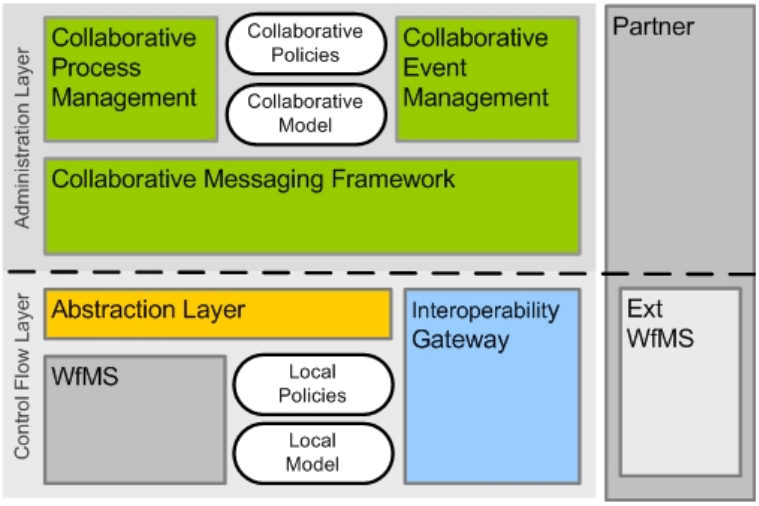

Figure 6.1: Modular architecture in R4eGov

that are not directly part of the workflow-based message exchange (e.g. local compensation events have to be send to the other collaboration partners). It mediates local events, such as message sending, message reception, local state transitions, and audit data to the collaborative partners according to collaborative policies.

Collaborative Process Manager : The Collaborative process manager (CPM) has access to the overall collaborative process model and policies. Each partner has information about the overall collaborative process. The global model is used to align received events (e.g. a cancellation event) and collaborative information with the overall workflow and participating peers.

Collaborative Event Management : The event management is used to queue incoming information from external partners for later processing by the CPM. The event management is used to dispatch events to the process management or other local components (e.g. a policy decision component).

The described core components provide the minimal necessary functionality to enable the communication between distributed workflow systems by using the messaging framework. On top of the messaging framework additional components have to be deployed to add monitoring and security requirements supporting human interactions. One mandatory component is that of task delegation. Based on the process execution, we will identify specific events that will trigger delegation requests. **Delegation events will be conveyed via the messaging framework and processed using a communication layer to support user-to-user delegation**, thereby ensuring a delegation dialogue between the delegator and the delegatee.

6.3 Delegation for Human-centric Workflows

The motivation example (Mutual legal Assistance scenario) provided by the R4eGov project is mainly human-centric and contains only few automated processes. The existence of a task management system to deal with such human tasks is therefore a fundamental part of the extended runtime architecture to support task delegation.

6.3.1 Administrative communication layer

As discussed previously, the process needs to publish events or performs requests to the collaborative components. To do so, we propose to set up a workflow to workflow collaboration by defining a layer which we call an Administrative Communication Layer (ACL) [SIP07].

The ACL architecture is divided into a control-flow layer and an administration layer. The control-flow communication layer mediates workflow related messages of the workflow engines to directly involved actors of the workflow (users). The administrative collaboration layer is used by the collaborative process manager to send administrative messages beyond the control-flow via the messaging framework (see figure 6.2). The local message exchange can be identified between both layers. This allows to monitor the local workflow execution and to enforce global policies according to the principles of transparency and control.

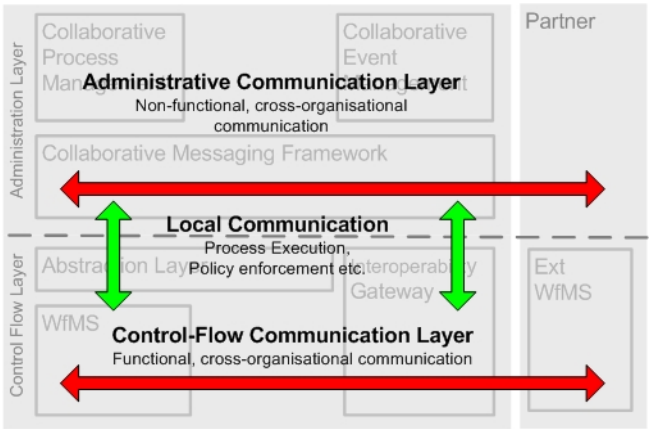

Figure 6.2: Administrative communication layer

The term Administrative Communication Layer (ACL) refers to the distinction of administrative events (e.g. starting/completion of one task) with control events (e.g. starting a workflow instance). The following aspects of administrative communication between the local process engine and the involved users are supported :

- Status management to represent the overall status of the collaborative workflow (displaying the local process of the executing users together with the overall workflow of all involved organisations).

- **Exceptions handling** and execution of alternative scenarios which cannot be handled as part of the regular process model (e.g. A *Prosecutor* delegates a part of his work to his *Assistant*).

The messaging framework enables administrative information exchange by mediating information to the collaborative event management and process management components. We mainly aim to illustrate such collaboration for delegation. We will leverage the messaging framework infrastructure to set up a first delegation protocol once a task is delegated within a workflow (see next section).

6.3.2 Collaborative workflow runtime

We use an open-source workflow engine which is chosen as a basis for the prototype development to **model and monitor task delegation within workflows**. We describe the Bonita[5] WfMS used to execute workflows/business processes. The Bonita WfMS is a workflow system having innovative features such as activities being able to start in anticipation. Bonita has an awareness infrastructure allowing user notification of any events occurring during the execution in a given process. Traditional workflow features such as dynamic user/roles resolution, and activity performer are also included in Bonita.

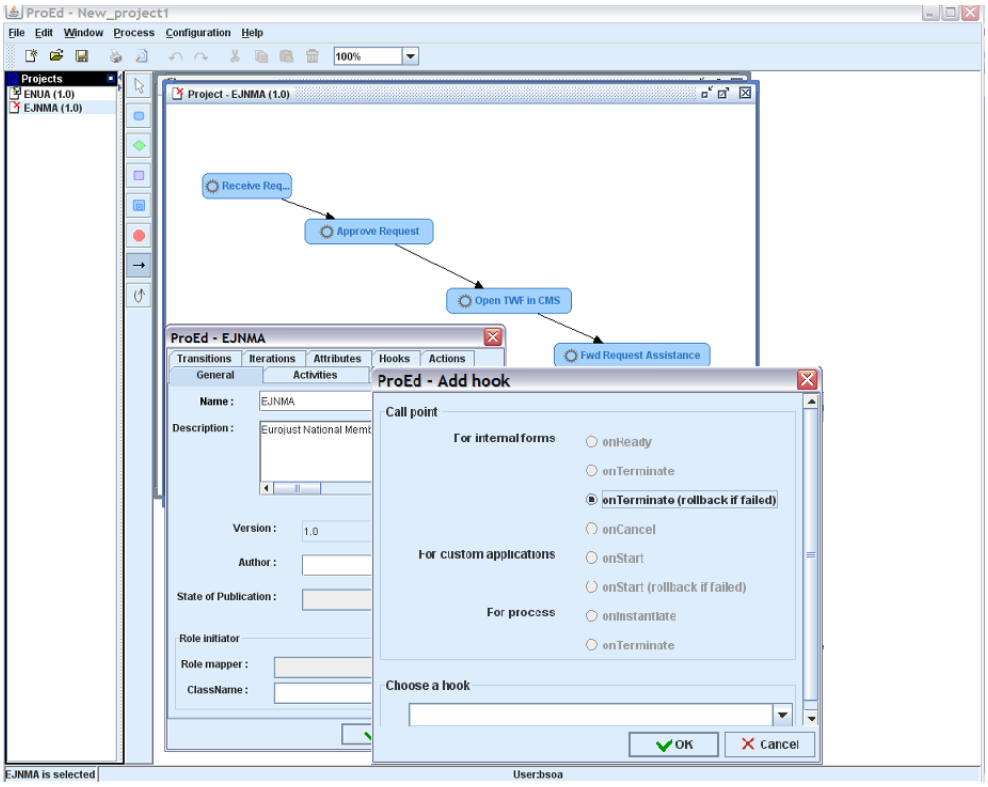

Figure 6.3: Adding Hook to MLA process using Bonita ProEd

[5]More details about Bonita can be found at : http://wiki.bonita.objectweb.org/

Bonita is a fully conformant J2EE application, taking advantage of the power and robustness of the J2EE platform. The Bonita API is accessible through a set of session beans. Processes are created using a graphical definition tool (ProEd see figure6.3). XPDL (XML Process Definition Language) specifies the modelling processes language which is suitable to support human interactions.

The terminologies regarding the concepts of Bonita that have been used extensively are as follows :

- An activity is the smallest unit of task/work that is realised by Nodes.
- A process is a set of activities. The term project is also used to mean process in Bonita.
- A hook is a piece of code that adds automatic behavior to activities/nodes and workflow processes/projects.

Our concern is related to the messaging service in Bonita. We aim to collect specific messages related to delegation. Activity Hooks are user-defined logic that can be triggered at some defined points in the activity life cycle. For instance, they can be used to start a delegation request once an activity is started (see figure 6.3). Thus, each method call will involve a state modification of the workflow system which is registered into a JMS (Java Message Service) topic. The notification can be done via Instant Messaging services, or a Traditional Mailer. The notification of a delegation requests will initiate the user-to-user delegation protocol (see next sections).

6.3.3 ACL plugin supporting delegation

The ACL plugin aims to allow the propagation of events from and to a Bonita workflow engine within the main R4eGov framework. These produced events are conceptually published to a JMS topic, called by default "TestTopic". These two available features were used when designing the plugin. The *BonitaLocalListener* and *BonitaLocalControl* components are acting as an abstracting layer for the underlying workflow engine. The *BonitaLocalListener* component is a message producer, which translates messages published by the engine on the *TestTopic* topic into messages.

The *BonitaLocalControl* is a message consumer, which can command the underlying workflow engine to a certain extent. Bonita offers a so-called Hook mechanism, allowing a programmer to enrich workflow instances with calls to specified java classes. For this plugin, we relied on this mechanism for intercepting tasks items interacting with workflow's actors. Basically, depending on the process events (activity start) hooks will handle the local communication via messages display or mails. We leverage this method to convey delegation events once a user needs to delegate his assigned task. To do so, a user acting as a delegator may specify via a hook message a delegation request that will be sent as an e-mail (see figure 6.4). The detailed method and its implementation is discussed in the following section.

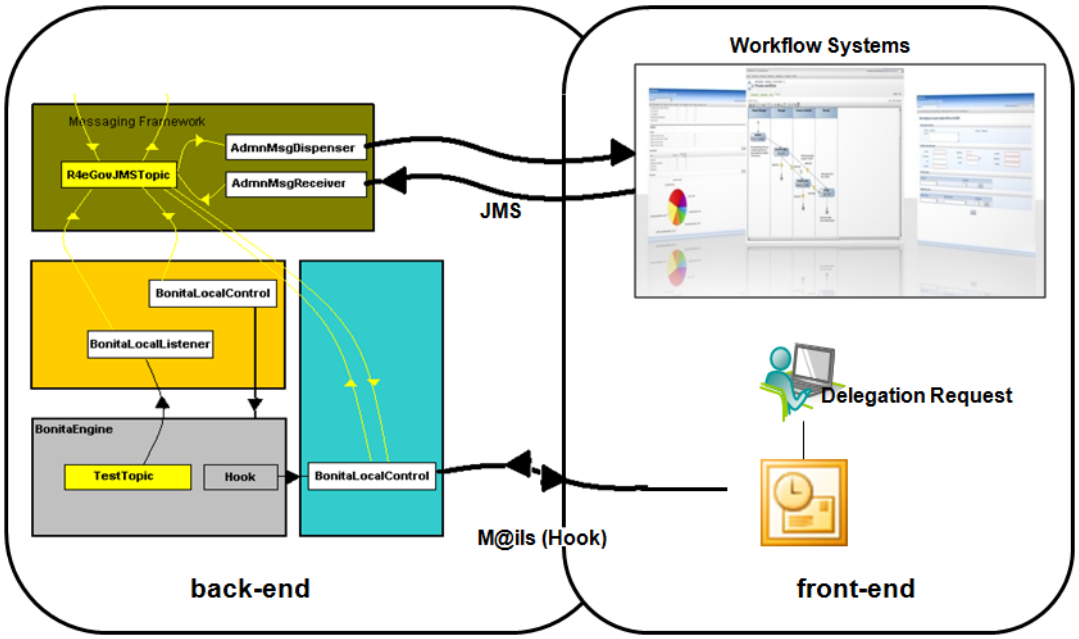

Figure 6.4: Integrating the Bonita engine within the core R4eGov framework

6.3.4 Email-centric task delegation tool

We developed a solution to support user-to-user delegation via e-mail. We use the collaborative task management solution (CTM)[6]. CTM is designed to support user's task management. It assists the user in delegating tasks to colleagues and in controlling the status of work [Gaa08].

CTM is closely related to the messaging framework component and the administration layer already developed in the R4eGov framework. The CTM Office-Integrated Task Management Client is a Microsoft Outlook Add-In with proprietary extensions of the Outlook mail and task items.

CTM features offers a suitable solution to support our delegation protocol. It support the delegation's negotiation and the acceptance (see figure 6.5). The delegator defines his delegation requests and can add negotiation factors related to the request deadline, the workload, etc. Once the request is issued, the delegatee, as the email receiver, may accept or decline this request. In addition, he can negotiate the deadline or asks for less work. This will give us more flexibility when defining a delegation request.

The next step will be to discuss security issue when delegating a task. Once the delegation negotiation is validated and the request is accepted, we need to delegate privileges to grant resources access to the delegatee. This part will be discussed in the following section.

[6]For more details : http://www.eudismes.de/

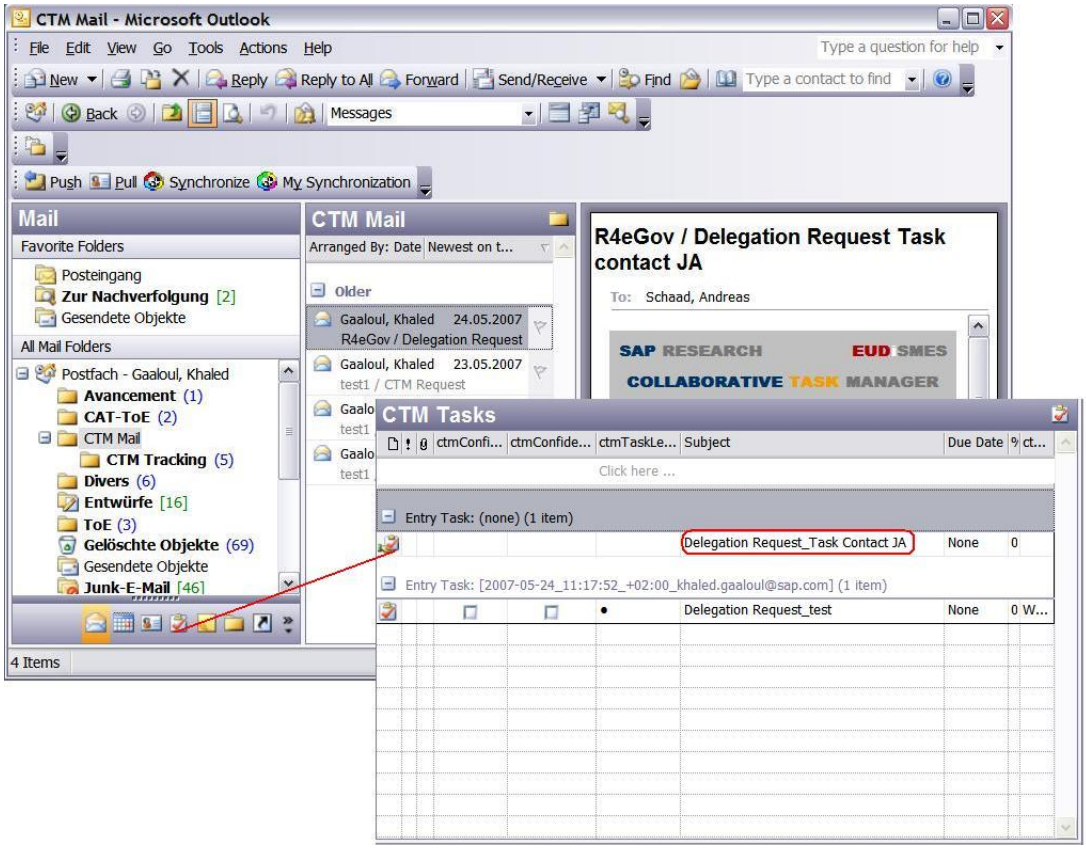

Figure 6.5: CTM overview

6.4 Delegation Policies Enforcement

In the R4eGov framework, policy enforcement extension modules are a mean to extent the built-in security mechanisms of the encapsulated workflow management system. Delegation policies represent an integral part of the policy enforcement component. In this section, we discuss how **a delegation request is issued in a secure manner and analyse its implementation within an access control system**.

6.4.1 A secure delegation protocol

In order to secure task delegation, we have to deal with delegated privileges. Privileges define rights or permissions to execute a task. In the context of access control systems, we delegate a privilege by ensuring access rights to the delegatee. Basically, once a delegated task is accepted, a delegatee must be authenticated and then authorised to perform the delegated task.

Delegated privileges may be associated with roles to execute a delegated task and to access to task's resources. Delegated privileges are granted by the delegator to the delegatee. In order to control access to the delegated task's resources, both **authentication** and **authorisation** are required. Authentication and authorisation requirements are both

specified in the delegated privileges.

We developed a privilege management infrastructure based on digital certificates. We consider that a public key certificate (PKC) is used for authentication and maintains a strong binding between a user's name and his public key, whilst an attribute certificate (AC) is used for authorisation and maintains a strong binding between a user's name and one or more privilege attributes. Therefore, we consider PKC as a passport and AC as a visa. We assume that both PKC and AC will be issued by the delegator to the delegatee. The scenario below (see figure 6.6) illustrates how a delegatee gets access to task resources. This scenario assumes that the delegator issued already a certificate for the delegatee to act on his behalf.

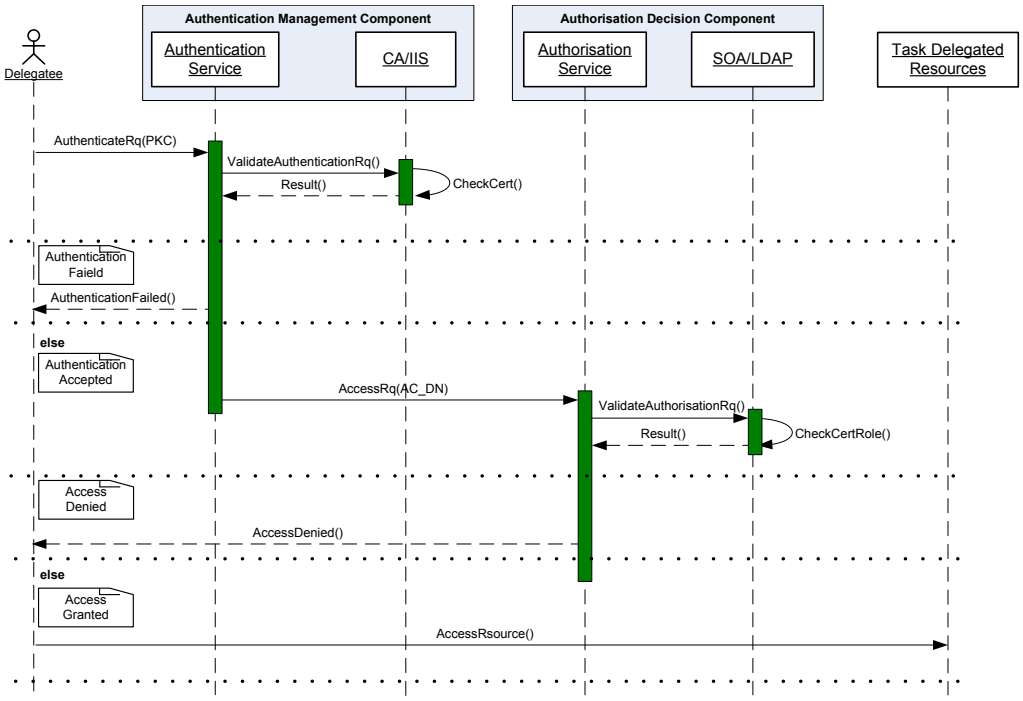

Figure 6.6: Delegated privileges during task's resources access

In figure 6.6, the Authentication Management Component (AMC) authenticates the user, and then asks the Authorisation Decision Component (ADC) if the user is allowed to perform the requested action on the particular task resource. The ADC accesses one or more LDAP (Lightweight Directory Access Protocol) directories to retrieve the authorisation policy and the role ACs for the user, and bases its decision on these. The main steps are detailed as follows :

1. A delegatee initiates an authentication request to the AMC using his PKC.

2. The AMC check the authentication request based on his CA (e.g. CA defined in the IIS Windows OS). If it is accepted then AMC passes the Distinguish Name (DN) of the delegatee to the ADC through a call to get his attribute certificate (AC) and check his permissions afterwards. For instance, the delegatee would be

authenticated in different ways such as sending an S/MIME email message to the ADC, or opening an SSL connection. In all cases the delegatee will be digitally signing the opening message, and verification of the signature will yield his DN.

3. The ADC uses this DN to retrieve the role ACs of the delegatee from the list of LDAP. The role ACs are validated against the policy (e.g. to check that the DN is within a valid subject domain, and to check that the ACs are within the validity time of the policy etc). Policy based authorisation is specified by the domain administrator such as the source of authority (SOA).

4. Once the delegatee has been successfully authenticated he will attempt to perform certain actions on the resources. At each attempt, the AMC passes the access request to the ADC to decide. Decision checks if the action is allowed for the role that the delegatee has in his AC. If the action is allowed, ADC returns access granted, if it is not allowed it returns denied.

The next step will implement the different components for delegation. First, we developed the authentication component based on digital certificates. We then leveraged certificates properties to manage delegated privileges and compute authorisation credentials for the delegatee.

6.4.2 Authentication management component

We developed a solution based Windows Communication Foundation (WCF)[7] to support authentication. WCF is a migration from the WSE 3.0 Web Services to WCF. The benefits of the migration include improved performance and the support of additional transports, additional security scenarios, and WS-* specifications. WCF security provides confidentiality and integrity for message exchanges. It supports both symmetric and asymmetric cryptography implementation. In this work, we discuss WCF security using X509 (asymmetric) specifically for authentication purposes. WCF enables three different bindings using X509 namely Https, SSL over TCP and Message Security.

We developed a solution supporting message level security. Message security uses WS-* protocol to secure messages. WCF enables this through its various standard bindings with message security mode. Moreover, it supports both SSL and Mutual Certificate protocol.

In our application, we implemented a certificate-based mutual authentication where the server and the client (the delegatee) authenticate one another. First, we use the tool *makecert.exe* to create X509 certificate. This tool is packed along with Microsoft .Net 2008. It permits to create a temporary certificate that will sign and authorise client/server certificates. Both certificates are made to secure the endpoint of the client and web service, respectively. Then, we create a new website called WCF509 service for authentication via X509 certificates. This service will be then executed by a client application. We focus on the aspect of authentication when a client is requesting a service and need to be authenticated using digital certificates. The first step of the application is to ensure the

[7]For more details : http://msdn.microsoft.com/en-us/library/ms735119.aspx

authentication of the requestor by identifying his credentials attributes by the service. The client application calls the service as follows :

```
using WCFX509Client.WCFX509Service;

class Client
{
    public static void Main(string[] args)
    {

        WCFX509Client client = new WCFX509Client();

        client.ClientCredentials.ClientCertificate.SetCertificate(
            StoreLocation.CurrentUser,
            StoreName.My,
            X509FindType.FindBySubjectDistinguishedName,
            "CN=client, O=SAP, L=KA, S=BW, C=DE");

// Authentication Request

                Console.WriteLine(client.Authenticate());

                Console.Read();

                 client.Close();
}
```

Figure 6.7: The client application

The idea is to retrieve the client credentials from his X509 certificate, to compute his attribute using his distinguish name and to authenticate him against the service, thereby calling the java class *Authenticate()*. Once the server/client sides implemented and configured, the last step is to add the service reference that the client will reach in the WCFX509Client class. It defines the URL address of the service web. The deployment of the authentication component if successful will return the requestor attributes and authenticate the requestor.

In the context of delegation, authorisation attributes provided to the requestor (the delegatee) are restrictive in terms of actions on targets (resources). The next step will be the computation of his authorisation attributes against the authorisation policy.

6.4.3 Authorisation decision component

In order to secure our delegation protocol, we developed a privilege management application responsible for the authentication and authorisation of the delegatee. Authorisation will define delegation policies to be integrated into existing access control systems. Using an authorisation decision making mechanism, a delegatee will access task's resources via an application gateway.

a - Access control framework architecture

The application gateway known as application proxy or application-level proxy is an application program that runs on a firewall system between two networks. When a delegatee establishes a connection to a destination service, it connects to an application gateway, or proxy. We present the implementation of our access control framework for delegation in figure 6.8. **We implemented a policy plugin within the authorisation decision engine PERMIS**. The arrows in yellow, X509 Attribute Certificate and PERMIS

Authorisation decision, are implemented in the application gateway as AMC and ADC components, respectively. The Contact Point/SOA is not supported in this application.

The AMC authenticates the user, and then asks the authorisation component (ADC) if the delegatee is allowed to perform the requested action on the particular target resource. The ADC accesses the specified LDAP directory to retrieve the authorisation policy and the role ACs for the delegatee, and bases its decision on these using the PERMIS decision engine.

In summary, when a delegatee submits an access request to a target, the AMC, acting as PEP (Policy Enforcement Point), gives the credentials to the ADC acting as PDP (Policy Decision Point). The delegatee's identifier and his role/attribute are given in our AMC application. The first step is to recognise the delegatee's identifier and his different roles/attributes. The second step is to convert the user's identifier into a distinguished name. The third step is to convert the different roles/attributes into the PERMIS authorisation token in order to be understandable by the PERMIS decision engine. Because the PERMIS decision engine is an RBAC-Based Authorisation infrastructure and defines roles/attributes to make access control decisions [ASL08].

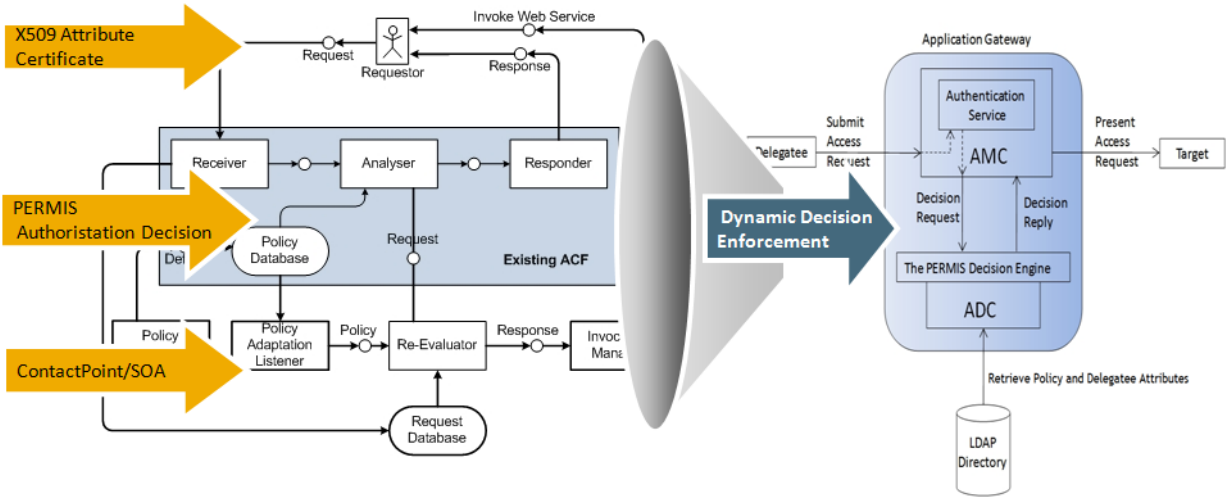

Figure 6.8: Implementing the access control framework for delegation

b - PERMIS decision engine

PERMIS[8] is a policy based authorisation system, a Privilege Management Infrastructure. It can work with any and every authentication system (Shibboleth, Kerberos, PKI, username/PW, etc.). Given a username, a target and an action, the PERMIS decision engine says whether the user is granted or denied access based on the policy for the target. The policy is role/attribute based i.e. users are given roles/attributes and roles/attributes are given permissions to access targets.

[8]For more details : http://sec.cs.kent.ac.uk/permis/index.shtml

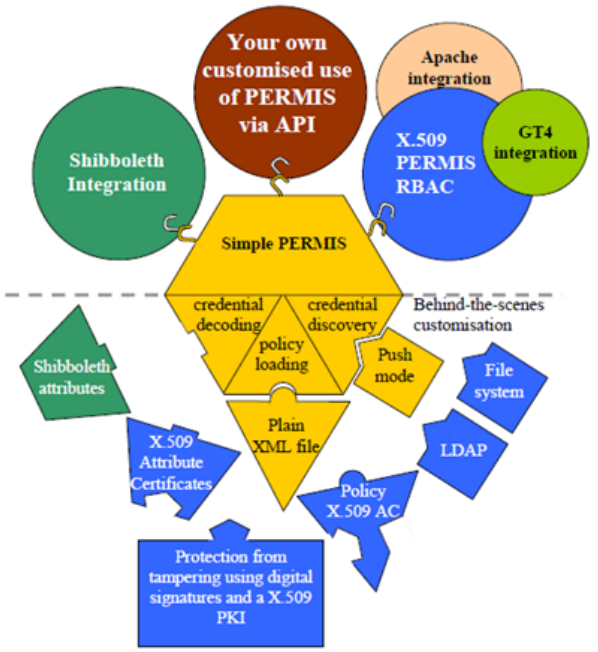

Figure 6.9: PERMIS decision engine architecture

The interface to the PERMIS decision engine (PDP) has being enhanced to support the XACML request context and to return the XACML response context. This presents an interest for our work to support features that are not currently supported by XACML, such as dynamic delegation of authority [CON09]. Moreover, the PERMIS decision engine can work in push mode (attributes are sent to PERMIS by the application) or pull mode (PERMIS fetches them itself given the distinguished name of the user by the application). In our application, we used push mode for the sake of simplicity. It can be considered as a lightweight authorisation decision engine.

c - Decoding credentials

The PERMIS API for the decision engine comprises 3 methods : *GetCreds*, *Decision*, *Shutdown*, and a Constructor. The Constructor builds the PERMIS API java object. For construction, the AMC passes the delegatee's credentials to the ADC which retrieves the policy AC and subsequently the role ACs (see figure 6.10).

When a delegatee initiates a call to the target, the AMC authenticates the delegatee, and then passes the LDAP DN of the delegatee to the ADC through a call to *GetCreds*. The delegatee will be digitally signing the opening message, and verification of the signature will yield the delegatee's DN. The ADC uses this DN to retrieve the role of each attribute certificate of the delegatee from the list of LDAP URIs passed at initialisation time. The role ACs are validated against the policy e.g. to check that the DN is within a valid subject domain, and to check that the ACs are within the validity time of the policy etc. Invalid role ACs are discarded, whilst the roles from the valid ACs are extracted and kept for the delegatee. In our model, *GetCreds* supports the "push" model, whereby

the AMC can pass a set of ACs to the ADC, instead of retrieving them from the LDAP directories.

```
    protected static String consult(issrg.simplePERMIS.SimplePERMISPrincipal
principal,
    issrg.pba.Action action, issrg.pba.Target target)
    {
        //Parsing User Credentials
        issrg.pba.Subject subject;
        issrg.simplePERMIS.SimplePERMISTokenParser testParserTok = new
        issrg.simplePERMIS.SimplePERMISTokenParser(principal.getName());

testParserTok.setAuthTokenParsingRules(ssampf.getParsedPolicy().getAuthTokenParsing
Rules());
        subject = null;
        java.util.Vector newCreds = new java.util.Vector();
        newCreds.add(principal);
        subject = adf.getCreds(principal.getEntryName(), newCreds.toArray());
        try
        {
            if (!adf.decision(subject, action, target, null))
                return "1: the action is not allowed";

            else
                return "0: action succeeded";
        }
        catch (Exception ex)
        {
            throw new Exception(POLICY_LOAD_ERROR + " " + ex.Message);
        }

    }
```

Figure 6.10: Parsing delegatee's credentials

d - Computing delegatee's request

Once the delegatee has been successfully authenticated he will attempt to perform certain actions on the target. At each attempt, the ACM passes the target name and the attempted action along with its parameters, to the ADC via a call to Decision. Decision checks if the action is allowed for the roles that the delegatee has, taking into account all the conditions specified in the *TargetAccessPolicy*. If the action is allowed, Decision returns granted, if it is not allowed it returns denied.

The delegatee may attempt an arbitrary number of actions on different targets, and Decision is called for each one. In order to stop the delegatee keeping the connection open for an infinite amount of time (for example until after his ACs have expired), the PERMIS API supports the concept of a session time out. On the call to *GetCreds* the AMC can say how long the session may stay open before the credentials should be refreshed. If the session times out, its PDP will throw an exception, telling the AMC to either close the delegatee's connection or call *GetCreds* again.

6.4.4 Deployment and evaluation

Here, **we deploy the policy plugin within the authorisation decision engine PERMIS**. We firstly present the authorisation policy definition. We then specify the delegation policy and deploy it in the runtime environment based on the delegation scenario DS1. We conclude our test by analysing the results.

a - The authorisation policy

The authorisation policy specifies who has what type of access to which targets, and under which conditions. PERMIS uses the hierarchical RBAC model for specifying authorisations. We have specified an RBAC policy specifically designed for use with an X.509 attribute certificate based PMI. The top level X.509 PMI RBAC Policy is composed of a number of sub-policies as shown in the figure below. The domain of the PMI Policy is the union of all the domains of the subpolicies. Each policy is given a unique object identifier (OID) that globally unambiguously identifies it. This OID is passed to the Permis API by the caller, in order to guarantee that the correct policy will be used in all the subsequent access control decisions made by the API implementation.

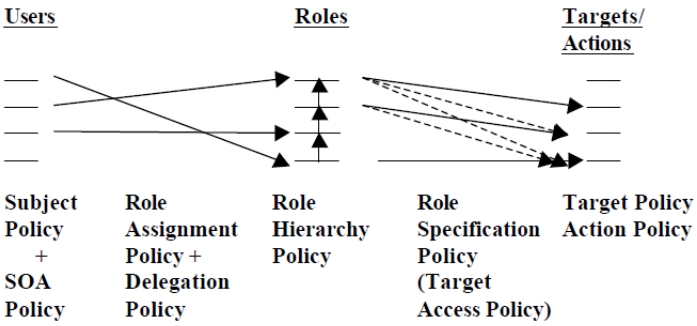

Figure 6.11: The X.509 PMI RBAC Policy

Data Type Definition (DTD) for the PERMIS policy is a meta-language that holds the rules for creating the XML policies. The DTD comprises the following components :

- SubjectPolicy : this specifies the subject domains i.e. only users from a subject domain may be authorised to access resources covered by the policy.
- RoleHierarchyPolicy : this specifies the different roles and their hierarchical relationships to each other.
- SOAPolicy : this specifies which SOAs are trusted to allocate roles, and permits the distributed managements of role allocation to take place.
- RoleAssignmentPolicy : this specifies which roles may be allocated to which subjects by which SOAs, whether delegation of roles may take place or not, and how long the roles may be assigned for.

- TargetPolicy : this specifies the target domains covered by this policy.
- ActionPolicy : this specifies the actions (or methods) supported by the targets, along with the parameters that should be passed along with each action e.g. action Open with parameter Filename.
- TargetAccessPolicy : this specifies which roles have permission to perform which actions on which targets, and under which conditions.

b - Deploying delegation policies

We use the PERMIS Policy Editor for creating and editing delegation policies (see figure 6.12). Our policy file (DS1.xml) specifies which action is allowed or denied based on the DS1 scenario. If a request is not relevant to the policy, it will be reported as an error encountered by the application. We define the delegator and the delegatee as *Prosecutor* and *Assistant*, respectively. We specify privileges for the delegator and give him the right to delegate the action *translate()* on the resource legal document *MLAdoc* (see DS1.0.xml policy file in figure 6.12).

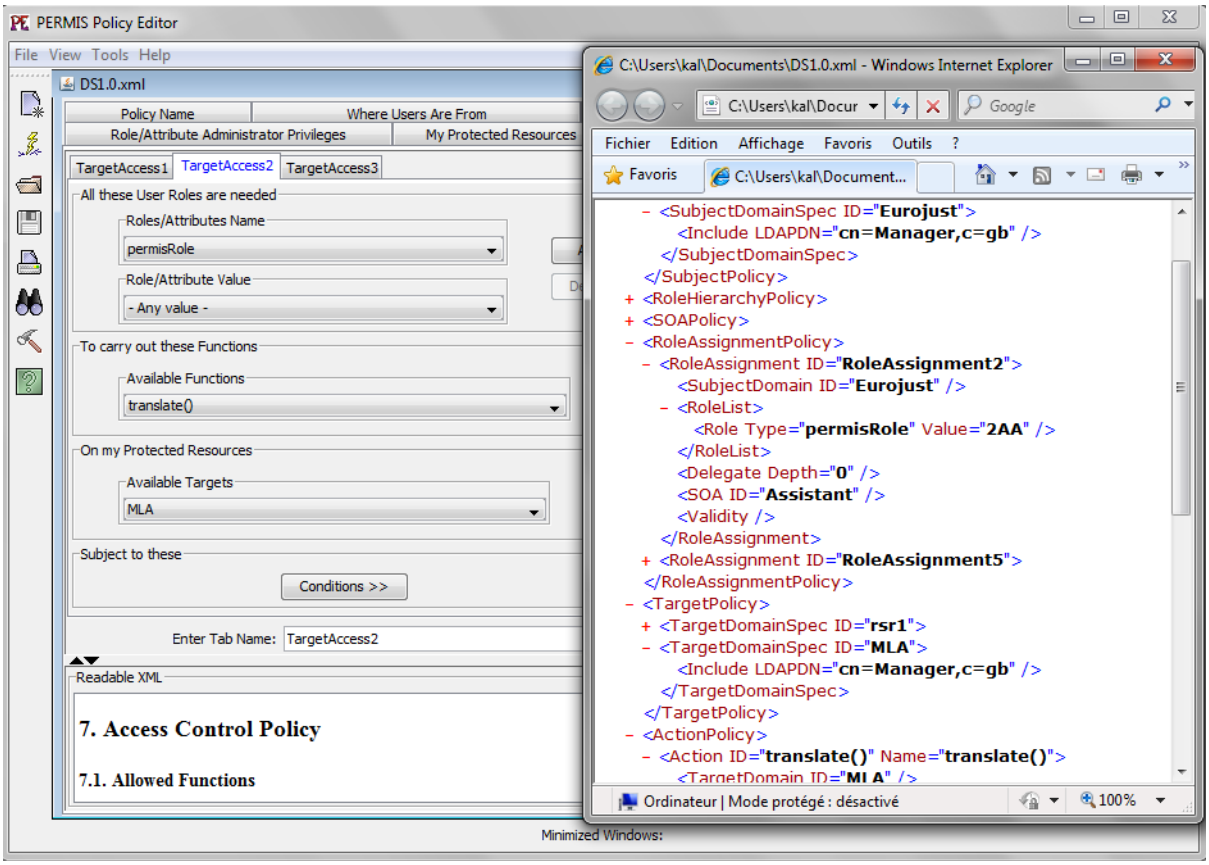

Figure 6.12: Policy editor for DS1

The delegatee member of role *Assistant* is assigned to this target and allowed to

execute it. However, he is not allowed to further delegate it due to privacy restrictions. Additional delegation constraints such as obligations, conditions or validity (deadline) can be specified based on the delegation requirements for this task.

In order to execute our delegation request, we use the Policy Tester which is a tool used to test PERMIS policies created by the Policy Editor. Test cases can be created to simulate any users accessing any resources under any conditions, and the users will then be granted or denied access according to the delegation policy that is read into the Policy Tester. The result of running the tests are shown in the output window of the Policy Tester (see figure 6.13). We illustrate the computation of the *Assistant* authorisation against to the DS1 policy. The delegatee attributes with the role *Assistant* is identified in the policy but his delegated privileges are limited to *translate()* in the defined target. Subsequently, the request of another target such as *MLA rsr1* resources on the service resource is not allowed (a deny decision).

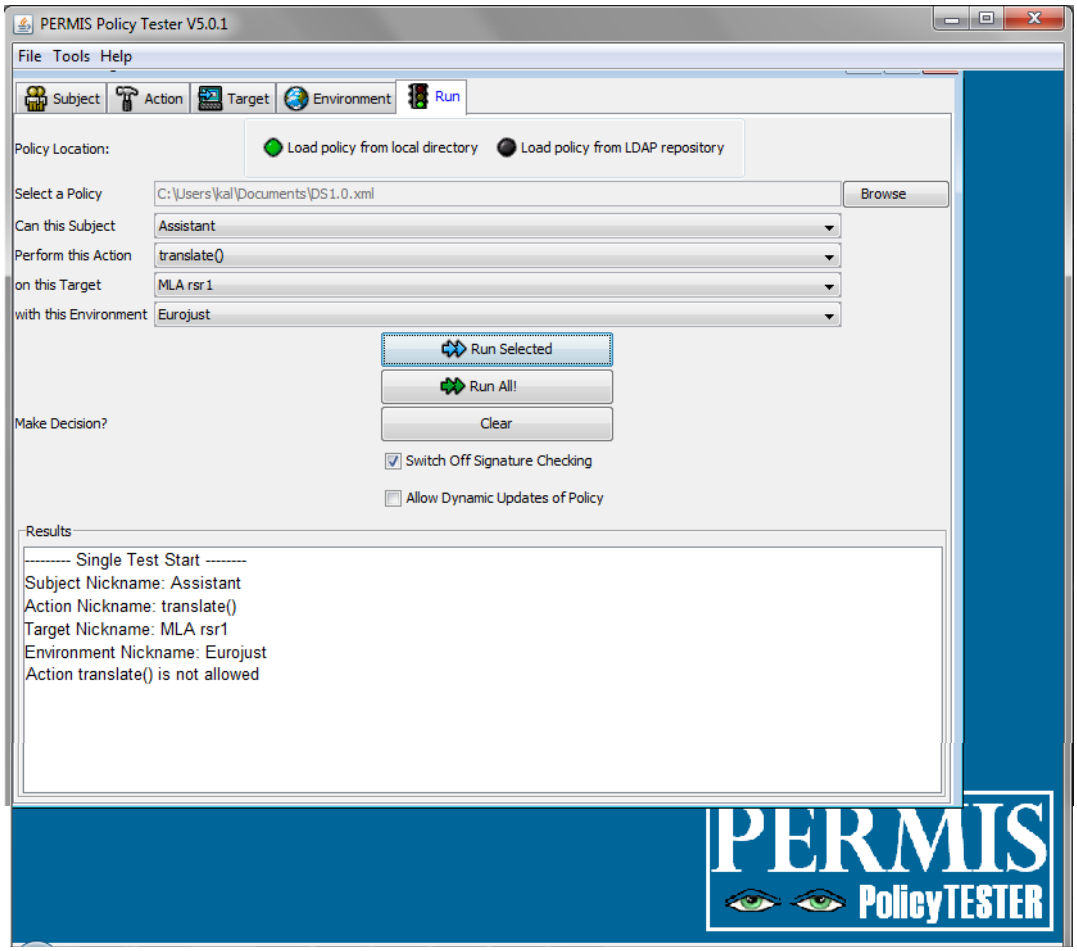

Figure 6.13: Policy tester for DS1

c - Evaluation and discussion

The PERMIS Policy Tester can also allow dynamic updates of policies. This offers a suitable solution to add new delegation rules that grant or revoke delegated privileges, thereby supporting dynamic delegation policies. However, this tool does not support event-based approach for proactive policies. The Contact Point/SOA defined in chapter 5 is not supported in this tool. For that reason, any further changes in policy will be made externally from PERMIS and does not depend on the event-based delegation approach. Moreover, the adding of delegation policies to existing policy cannot be checked. Thus, we may have conflicts with the predefined policy due to security constraints (e.g. SoD).

The next steps of our implementation will be the development of the delegation automation approach using *event calculus* in access control systems. Currently, we are looking to enrich our delegation policy plugin with an *event listener* component that permits to interact with the delegation execution and its policy definition based on the automation approach presented in chapter 5.

6.5 Conclusion

In this chapter, we have presented the implementation of a delegation tool within a workflow management system. We have introduced the development environment inspired from the R4eGov project. We then focused on the organisational aspect in order to **support human interactions in general and user-to-user delegation in particular**. We have integrated this aspect via an email plugin within the workflow runtime Bonita. Finally, we considered the security requirements to support delegation policies. We presented a solution to **integrate and test delegation policies dynamically**. To do so, we have implemented a policy plugin within the authorisation decision engine PERMIS and we have discussed its functionalities and limitations.

The deployment of the delegation framework presents, however, some limits. On the organisational level, we developed an e-mail centric solution for user-to-user delegation. Basically, we triggered delegation events using messaging Hook in the workflow system Bonita. However, this is not a standardised approach supporting a user-to-user communication within collaborative workflows. On the security level, we integrated delegation policies in a local manner and we did not test it in the global policy. Actually, PERMIS does not offer rich specifications for distributed resources and external roles when defining a policy. For that reason, we were not able to deploy the second delegation scenario DS2. Moreover, the security plugin does not support event-based approach for proactive policies. The Contact Point/SOA for event-based policy enforcement is not supported yet in our framework. Hence, any further changes in the policy does not depend on the event-based delegation approach and will be made externally from PERMIS. These limits will be a part of our future works.

Chapter 7

Conclusion and Perspectives

Providing access control mechanisms to secure task delegation in workflow management systems is a non-trivial task to model and engineer. In this dissertation, we have presented problems and requirements that such a model demands, and developed a solution to model task delegation within workflows and to specify delegation of authority into access control systems. The motivation of this direction is based on a real world process from an e-government scenario, where a task delegation is required and may support dynamic changes during execution. We aim to bridge the gap between organisational needs and security functionalities in workflow systems. To do so, we have considered two strong concepts namely workflow and access control systems in order to reach our objective. This can be detailed as follows :

- **Organisational flexibility :** An organisation obtains flexibility by offering means to manage human resources when dealing with user-o-user delegation. Accordingly, workflow's actors can evolve and change from the predefined workflow model. This can lead to a new rearrangement of users in order to ensure alternative scenarios by making the workflow more flexible and efficient.

- **Dynamic access control enforcement :** The idea is to give a coherent link from workflow modelling to access control requirements. Task delegation will enforce new authorisation requirements, thereby inquiring additional assignment and synchronisation for authorisation policies. In addition, any policy change due to delegation has to be computed and integrated in the existing policy automatically with regards to the organisational policy compliance.

Most of the work done in the area of workflow and access control systems does not treat delegation in sufficient details and deserves from us more investigations. On one hand, existing work in the domain of organisational management in workflows remain static and lack of flexibility [AW05, CK08a]. On the other hand, current solutions for access control are relatively stateless and rigid [SRS+05, BFA99]. At present, no mechanism exists that allows to ensure dynamic delegation of authority. Such a mechanism is vital for supporting a secure framework for dynamic task delegation in workflow management systems.

7.1 Thesis Summary and Contributions

The main contribution of this thesis is the development of a methodology with a supporting framework to secure dynamic task delegation within workflows. Securing delegation involves the definition of authorisation policies which are compliant with the existing policy of the workflow. To that end, we need to address two important issues, namely allowing the delegation completion, and having a secure delegation within a workflow. Allowing task delegation to complete requires a model that forms the basis of what can be analysed during the delegation process. Secure delegation implies the controlled propagation of authority during task execution. To ease the monitoring of task delegation and the propagation of delegation authority, we have presented an event-based task delegation model (TDM) amenable for supporting a framework to model, analyse and generate delegation policies. The thesis contributions can be summarised as follows :

- A **delegation model** supporting access control in business processes based on the task lifecycle. We have identified delegation interactions within workflow's layers namely task, control and data based on delegation events. Delegation events define the delegation process and build the task delegation model to be integrated in the business process.

- Delegation as an **advanced security mechanism** defines an enhanced access control model used to capture task requirements and to express it in terms of authorisation policy rules. We defined a Task-oriented Access Control (TAC) model with new modelling elements to enable the specification of access control requirements such as organisational roles and resources requirements between users and tasks. In addition, we presented a technique to optimise delegated privilege computation based on the delegated task instance defined in the TAC model.

- We have applied **formal methods** to integrate delegation policies and to detect compliance properties with the authorisation policy. We reasoned using event calculus formalism to address the delegation policy integration issue. Moreover, we came up with a technique to increase control and compliance of all delegation changes.

- Finally, we have presented an **access control framework** to specify delegation policies. We have architected a solution based on existing access control systems and extended it with a dynamic policy enforcement component for delegation policies.

Finally, we have presented the implementation of our framework within a workflow management system. We have integrated the organisational aspect via a task management plugin within the workflow runtime Bonita in order to support human interactions in general and user-to-user delegation in particular. We then focused on the security requirements to support delegation policies. We presented a solution to integrate and to test delegation policies dynamically within the authorisation decision engine PERMIS.

7.2 Limits and Perspectives

Throughout this thesis, we have considered task delegation as an atomic unit of work. We do not consider sub tasks delegation where delegation can be done partially. In addition, we assumed that user-to-user delegation is exclusive to the delegator and the delegatee and that there is no further delegations supporting cascaded or multistep delegation [ZAC03]. In the context of human-to-human interactions, we argued that a delegated task is atomic when considering the user-to-user delegation protocol. The protocol introduced steps such as negotiation, acceptance and declination to give more choice and flexibility when defining a delegation request. Negotiation includes facts (i.e. evidence, deadline, privileges) that were agreed between delegation principals which are exclusive to them and can not be shared by additional delegatees. For instance, if a delegatee agreed on a time deadline to perform a task, he is not allowed to extend this deadline and delegates it further to another delegatee. In addition, such a delegation may be a security threat since delegated privileges are granted based on the delegatee's role in the organisation and his associated permissions in the authorisation policy.

Moreover, the deployment of the delegation framework presents some limits. On the organisational level, the solution for user-to-user delegation using messaging Hook is not a standardised approach within workflow systems. On the security level, the integration of delegation policies is done locally and does not support distributed resources and external roles when considering the second delegation scenario cross organisations. Besides that, the security plugin does not support event-based approach for proactive policies. Hence, any further changes in the authorisation policy does not depend on the event-based delegation approach and will be made externally from the access control framework. These discussed limits will be a part of our future works.

Future perspectives can be considered in this dissertation. The next steps of our implementation will be the development of the delegation automation approach using *event calculus*. We aim to enrich the delegation policy component with an *event listener* component in the access control framework. Having an *event listener* component will allow to synchronise between the policy definition and the automation approach. In addition, we will work on implementing our framework using XACML standard. We aim to enrich delegation policy constraints with XACML elements such as condition and obligation. The PERMIS decision engine has being enhanced to support the XACML request context and to return the XACML response context. This will present an interest for our future works to support features that are not currently supported by XACML, such as dynamic delegation of authority [CON09].

In addition, we will look at enriching our approach with additional delegation constraints supporting historical records. Delegation history will be used to record all delegations that have been made to address administrative requirements such as auditing. In general, the encoding process does not scale well especially with the increase in *timepoints*. Our reasoning model will use historical records to guarantee that delegations are enforced correctly. We aim to modify the proposed DECreasoner encoding to make the process faster. A key observation is that it always takes less time than the initial solution as we do have a partial plan and that reduces the problem. Subsequently, we can leverage the trace of encoding in DECReasoner to give all necessary information (events, fluents and

timepoints) to detect policies problems (e.g. indeterminate rule). To that end, we will use an auditing technique allowing us to choose the best candidate for delegation based on the delegatee's historical performance.

Finally, we consider as a future work delegation in Web services (WS). Web service is the emerging standard that supports the seamless interoperation between different applications. While the interoperability, flexibility and automated composition are continuously enhanced, security is still the major hurdle. In recent years, lots of studies have been conducted in web service security and various security standards have been proposed [STY07, BSM04, BS00]. But most of these studies and standards focus on the access control policies for individual web services and do not consider the access issues in composed services. However, problems such as how much privilege to delegate, how to confirm cross-domain delegation, how to delegate additional privilege when needed arise. WS Policy specifies a framework for expressing web service constraints and requirements as authorisation policies using policy assertions. WS Security Policy extends WS security by specifying the policy assertions to describe security policies [Asi07]. We would propose a delegation-based security model to address all these issues. It will extend the basic security model defined in WS Policy and will support flexible delegation and evaluation-based access control model for web services.

Appendix A

Discrete Event Calculus Reasoner

A.1 Introduction

The Discrete Event Calculus Reasoner is a program for performing automated commonsense reasoning using the discrete event calculus [Mue06], a version of the classical logic event calculus [KS89]. The program supports such types of reasoning as deduction, temporal projection, abduction, planning, postdiction, and model finding. The Discrete Event Calculus Reasoner supports the following commonsense phenomena :

- The *commonsense law of inertia*, which states that an event typically changes only a small number of things and everything else in the world remains unchanged. For example, moving
- *Conditional effects of events*. For example, the results of turning on a television set depend on whether or not it is plugged in.
- *Release from the commonsense law of inertia*. For example, if a person is holding a glass, then the location of the glass is released from the commonsense law of inertia so that the location of the glass is permitted to vary.
- *Event ramifications* or indirect effects of events. The tool supports state constraints. For example, a glass moves along with the person holding it. The tool supports causal constraints, which deal with the instantaneous propagation of interacting indirect effects, as in idealised electrical circuits.
- *Events with nondeterministic* effects. For example, flipping a coin results in the coin landing either heads or tails.
- *Gradual change* such as the changing height of a falling object or volume of a balloon in the process of inflation.
- *Triggered events* or events that are triggered under certain conditions. For example, if water is owing from a faucet into a sink, then once the water reaches a certain level the water will overflow.

- *Concurrent events* with cumulative or canceling effects. For example, if a shopping cart is simultaneously pulled and pushed, then it will spin around.

Here is how you use the Discrete Event Calculus Reasoner. First, you place a domain description into a file. The domain description consists of :

- an *axiomatization* describing a commonsense domain or domains of interest,
- *observations* of world properties at various times, and
- a *narrative* of known event occurrences.

The domain description is expressed using the Discrete Event Calculus Reasoner language. Then, you invoke the Discrete Event Calculus Reasoner on the domain description. The program transforms the domain description into a satisfability (SAT) problem. The SAT problem is expressed in the standard format used by most SAT solvers [Eri08]. The program then runs a SAT solver, which produces zero or more solutions, called models. The program decodes these models and displays them.

A.2 Discrete Event Calculus Reasoner Language

In this section, we describe the Discrete Event Calculus Reasoner in more detail.

A.2.1 Sorts

Sentences are expressed in the language of many-sorted first-order predicate calculus with subsort orders. This means that:

- sorts can be subsorts of other sorts,
- every variable, constant, and function symbol has an associated sort,
- every argument position of every function and predicate symbol has an associated sort, and
- for a term to fill an argument position of a function or predicate symbol, the sort associated with the term must be a subsort of the sort associated with the argument position.

We define a sort called object as follows :

```
sort object
```

We define a sort agent that is a subsort of object as follows :

```
sort agent:  object
```

The sort associated with a variable is determined by removing digits from the variable. For example, the sort of the variable `snowflake72` is `snowflake`. A constant's sort is specified when the constant is defined. For example, the following defines three constants whose sort is agent :

`agent Fred, Annie, Thea`
Each object in the world is assumed to be named by a unique constant. That is, the constants `Annie` and `Thea` do not refer to the same person. This is known as the unique names assumption. Here is a definition of a function symbol `Floor` whose sort is `integer` and whose first and only argument position is of sort room :
`function Floor(room): integer` Here is a definition of a predicate symbol `PartOf` that takes two arguments of sort `physobj` and `object` :
`predicate PartOf(physobj,object)`

A.2.2 Formulas

The following grammar, which is based on that of the Bliksem theorem prover [Han99], in conjunction with the sort constraints described above, specifies how sentences are constructed as follows :

term ::= *variable* | *constant* |
functionsymbol`(`*arguments*`)` |
term `+` *term* | *term* `-` *term* | *term* `*` *term* | *term* `/` *term* |
term `%` *term* | `-` *term* |
`(`*term*`)` | *reifiedformula*
formula ::= *predicatesymbol*`(`*arguments*`)` |
term `<` *term* | *term* `<=` *term* | *term* `=` *term* |
term `>=` *term* | *term* `>` *term* | *term* `!=` *term* |
formula `|` *formula* | *formula* `&` *formula* | `!` *formula* |
formula `->` *formula* | *formula* `<->` *formula* |
`{`*variables*`}` *formula* | `[`*variables*`]` *formula* |
`(`*formula*`)`
reifiedformula ::= *formula*
arguments ::= *term* | *arguments*`,` *term*
variables ::= *variable* | *variables*`,` *variable*

A *variable* consists of one or more lowercase letters followed by zero or more digits. A *constant* consists of (a) one or more digits or (b) an uppercase letter followed by zero or more letters or digits. *functionsymbols* and *predicatesymbols* consist of an uppercase letter followed by zero or more letters or digits. *Fluent* and *event* symbols are predicate symbols. The meaning of the symbols is defined in the table A.2.2 :

A.2.3 Options

The option statement can be used to specify the values of certain program options [Eri08]. The syntax of this statement is :
`option optionname optionvalue`
The following optionnames are supported :
`debug` If the *optionvalue* is on, detailed debugging output is produced. The default value

Symbol	Meaning	Symbol	Meaning
`+`	addition	`!=`	not equal to
`-`	subtraction, negation	`|`	disjunction (OR, $\vee$)
`*`	multiplication	`&`	conjunction (AND, $\wedge$)
`/`	division	`!`	logical negation
`%`	modulus	`->`	implication
`<`	less than	`<->`	bi-implication
`<=`	less than or equal to	`{ }`	existential quantification ($\exists$)
`=`	equal to	`[ ]`	universal quantification ($\forall$)
`>=`	greater than or equal to	`( )`	grouping
`>`	greater than	`,`	separator

Table A.1: The meaning of the symbols in the DECReasoner language

of this option is off.
`encoding` The value of this option specifies the event calculus-to-SAT encoding method.
`finalstatefile` The value of this option specifies the name of a file to which the final state of the run will be written.
`manualrelease` If *on*, automatic generation of assertions of the form `!Released(fluent,0)` is inhibited, so that such assertions can be added manually on a case-by-case basis. The default value of this option is *off*.
`modeldiff` If *on*, differences from one model to the next are shown, instead of complete models. The default value of this option is *off*.
`renaming` If *on*, the technique of renaming subformulas is used to convert to a compact conjunctive normal. The default value of this option is *on*.
`showpred` If *on*, the truth values of all predicates other than fluents and events are shown. The default value of this option is *on*.
`solver` The value of this option specifies the name of the solver to use. The default value of this option is *relsat*.
`timediff` If *on*, differences from one timepoint to the next are shown, instead of complete states. The default value of this option is *on*.
`tmpfilerm` If *on*, temporary files, which are stored in the temporary directory, are removed at the end of each run. The default value of this option is *on*.
`trajectory` If *on*, Trajectory axioms are supported. The default value of this option is *off*.

Appendix B

Commonsense Reasoning for Task Delegation

The Verification results and encoding details of the delegation scenario DS1 using our reasoning model in DECReasoner is detailed as follows :

```
load foundations/Root.e
load foundations/EC.e

sort task, effect, condition, obligation

task T3
effect Permit, Deny
condition Push, Pull
obligation Grant, Transer, Evidence, NoEvidence}

fluent  Initial(task)
fluent  Assigned(task)
fluent  WaitingDelegation(task)
fluent  WaitingCompletion(task)
fluent  Delegated(task)
fluent  Started(task)
fluent  WaitingValidation(task)
fluent  Failed(task)
fluent  Completed(task)
fluent  Cancelled(task)

fluent RuleAdded(effect, condition, obligation)
event AddPoliycRule(effect, condition, obligation)

[effect, condition, obligation, time]
Initiates(AddPoliycRule(effect, condition, obligation) ,
RuleAdded(effect, condition, obligation) ,time).

;[task, effect, condition, obligation, time1, time2]
```

```
Happens(PushDelegateAcceptExecuteGrant(task), time1) & time2 != time1 ->
!Happens(AddPoliycRule(effect, condition, obligation), time2).
;[task, effect, condition, obligation, time]
Happens(PushDelegateAcceptExecuteGrant(task), time) & effect != Permit
& condition != Push & obligation != Evidnce ->
!Happens(AddPoliycRule(effect, condition, obligation), time).

event Create(task) ;Initial
[task, time] Initiates(Create(task), Initial(task) ,time).

event Assign(task) ;Assigned ; once created
[task, time]Initiates(Assign(task), Assigned(task) ,time).
[task, time1]  Happens(Assign(task), time1) ->
{time2} HoldsAt(Initial(task), time2) & time1 > time2.

event PushDelegate(task) ; WaitingDelegation; once assigned
[task, time] Initiates(PushDelegate(task), WaitingDelegation(task) ,time).
[task, time1]  Happens(PushDelegate(task), time1) ->
{time2} HoldsAt(Assigned(task), time2) & time1 > time2.

event PushDelegateCancel(task) ; Assigned; once WaitingDelegation
[task, time] Initiates(PushDelegateCancel(task), Assigned(task) ,time).
[task, time1]  Happens(PushDelegateCancel(task), time1) ->
{time2} HoldsAt(WaitingDelegation(task), time2) & time1 > time2.

event PushDelegateAccept(task) ; WaitingCompletion; once waitingDelegation
[task, time] Initiates(PushDelegateAccept(task),
WaitingCompletion(task) ,time).
[task, time1]  Happens(PushDelegateAccept(task), time1) ->
{time2} HoldsAt(WaitingDelegation(task), time2) & time1 > time2.


event PullDelegate(task) ; Delegated ; once Assigned
[task, time] Initiates(PullDelegate(task), Delegated(task) ,time).
[task, time1]  Happens(PullDelegate(task), time1) ->
{time2} HoldsAt(Assigned(task), time2) & time1 > time2.

event Start(task) ;  ;Started ; once Assigned
[task, time] Initiates(Start(task), Started(task) ,time).
[task, time1]  Happens(Start(task), time1) ->
{time2} HoldsAt(Assigned(task), time2) & time1 > time2.

event PushDelegateAcceptRevoke(task) ; Assigned; once waitingCompletion
[task, time] Initiates(PushDelegateAcceptRevoke(task),
Assigned(task) ,time).
[task, time1]  Happens(PushDelegateAcceptRevoke(task), time1) ->
{time2} HoldsAt(WaitingCompletion(task), time2) & time1 > time2.
```

```
event PushDelegateAcceptExecuteGrant(task) ; WaitingValidation;
once waitingCompletion
[task, time] Initiates(PushDelegateAcceptExecuteGrant(task),
WaitingValidation(task) ,time).
[task, time1]  Happens(PushDelegateAcceptExecuteGrant(task), time1) ->
{time2} HoldsAt(WaitingCompletion(task), time2) & time1 > time2.

event PushDelegateAcceptFailTransfer(task) ; Failed;
once waitingCompletion
[task, time] Initiates(PushDelegateAcceptFailTransfer(task),
Failed(task) ,time).
[task, time1]  Happens(PushDelegateAcceptFailTransfer(task), time1) ->
{time2} HoldsAt(WaitingCompletion(task), time2) & time1 > time2.

event PushDelegateAcceptCompleteTransfer(task) ; Completed;
once waitingCompletion
[task, time] Initiates(PushDelegateAcceptCompleteTransfer(task),
Completed(task) ,time).
[task, time1]  Happens(PushDelegateAcceptCompleteTransfer(task),time1)
-> {time2} HoldsAt(WaitingCompletion(task), time2) & time1 > time2.

event PullDelegateStart(task); Started; once delegated
[task, time] Initiates(PullDelegateStart(task), Started(task) ,time).
[task, time1]  Happens(PullDelegateStart(task), time1) ->
{time2} HoldsAt(Delegated(task), time2) & time1 > time2.

event PullDelegateStartExecuteGrant(task);WaitingValidation,
Once Started,
once pull delegated,
[task, time] Initiates(PullDelegateStartExecuteGrant(task),
WaitingValidation(task) ,time).
[task, time1]  Happens(PullDelegateStartExecuteGrant(task), time1)
-> {time2} HoldsAt(Delegated(task), time2) & time1 > time2.
[task, time1]  Happens(PullDelegateStartExecuteGrant(task), time1)
-> {time2} HoldsAt(Started(task), time2) & time1 > time2.

event Complete(task); Completed; once Started
[task, time] Initiates(Complete(task), Completed(task) ,time).
[task, time1]  Happens(Complete(task), time1) ->
{time2} HoldsAt(Started(task), time2) & time1 > time2.

event Fail(task); Failed; once started
[task, time] Initiates(Fail(task), Failed(task) ,time).
[task, time1]  Happens(Fail(task), time1) ->
{time2} HoldsAt(Started(task), time2) & time1 > time2.
```

```
event Abort(task); Cancelled; once assigned
[task, time] Initiates(Abort(task), Cancelled(task) ,time).
[task, time1]  Happens(Abort(task), time1) ->
{time2} HoldsAt(Assigned(task), time2) & time1 > time2.

event PullDelegateStartExecuteGrantRevoke(task); Failed ;
once WaitingValidation, once pull delegated
[task, time] Initiates(PullDelegateStartExecuteGrantRevoke(task),
Failed(task) ,time).
[task, time1]  Happens(PullDelegateStartExecuteGrantRevoke(task),time1)
-> {time2} HoldsAt(WaitingValidation(task), time2) & time1 > time2.
[task, time1]  Happens(PullDelegateStartExecuteGrantRevoke(task), time1)
-> {time2} HoldsAt(Delegated(task), time2) & time1 > time2.

event PushDelegateAcceptExecuteGrantValidate(task); Completed;
once WaitingValidation, once WaitingCompletion
[task, time] Initiates(PushDelegateAcceptExecuteGrantValidate(task),
Completed(task) ,time).
[task, time1] Happens(PushDelegateAcceptExecuteGrantValidate(task),time1)
-> {time2} HoldsAt(WaitingValidation(task), time2) & time1 > time2.

[task, time1] Happens(PushDelegateAcceptExecuteGrantValidate(task),time1)
-> {time2} HoldsAt(WaitingCompletion(task), time2) & time1 > time2.

;Ordering
;----------

[task, time] HoldsAt(Initial(task), time) -> !Happens(Create(task), time)
[task, time] HoldsAt(Assigned(task), time) -> !Happens(Assign(task), time)
[task, time] HoldsAt(WaitingDelegation(task), time) ->
!Happens(PushDelegate(task), time).
[task, time] HoldsAt(WaitingCompletion(task), time) ->
!Happens(PushDelegateAccept(task), time).
[task, time] HoldsAt(Delegated(task), time) ->
!Happens(PullDelegate(task), time).
[task, time] HoldsAt(Started(task), time) ->
!Happens(Start(task), time).

;event PushDelegateAcceptRevoke(task) ; Assigned; once waitingCompletion

[task, time] HoldsAt(WaitingValidation(task), time)
-> !Happens(PushDelegateAcceptExecuteGrant(task), time).
[task, time] HoldsAt(Failed(task), time)    ->
!Happens(PushDelegateAcceptFailTransfer(task), time).
[task, time] HoldsAt(Completed(task), time)    ->
!Happens(PushDelegateAcceptCompleteTransfer(task), time).
[task, time] HoldsAt(Started(task), time)    ->
```

```
!Happens(PullDelegateStart(task), time).
[task, time] HoldsAt(WaitingValidation(task), time)    ->
!Happens(PullDelegateStartExecuteGrant(task), time).
[task, time] HoldsAt(Completed(task), time)    ->
!Happens(Complete(task), time).
[task, time] HoldsAt(Failed(task), time)    ->
!Happens(Fail(task), time).
[task, time] HoldsAt(Cancelled(task), time)    ->
!Happens(Abort(task), time).
[task, time] HoldsAt(Failed(task), time)    ->
!Happens(PullDelegateStartExecuteGrantRevoke(task), time).
[task, time] HoldsAt(Completed(task), time)    ->
!Happens(PushDelegateAcceptExecuteGrantValidate(task), time).

;Initialisations
;------------------
[task] !HoldsAt(Initial(task),0).
[task] !HoldsAt(Assigned(task),0).
[task] !HoldsAt(WaitingDelegation(task),0).
[task] !HoldsAt(WaitingCompletion(task),0).
[task] !HoldsAt(Delegated(task),0).
[task] !HoldsAt(Started(task),0).
[task] !HoldsAt(WaitingValidation(task),0).
[task] !HoldsAt(Failed(task),0).
[task] !HoldsAt(Completed(task),0).
[task] !HoldsAt(Cancelled(task),0).
[effect, condition, obligation]
!HoldsAt(RuleAdded(effect, condition, obligation),0).

;Implicit Orderings
;------------------
[task, time1, time2] Happens(Abort(task), time1) & time2 >= time1 ->
!Happens(Start(task), time2) & !Happens(PullDelegate(task), time2)
& !Happens(PushDelegate(task), time2).

[task, time1, time2] Happens(PushDelegate(task), time1) & time2>=time1
-> !Happens(Start(task), time2) & !Happens(PullDelegate(task), time2)
& !Happens(Abort(task), time2).
[task, time1, time2] Happens(PullDelegate(task), time1) & time2>=time1
-> !Happens(Start(task), time2) & !Happens(PushDelegate(task), time2)
& !Happens(Abort(task), time2).
[task, time1, time2] Happens(Start(task), time1) & time2 >= time1 ->
!Happens(PullDelegate(task), time2) & !Happens(PushDelegate(task), time2)
& !Happens(Abort(task), time2).

;PushDelegateAcceptExecuteGrant
;PushDelegateAcceptFailTransfer
```

```
;PushDelegateAcceptCompleteTransfer
;PushDelegateAcceptRevoke

[task, time1, time2] Happens(PushDelegateAcceptExecuteGrant(task), time1)
& time2 >= time1 ->
!Happens(PushDelegateAcceptFailTransfer(task), time2)
& !Happens(PushDelegateAcceptCompleteTransfer(task), time2)
& !Happens(PushDelegateAcceptRevoke(task), time2).

[task, time1, time2] Happens(PushDelegateAcceptFailTransfer(task), time1)
& time2 >= time1 ->
!Happens(PushDelegateAcceptExecuteGrant(task), time2)
& !Happens(PushDelegateAcceptCompleteTransfer(task), time2)
& !Happens(PushDelegateAcceptRevoke(task), time2).

[task, time1, time2] Happens(PushDelegateAcceptCompleteTransfer(task),time1)
& time2 >= time1 ->
!Happens(PushDelegateAcceptExecuteGrant(task), time2)
& !Happens(PushDelegateAcceptFailTransfer(task), time2)
& !Happens(PushDelegateAcceptRevoke(task), time2).

[task, time1, time2] Happens(PushDelegateAcceptRevoke(task), time1)
& time2 >= time1 ->
!Happens(PushDelegateAcceptExecuteGrant(task), time2)
& !Happens(PushDelegateAcceptFailTransfer(task), time2)
& !Happens(PushDelegateAcceptCompleteTransfer(task), time2).

;Happens axioms to force reasoner to choose delegation
;-----------------------------------------------------
;which delegation mode, ater assignment
[task, time] !Happens(Abort(task), time).
[task, time] !Happens(Start(task), time).
[task, time] !Happens(PullDelegate(task), time).
;[task, time] !Happens(PushDelegate(task), time).

;which push delegation mode
[task, time] !Happens(PushDelegateAcceptExecuteGrant(task), time).
;[task, time] !Happens(PushDelegateAcceptFailTransfer(task), time).
[task, time] !Happens(PushDelegateAcceptCompleteTransfer(task), time).

;we dont want delagation cancelled and revoked
[task, time] !Happens(PushDelegateAcceptRevoke(task), time).
[task, time] !Happens(PushDelegateCancel(task), time).

;Policy updations
;----------------
[task, effect, condition, obligation, time]
```

```
Happens(AddPoliycRule(effect, condition, obligation), time) ->
Happens(PushDelegateAcceptExecuteGrant(task), time) |
Happens(PushDelegateAcceptFailTransfer(task), time) |
Happens(PushDelegateAcceptCompleteTransfer(task), time).
[task, time]  Happens(PushDelegateAcceptExecuteGrant(task), time)
-> Happens(AddPoliycRule(Permit, Push, Evidence), time)
& !Happens(AddPoliycRule(Deny, Push, Evidence), time)
& !Happens(AddPoliycRule(Deny, Push, Transer), time)
& !Happens(AddPoliycRule(Permit, Pull, Grant), time)
& !Happens(AddPoliycRule(Permit, Push, Transer), time)
& !Happens(AddPoliycRule(Deny, Push, Grant), time)
& !Happens(AddPoliycRule(Deny, Push, NoEvidence), time)
& !Happens(AddPoliycRule(Permit, Pull, Evidence), time)
& !Happens(AddPoliycRule(Permit, Push, Grant), time)
& !Happens(AddPoliycRule(Permit, Push, NoEvidence), time)
& !Happens(AddPoliycRule(Deny, Pull, NoEvidence), time)
& !Happens(AddPoliycRule(Permit, Pull, Transer), time)
& !Happens(AddPoliycRule(Deny, Pull, Grant), time)
& !Happens(AddPoliycRule(Deny, Pull, Transer), time)
& !Happens(AddPoliycRule(Permit, Pull, NoEvidence), time)
& !Happens(AddPoliycRule(Deny, Pull, Evidence), time).

[task, time]  Happens(PushDelegateAcceptFailTransfer(task), time)
-> Happens(AddPoliycRule(Deny, Push, Transer), time)
& !Happens(AddPoliycRule(Deny, Push, Evidence), time)
& !Happens(AddPoliycRule(Permit, Push, Evidence), time)
& !Happens(AddPoliycRule(Permit, Pull, Grant), time)
& !Happens(AddPoliycRule(Permit, Push, Transer), time)
& !Happens(AddPoliycRule(Deny, Push, Grant), time)
& !Happens(AddPoliycRule(Deny, Push, NoEvidence), time)
& !Happens(AddPoliycRule(Permit, Pull, Evidence), time)
& !Happens(AddPoliycRule(Permit, Push, Grant), time)
& !Happens(AddPoliycRule(Permit, Push, NoEvidence), time)
& !Happens(AddPoliycRule(Deny, Pull, NoEvidence), time)
& !Happens(AddPoliycRule(Permit, Pull, Transer), time)
& !Happens(AddPoliycRule(Deny, Pull, Grant), time)
& !Happens(AddPoliycRule(Deny, Pull, Transer), time)
& !Happens(AddPoliycRule(Permit, Pull, NoEvidence), time)
& !Happens(AddPoliycRule(Deny, Pull, Evidence), time).

[task, time]  Happens(PushDelegateAcceptCompleteTransfer(task), time)
-> Happens(AddPoliycRule(Permit, Push, NoEvidence), time)
& !Happens(AddPoliycRule(Deny, Push, Evidence), time)
& !Happens(AddPoliycRule(Permit, Push, Evidence), time)
& !Happens(AddPoliycRule(Permit, Pull, Grant), time)
& !Happens(AddPoliycRule(Permit, Push, Transer), time)
& !Happens(AddPoliycRule(Deny, Push, Grant), time)
```

```
& !Happens(AddPoliycRule(Deny, Push, NoEvidence), time)
& !Happens(AddPoliycRule(Permit, Pull, Evidence), time)
& !Happens(AddPoliycRule(Permit, Push, Grant), time)
& !Happens(AddPoliycRule(Deny, Push, Transer), time)
& !Happens(AddPoliycRule(Deny, Pull, NoEvidence), time)
& !Happens(AddPoliycRule(Permit, Pull, Transer), time)
& !Happens(AddPoliycRule(Deny, Pull, Grant), time)
& !Happens(AddPoliycRule(Deny, Pull, Transer), time)
& !Happens(AddPoliycRule(Permit, Pull, NoEvidence), time)
& !Happens(AddPoliycRule(Deny, Pull, Evidence), time).

;[task, time] Happens(PushDelegateAcceptExecuteGrant(task), time)
-> Happens(AddPolicyRule().

;Goal
;----------

[task] HoldsAt(Completed(task),15) | HoldsAt(Failed(task),15) |
HoldsAt(Cancelled(task),15).

range time 0 20
range offset 1 1
option showpred off
```

Appendix C

List of Acronyms

ACF	Access Control Framework
ACL	Administrative Communication Layer
ADC	Authorisation Decision Component
AMC	Authentication Management Component
BoD	Binding of Duty
BPM	Business Process management
CMS	Case management System
DAC	Discretionary Access Control
DC	Delegation Constraints
DPolicy	Delegation Policy
DR	Delegation Relation
DS	Delegation Scenario
EC	Event Calculus
e-Government	Electronic Government
Eurojust	European Judicial Cooperation Unit
Europol	European Police Office
JAO	Judicial Authority Officer
MAC	Mandatory Access Control
MLA	Mutual Legal Assistance
NC	National Correspondent
OU	Organisation Unit
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
RBAC	Role Based Access Control model
RH	Role Hierarchy
RM	Role Mapping
RPA	Role Permission Assignment
SoD	Separation of Duty

SR	A function mapping each subject (S) to a set of roles (R)
SU	A function mapping a subject (S) to the corresponding user (U)
TAC	Task-oriented Access Control model
SU	A function mapping a subject (S) to the corresponding user (U)
TAC	Task-oriented Access Control model
TDM	Task Delegation Model
TI	Task Instance
TPA	Task Permission Assignment relation
TRA	Task Role Assignment relation
URA	User Role Assignment relation
WfMC	Workflow Management Coalition
WfMS	Workflow Management System
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language

Bibliography

[AHW03] Wil M. P. Van Der Aalst, Arthur H. M. Hofstede, and Mathias Weske. Business process management: A survey. In *Business Process Management, International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003, Proceedings*, pages 1–12. Springer, 2003.

[Ari02] Arindam Banerji, Claudio Bartolini, Dorothea Beringer, Venkatesh Chopella, Kannan Govindarajan, Alan Karp, Harumi Kuno, Mike Lemon, Gregory Pogossiants, Shamik Sharma, Scott Williams. Web Services Conversation Language (WSCL) 1.0, 2002. World Wide Web Consortium (W3C), Palo Alto, CA, USA.

[Asi07] Asir S Vedamuthu, David Orchard, Frederick Hirsch, Maryann Hondo, Prasad Yendluri, Toufic Boubez. Web Services Policy 1.5 - Framework, 04 September 2007. W3C Recommendation.

[ASL08] Mickael von Riegen Andreas Schaad, Khaled Gaaloul and Hanna Lee. Specification of advanced security and privacy mechanisms. Technical report, 6th Framework Programme, Information Society Technologies, R4eGov, April 2008.

[AW05] Vijayalakshmi Atluri and Janice Warner. Supporting conditional delegation in secure workflow management systems. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 49–58, New York, NY, USA, 2005. ACM.

[Bal98] Raman Balasubramanian. Adding workflow analysis techniques to the is development toolkit. In *HICSS '98: Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences-Volume 4*, page 312, Washington, DC, USA, 1998. IEEE Computer Society.

[Bar03] Albert-László Barabási. Linked: The new science of networks. *J. Artificial Societies and Social Simulation*, 6(2), 2003.

[BCFM00] Elisa Bertino, Silvana Castano, Elena Ferrari, and Marco Mesiti. Specifying and enforcing access control policies for xml document sources. *World Wide Web*, 3(3):139–151, 2000.

[BE01] Reinhardt A. Botha and Jan H. P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001.

[BFA99] Elisa Bertino, Elena Ferrari, and Vijayalakshmi Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.*, 2(1):65–104, 1999.

[BGMG02] Jörg Becker, Marc Gille, Michael Zur Muehlen, and Dr. Marc Gille. Workflow application architectures: Classification and characteristics of workflow-based information systems. In *in: Fischer, L. (Ed.): Workflow Handbook 2002, Future Strategies, Lighthouse Point, FL*, pages 39–50, 2002.

[Bib75] Kenneth J. Biba. Integrity considerations for secure computer systems. technical report mtr-3153, mitre corporation, 1975.

[BN89] Dr. David F.C. Brewer and Dr. Micheal J. Nash. The chinese wall security policy. *Security and Privacy, IEEE Symposium on*, 0:206, 1989.

[BS00] E. Barka and R. Sandhu. Framework for role-based delegation models. In *Proceedings of the 16th Annual Computer Security Applications Conference*, pages 168–176, Washington, DC, USA, 2000. IEEE Computer Society.

[BSM04] E. Bertino, A. C. Squicciarini, and D. Mevi. A fine-grained access control model for web services. In *SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing*, pages 33–40, Washington, DC, USA, 2004. IEEE Computer Society.

[CK06] Jason Crampton and Hemanth Khambhammettu. Delegation in role-based access control. In *Proceedings of the Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006*, Lecture Notes in Computer Science, pages 174–191. Springer, 2006.

[CK08a] Jason Crampton and Hemanth Khambhammettu. Delegation and satisfiability in workflow systems. In *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 31–40, New York, NY, USA, 2008. ACM.

[CK08b] Jason Crampton and Hemanth Khambhammettu. On delegation and workflow execution models. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 2137–2144, New York, NY, USA, 2008. ACM.

[CKO92] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Commun. ACM*, 35(9):75–90, 1992.

[CO02] David W. Chadwick and Alexander Otenko. The permis x.509 role based privilege management infrastructure. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 135–140, New York, NY, USA, 2002. ACM.

[Con02] Connie Moore. Common Mistakes in Workflow Implementations, 2002. Cambridge, MA: Giga Information Group. IdeaByte, RIB-062002-00118.

[CON06] David W. Chadwick, Sassa Otenko, and Tuan-Anh Nguyen. Adding support to xacml for dynamic delegation of authority in multiple domains. In *Communications and Multimedia Security, 10th IFIP TC-6 TC-11 International Conference, CMS 2006, Heraklion, Crete, Greece, October 19-21, 2006*, pages 67–86, 2006.

[CON09] David W. Chadwick, Sassa Otenko, and Tuan-Anh Nguyen. Adding support to xacml for multi-domain user to user dynamic delegation of authority. *Int. Journal Information Security*, 8(2):137–152, 2009.

[Cra05] Jason Crampton. A reference monitor for workflow systems with constrained task execution. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 38–47, New York, NY, USA, 2005. ACM.

[CSBE08] Manuel Clavel, Viviane Silva, Christiano Braga, and Marina Egea. Model-driven security in practice: An industrial experience. In *ECMDA-FA '08: Proceedings of the 4th European conference on Model Driven Architecture*, pages 326–337, Berlin, Heidelberg, 2008. Springer-Verlag.

[CW87] David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. *Security and Privacy, IEEE Symposium on*, 0:184, 1987.

[Dav93] Thomas H. Davenport. *Process Innovation – Reengineering Work through Information Technology.* Harvard Business School Press, 1993.

[Eri08] Erik T. Mueller. Discrete Event Calculus Reasoner Documentation, March 2008. IBM Thomas J. Watson Research Center, P.O. Box 704 Yorktown Heights, NY 10598,USA.

[FFSF95] Michael A. Cusumano Fernando F. Suarez and Charles H. Fine. An empirical study of flexibility in manufacturing. In *Sloan Management Review Fall*, 1995.

[FIS02] Federal Information Security Management Act FISMA. The 2002 Federal Information Security Management Act (FISMA), May 2002.

[FPP+02] Eric Freudenthal, Tracy Pesin, Lawrence Port, Edward Keenan, and Vijay Karamcheti. drbac: Distributed role-based access control for dynamic

coalition environments. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 411, Washington, DC, USA, 2002. IEEE Computer Society.

[FSG+01] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[Gaa08] Khaled Gaaloul. Basic mapping of rights to tasks : Delegation and revocation in r4egov. Technical report, 6th Framework Programme, Information Society Technologies, R4eGov, June 2008.

[Gay05] Sébastien Gayral. Access control over Task Delegation in Workflow Management Systems. Matser thesis, Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzer, 2005.

[GC] Khaled Gaaloul and François Charoy. Task delegation based access control models for workflow systems. In *I3E 2009: Proceedings of Software Services for e-Business and e-Society, 9th IFIP WG 6.1 Conference on e-Business, e-Services and e-Society, Nancy, France, September 23-25, 2009.*, volume 305 of *IFIP*. Springer.

[GCSL07] Khaled Gaaloul, François Charoy, Andreas Schaad, and Hannah Lee. Collaboration for human-centric egovernment workflows. In *WISE '07: Proceedings of the 8th Web Information Systems Engineering 2007, Nancy, France*, Lecture Notes in Computer Science, pages 201–212. Springer, 2007.

[GHS95] Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: from process modeling to workflow automation infrastructure. *Distrib. Parallel Databases*, 3(2):119–153, 1995.

[GMC09] Khaled Gaaloul, Philip Miseldine, and François Charoy. Towards proactive policies supporting event-based task delegation. *SECURWARE '09: Proceedings of the 3rd International Conference on Emerging Security Information, Systems, and Technologies*, 0:99–104, 2009.

[GP00] William Golden and Philip Powell. Towards a definition of flexibility: in search of the holy grail? *Omega*, 28(4):373–384, August 2000.

[GSFC08] Khaled Gaaloul, Andreas Schaad, Ulrich Flegel, and François Charoy. A secure task delegation model for workflows. In *SECURWARE '08: Proceedings of the Second International Conference on Emerging Security Information, Systems and Technologies*, pages 10–15, Washington, DC, USA, 2008. IEEE Computer Society.

[GZCG] Khaled Gaaloul, Ehtesham Zahoor, François Charoy, and Claude Godart. Dynamic authorisation policies for event-based task delegation. In *Advanced Information Systems Engineering, 22nd International Conference, CAiSE 2010, Hammamet, Tunisia, June 7-9, 2010.*, pages 135–149.

[Han99] Hand de Neville. Bliksem 1.10 user manual, 1999.

[Har04] Shon Harris. *CISSP(R) All-in-One Exam Guide, Third Edition*. McGraw-Hill Osborne Media, 2004.

[Hav05] Michael Havey. *Essential Business Process Modeling*. O'Reilly Media, Inc., 2005.

[HJPPW01] Asa Hagstrom, Sushil Jajodia, Francesco Parisi-Presicce, and Duminda Wijesekera. Revocations-A Classification. In *CSFW '01: Proceedings of the 14th IEEE workshop on Computer Security Foundations*, page 44, Washington, DC, USA, 2001. IEEE Computer Society.

[HK03] Patrick C. K. Hung and Kamalakar Karlapalem. A secure workflow model. In *ACSW Frontiers '03: Proceedings of the Australasian information security workshop conference on ACSW frontiers*, pages 33–41. Australian Computer Society, Inc., 2003.

[IPS89] IPS, Information Processing Systems - Open System Interconnection - Basic Reference Model. Part 2: Security Architecture, 1989.

[ITG07] IT Governance Institute ITGI. *COBIT 4.1*. ISA, 2007.

[KLL09] Ryan K L Ko, Stephen S G Lee, and Eng Wah Lee. Business process management (bpm) standards: A survey. In *To appear on Business Process Management journal Vol. 15 No. 5, 2009. Emerald Group Publishing Limited. Accepted on 2 Dec 2008*, 2009.

[KS89] Robert Kowalski and Marek Sergot. A logic-based calculus of events. pages 23–51, 1989.

[Law97] Peter Lawrence, editor. *Workflow handbook 1997*. John Wiley & Sons, Inc., New York, NY, USA, 1997.

[LB73] Len Lapadula and D. Elliott Bell. Secure computer systems: A mathematical model. technical report mtr 2547 vol. 1, mitre corporation, bedford, massachusetts., 1973.

[Moe07] Robert Moeller. *Coso enterprise risk management: understanding the new integrated erm framework*. John Wiley & Sons, Inc., New York, NY, USA, 2007.

[Mue99] Michael Zur Muehlen. Resource modeling in workflow applications. In *Proceedings of the 1999 Workflow Management Conference (WFM99*, pages 137–153, 1999.

[Mue06] Erik T. Mueller. *Commonsense Reasoning*. Morgan Kaufmann Publishers Inc., CA, USA, 2006.

[NoC95] National Institute of Standards NIST and Technology Technology Administration U.S. Department of Commerce. An Introduction to Computer Security: The NIST Handbook, October 1995.

[OMG04] Object Management Group OMG. UML 2.0 Superstructure Specification", available at: "http://www.omg.org/cgibin/ doc?ptc/2004-10-02., 2004.

[OMG06] OMG, Object Management Group. Business Process Modeling Notation Specification, March 2006. `http://wwww.bpmn.org/`.

[OMG07] Object Management Group OMG. Business Process Modeling Notation (BPMN)", available at: "http://www.bpmn.org/", 2007.

[OvdRG98] Martin S. Olivier, Reind P. van de Riet, and Ehud Gudes. Specifying application-level security in workflow systems. In *DEXA '98: Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, pages 346–351, Washington, DC, USA, 1998. IEEE Computer Society.

[PP06] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing (4th Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006.

[R4e06] Technical Annex R4eGov. Towards e-administration in the large, March 2006. `http://www.r4egov.eu/`.

[RvdAHE05] Nick Russell, Wil M. P. van der Aalst, Arthur H. M. Hofstede, and David Edmond. Workflow resource patterns: Identification, representation and tool support. In *Proceedings of the Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal*, pages 216–232, 2005.

[RvdAHW06] Nick Russell, Wil M. P. van der Aalst, Arthur H. M. Hofstede, and Petia Wohed. On the suitability of uml 2.0 activity diagrams for business process modelling. In *APCCM '06: Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling*, pages 95–104, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[Sch] Andreas Schaad. An extended analysis of delegating obligations. In *Research Directions in Data and Applications Security XVIII, IFIP TC11/WG 11.3 Eighteenth Annual Conference on Data and Applications Security, July 25-28, 2004, Sitges, Catalonia, Spain*, pages 49–64. Kluwer.

[Sch92] August-Wilhelm Scheer. *Architecture of Integrated Information Systems: Foundations of Enterprise Modelling*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992.

[Sch03] Andreas Schaad. A Framework for Organisational Control Principles. Ph.D. thesis, The University of York, England, 2003.

[Sch07] Andreas Schaad. A framework for evidence lifecycle management. In *Web Information Systems Engineering, Proceedings of the WISE 2007 International Workshops, Nancy, France*, Lecture Notes in Computer Science, pages 191–200. Springer, 2007.

[SGN] Shazia Wasim Sadiq, Guido Governatori, and Kioumars Namiri. Modeling control objectives for business process compliance. In *Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, year = 2007, pages = 149-164, publisher = Springer, series = Lecture Notes in Computer Science.*

[SIP07] Mohammed Ashiqur Rahaman Sarath Indrakanti, Khaled Gaaloul and Henrik Plate. Prototype extended collaborative workflow tool. Technical report, 6th Framework Programme, Information Society Technologies, R4eGov, March 2007.

[SRS+05] L. Seitz, E. Rissanen, T. Sandholm, B. S. Firozabadi, and O. Mulmo. Policy administration control and delegation using xacml and delegent. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 49–54, Washington, DC, USA, 2005. IEEE Computer Society.

[SS93] Paul R. Smith and Richard Sarfaty. Creating a strategic plan for configuration management using Computer Aided Software Engineering (CASE) tool. Paper For 1993 National DOE/Contractors and Facilities CAD/CAE User's Group, 1993.

[STY07] Wei She, Bhavani Thuraisingham, and I-Ling Yen. Delegation-based security model for web services. *High-Assurance Systems Engineering, IEEE International Symposium on*, 0:82–91, 2007.

[TH99] Jui Chiew Tan and Patrick T. Harker. Designing workflow coordination: Centralized versus market-based mechanisms. *Information System Research*, 10(4):328–342, 1999.

[Tho97] Roshan K. Thomas. Team-based access control (tmac): a primitive for applying role-based access controls in collaborative environments. In *RBAC '97: Proceedings of the second ACM workshop on Role-based access control*, pages 13–19, New York, NY, USA, 1997. ACM.

[Tim05] Tim Moses. eXtensible Access Control Markup Language (XACML) Version 2.0, 2005. Committee specification, OASIS.

[Ton03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith,

Satish Thatte, Ivana Trickovic, Sanjiva Weerawarana. Business Process Execution Language for Web Services Version 1.1, 2003. IBM, Microsoft.

[TS98] Roshan K. Thomas and Ravi S. Sandhu. Task-based authorization controls (tbac): A family of models for active and enterprise-oriented autorization management. In *Proceedings of the IFIP TC11 WG11.3 Eleventh International Conference on Database Securty XI*, pages 166–181, London, UK, UK, 1998. Chapman & Hall, Ltd.

[Ven03] Karin Venter. A Model for the Dynamic Delegation of Authorisation Rights in a Secure Workflow Management System. Ph.D. thesis, Faculty of Science at the Rand Afrikaans University, Johannesburg, South Africa, May 2003.

[Wes07] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag, 2007.

[WFM99] WFMC, The Workflow Management Coalition. Workflow Management Coalition Terminology and Glossary, 1999. Document Number WFMC-TC-1011.

[WFM01] WFMC, The Workflow Management Coalition. Workflow Security Considerations, 2001. White Paper, Document Number WFMC-TC-1019.

[WFM05] WFMC, The Workflow Management Coalition. Process Definition Interface : XML Process Definition Language, 2005. `http://www.wfmc.org/`.

[WGHS99] Mathias Weske, Thomas Goesmann, Roland Holten, and Rüdiger Striemer. A reference model for workflow application development processes. *SIGSOFT Softw. Eng. Notes*, 24(2):1–10, 1999.

[WKB07] Jacques Wainer, Akhil Kumar, and Paulo Barthelmess. Dw-rbac: A formal security model of delegation and revocation in workflow systems. *Information System*, 32(3):365–384, 2007.

[XLfC] Li Zhang Xu Liao and Stephen Chi fai Chan. A task-oriented access control model for wfms. pages 168–177. Information Security Practice and Experience, First International Conference, ISPEC 2005, Singapore, April 11-14, 2005, Proceedings, Springer, 2005.

[ZAC03] Longhua Zhang, Gail-Joon Ahn, and Bei-Tseng Chu. A rule-based framework for role-based delegation and revocation. *ACM Trans. Information System Security*, 6(3):404–441, 2003.

[ZM04a] Michael Zur Muehlen. Organizational management in workflow applications – issues and perspectives. *Inf. Technol. and Management*, 5(3-4):271–291, 2004.

[ZM04b] Michael Zur Muehlen. *Workflow-based Process Controlling. Foundation, Design, and Application of workflow-driven Process Information Systems.* Logos Verlag Berlin, 2004.

[ZOS03] Xinwen Zhang, Sejong Oh, and Ravi Sandhu. PBDM: a flexible delegation model in RBAC. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 149–157, New York, NY, USA, 2003. ACM Press.

[ZPG10] Ehtesham Zahoor, Olivier Perrin, and Claude Godart. A declarative approach to timed-properties aware Web services composition, INRIA internal report 00455405, February 2010.