Imperial College of Science, Technology and Medicine
University of London

Department of Computing

A Virtual Open Organisational Platform

For the Open Coop

By
THOMAS J. SALFIELD

Supervisor: Professor Morris Sloman

Submitted in partial fulfilment of the requirements for the MSc Degree in Computing Science of the University of London and for the Diploma of Imperial College of Science, Technology and Medicine.

September 2005

Abstract

This paper investigates possible solutions to the problems of governing virtual organisations in the context of one particular organisation, the Open Coop. Taking a process oriented approach to analysing organisations, the paper documents the process of designing and implementing an Open Source platform which enables virtual organisations to use a constraint-based notation to specify and enact their governance processes, define sub-groups, and allocate roles.

Acknowledgements

I would like to extend my thanks to Professor Morris Sloman of the Department of Computing for supervising this project and helping me to evolve my ideas; and to Emile Lupu for his thought provoking comments.

Dr. Gary Alexander, Josef Davies-Coates, and Oliver Sylvester-Bradley from the Open Coop helped make this project possible and I would like to acknowledge Tav of Espia for his time spent on valuable technical discussions.

Finally I am indebted to my family for their unwavering support and to Sonia Ali for her brilliant ideas and lovely cooking.

Table of Contents

1 Introduction

1.1 Motivation for the Project

The Open Coop is a cooperative and a virtual organisation, and functions as a network of member-governed organisations. The author of this paper is a founding member of the Open Coop. The Open Coop seeks to create partnerships between its member organisations.

The author was surprised to find that despite the success and maturity of much Open Source software, and the fact that most Open Source projects are member-governed virtual organisations, there is little Open Source software available for the formation and governance of organisations in a virtual environment. In addition, most traditional cooperatives whilst member governed, do not have any method of governing their organisations in a dynamic and geographically distributed way.

This project aims to specify the requirements for forming and governing organisations in the virtual realm, and to implement a prototypical solution.

1.2.Overview

The project focuses on building a platform for forming and governing member-run organisations in a virtual environment. Taking a fundamentally process-oriented approach to analysing organisations, this paper outlines the process of design and implementation of a *Virtual Open Organisational Platform*.

One key difference between this, and other platforms for virtual organisation is that it is intended to be an *open access* system facilitating *formation* of organisations in the virtual environment. The platform will be web-based, allowing any *web-user* to create an identity and begin taking part. There must therefore be clear mechanisms for users to gain the organisational roles necessary to take part in various aspects of an organisation. There will need to be mechanisms for individuals or groups, to “bootstrap” a virtual organisation, giving it clearly defined decision-making, and role allocation, mechanisms.

1.3 Report Structure

The structure of the remainder of this paper is outlined below.

Chapter 2 briefly describes the Open Coop as well as some key background concepts.

Chapter 3 reviews relevant literature regarding the study of virtual organisations as well as the study of processes in organisational theory.

In **Chapter 4** two use-case scenarios are considered, and some key features are distilled into a functional specification.

In **Chapter 5** the overall architecture of the system is described.

In **Chapter 6** some key aspects of the implementation are highlighted.

Chapter 7 explains the workflow specification notation which is used to represent types of organisational process in the system.

In **Chapter 8** some initial testing is undertaken and the implementation is evaluated.

Chapter 9 suggests possible future development directions for the implementation, as well as further research which could be undertaken.

2 Background

In this section the context of the Open Coop, and relevant background concepts are discussed.

2.1 The Open Coop

The Open Coop is a collaborative network of 'social enterprises', 'social entrepreneurs', and cooperatives. By creating greater collaboration amongst many such organisations and individuals, the Open Coop plans to increase efficiency of all member groups through effective partnership working. The fundamental strategy is for these small groups to pool resources in those areas where there are economies of scale, whilst maintaining autonomy in other areas. So for instance groups might share office space, accounting functions, or marketing campaigns wherever this is mutually beneficial. Such partnerships need to be able to effectively make collective decisions in order to share resources and collectively develop projects. Facilitating flexible collaborative creation and governance of partnerships developed by the Open Coop, is the core goal of this project.

The Open Coop[1] is in many senses a *virtual organisation.* It is a partnership between many autonomous organisations. Existing member organisations are predominantly UK based due to an initial UK membership drive. However future membership drives are expected to be more international in scope. Member organisations are spread around the country and therefore require computer supported methods of organisation. Further, the coordinating group of members is itself made up of a partnership of individuals and organisations. The Open Coop does not have any physical office.

[1] The Open Coop is legally constituted as a Limited Liability Partnership which is a new organisatio#nal form which allows partnerships to have flexibility in governance structures whilst retaining limited liability. LLPs were created by the Limited Liability Partnerships (LLP) Act 2000.

2.2 Collaborative Technologies

It is recognized by the Open Coop that to achieve flexible collaborative creation and governance of partnerships, will require a cutting-edge suite of organisational technologies, as well as facilitation techniques and technology training. First steps have been taken in the Open Coop's Corporate Plan. The Open Coop has recruited a number of member groups and has regular open meetings focused on building partnerships with specific purposes.

There is some basic collaboration technology in place at the Open Coop. This takes the form of a wiki[2]-based content management system, which functions as a basic but effective platform to organise the company and involve participants in building the knowledge base. Further there is an IRC (Internet Relay Chat) channel used which takes advantage of the latest in IRC bot technology[3].   While these technologies have been useful as a first-stage portal, the Open Coop requires a number of more targeted technologies in order to effectively progress towards its goals.

In particular the application development goals are:

- a system for governing the Open Coop's various initiatives in a distributed and geographically dispersed manner;
- a project management system with integrated “smart contracts“ [1]
- an ethical market place with trust and reputation mechanisms built-in.

[2] According to Ward Cunningham the inventor of the wiki, a wiki is “*The simplest online database that could possibly work.*” http://wiki.org/wiki.cgi?WhatIsWiki

[3] An IRC bot is a set of scripts or an independent program that performs special functions on Internet Relay Chat. In particular an the Open Coop they are used for logging the channel, collecting statistics, leaving messages for other users, looking up words and etymology, finding the latest exchange rates, and being rude and offensive in a funny way. http://en.wikipedia.org/wiki/IRC_bot

Since the first of these has been identified as the most urgent target by the Open Coop members, this project will focus on - *a distributed organisational governance system.*

This organisational governance system will aim to be an effective system for the Open Coop and therefore should aim specifically to facilitate the types of organisational governance relevant to cooperatives, social enterprises and partnerships between such organisations. Since the Open Coop is a virtual organisation, it is also hoped that this project will contribute more broadly to solving the issues of distributed governance in virtual organisations.

Aptly named, the Open Coop is both an open organisation and a cooperative. Various aspects of the design of the organisational governance system will be informed by these characteristics of the Open Coop, therefore both concepts are briefly reviewed below.

2.3 Open Organisation

Open organisations are a new phenomenon, and are probably best defined as follows:

“An open organisation is an organisation open to anyone who agrees to abide by its purpose and principles, with complete transparency and clearly defined decision making structures, ownership patterns, and exchange mechanisms; designed, defined, and refined, by all members as part of a continual transformative process.” [2]

The key point is that open organisations are defined by their transparency, which includes transparency of the mechanism by which one can join the organisation and assume various roles, and transparency in the ways decisions have been made and will be made in the future.

In the body of work relating to open organisations [3] there is a focus on designing, and agreeing on, explicit *processes*[4] which might otherwise take the form of unstructured social interactions between members. It is argued that, in order for a group to be transparently and purposefully organised, the interactions which occur within the group must be formally structured in the form of processes.

The design of this project was informed greatly by the small but well organised body of work relating to open organisations.[3]

[4] A full discussion of what is meant by processes is included later in the paper.

2.4 The Cooperative sector

As well as being a cooperative venture, the Open Coop aims to serve the cooperative sector and other member-run initiatives such as community groups and Limited Liability Partnerships. According to the International Cooperative Alliance:

“A co-operative is an autonomous association of persons united voluntarily to meet their common economic, social, and cultural needs and aspirations through a jointly-owned and democratically-controlled Enterprise.” [4]

This emphasizes the need for democratically control organisations which are considered cooperative. It follows that in order to have a cooperative virtual organisation we must have a way of governing the organisation in a democratic and geographically distributed manner. Since in our analysis organisations consist of participants interacting through processes, the act of governing the organisation is the act of collectively choosing processes and the roles in those processes. As a result the system which I design and implement in this project is intended to (in the context of a virtual organisation) :

facilitate the collective choice of processes, and roles in those processes, in a democratic and geographically distributed manner.

“Co-operatives are equitable businesses with a social purpose, democratically owned and controlled by their members. There are nearly three-quarters of a million co-operatives worldwide, providing jobs for over 100 million people - more than are employed by all the multinational corporations in the world.” [4]

This statement hints at the potential commercial power of all existing cooperatives, if properly connected and coordinated. The Open Coop's vision is to create a system in which all these organisations and similar membership-based organisations can easily create collaborative initiatives which are mutually beneficial. The desired result is, increased efficiency of their operation and cohesiveness of the cooperative sector[5] .

The potentially very large amount of users of a ***partnership management system*** for the cooperative sector has the important implication that the application should developed in this project should be highly scalable. At the current scale, with the Open Coop as a start-up, a centralised web-server will be sufficient. However since in the future it is likely that this system will have very high volume and traffic, it is essential that the system should be able to work over multiple servers, ideally using some kind of distributed objects system. More generally, all efforts should be made to make the system easy to extend into a scalable system. In addition since this application is likely to be the first in a suite of collaboration applications for the Open Coop, it is important to use a framework into which future applications could be easily integrated.

[5] The “cooperative sector” is used loosely here, to refer to member-governed initiatives that the Open Coop aims to served.

3 Literature Review

3.1 Virtual Organisations

While the Open Coop is a profit-making membership organisation, there are many areas of academic work which have influenced the Open Coop's strategy and philosophy. Areas include, cybernetics, systems theory, the viable systems model and monetary theory. However most relevant to this project is the body of work surrounding virtual organisations. This project aims to build a platform which is applicable to a specific type of virtual organisation – those which are democratically governed, open organisations. Further the focus of this project will be the governance[6] processes which occur within such v*irtual open organisations*.

Over recent years interest in “virtual organisations” has increased in many fields of academia from sociology to computer science. However the development of a generally accepted theoretical basis of virtual organisations has been elusive. There is not even full consensus on the definition of virtual organisations and various surrounding concepts. Perhaps the most complete effort at reaching consensus on the fundamental concepts of virtual organisation comes from the EU TrustCom project. [5] This project offers a summary of many definitions and key concepts relating to virtual organisations.

One definition which stands out as best encompassing the others is:

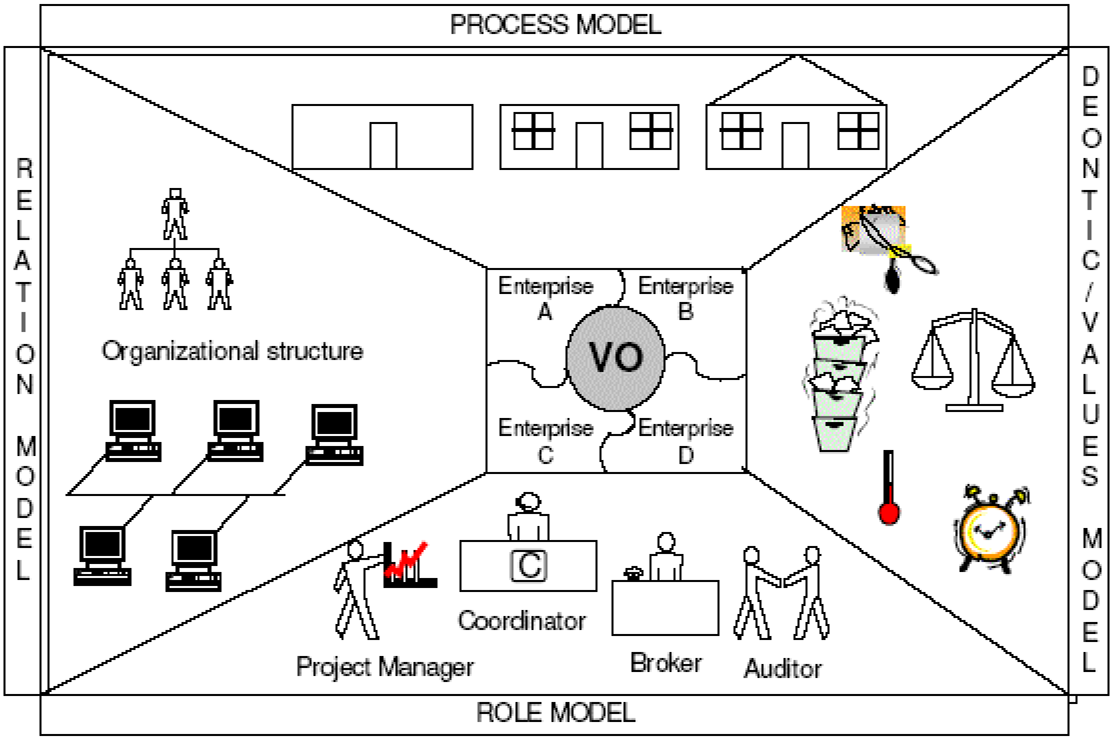
“A Virtual Organisation is a combination of various parties (persons and/or organisations) located over a wide geographical area which are committed to achieving a collective goal by pooling their core competencies and resources. The partners in a Virtual Organisation enjoy equal status and are dependent upon electronic connections (ICT infrastructure) for the co-ordination of their activities.” [6]

[6] Organisational Governance is defined in section 2.6

This project is also informed by earlier work on virtual organisations. Handy stresses the importance of trust in virtual organisations, contrasting virtual organisations to traditional organisations in which monitoring and auditing tend to replace the need for trust.[7] Handy argues powerfully that in a virtual organisation since day-to-day monitoring is not possible there is a need to establish more enduring forms of trust. However, paradoxically it is also harder to establish trust in virtual organisations due to the fact that people may never meet in person, and they may not be part of the same legal organisation. Therefore it is clear that a system for governing virtual organisations should focus on the issue of establishing trusting relationships. One way this can be achieved is through transparency of data relating to past interactions of a user in the system. This, is also an essential aspect of open organisations, and allows each actor in the system to assess the trustworthiness of potential collaborators based on past performance. This has the important implication that:

All data in the system should be transparent to all users, further a transaction system should be maintained in which each interaction between a member and an organisation is recorded.

As part of the THINKCreative project, Camarinha-Matos and Abreu have suggested four complementary perspectives from which we can model virtual organisations, as shown in the diagram below. [8]



<u>Relations models</u> describes the forms of interrelationship that can occur between components within a network. For instance, the following types of relationships can be identified: control relationships (which identify the structure of authority), dependence relationships (which identify topological dependence), ownership relationships (which define the boundaries of each agent).

<u>Role models</u> that describe all roles and their positioning within the network structure. A role model implicitly defines a topology of interactions.

<u>Process models</u> that focus on dynamic courses of events. Some generic concepts such as activity and actor, time dependencies such as equal, during, starts, finishes, and resource-related perspectives such as necessary, sufficient, have to exist.

Deontic / Values Models that define constraints for all agents within a network at different levels, such as:

- **economic level -** where one may place constraints about cost, utility, etc,
- **organizational level -** where behaviour constraints relating to one's organisational role are placed,
- **operational level -** where one may place constraints about the order and interdependencies of actions that need to be performed in order to meet an objective.
- **computational level,** where constraints about resources sharing, service and communication interoperability are placed.

Fig 3.1 Virtual Organisation Modelling Perspectives - *Extract from TrustCoM* [5]

This project attempts to design and implement a system which facilitates the flexible formation of groups, specification of processes, assignment of roles, and enactment of processes. The act of specifying a process, assigning roles, or creating groups, corresponds closely to the act of modeling a virtual organisation. Clearly an *operational platform* will also require the additional functionality to *enact* the specified (modelled) processes, and for the groups and individuals to be able to dynamically change their relationships to each other.

Thinking about the implementation from each of these four perspectives, will help us evaluate its flexibility in terms of the kinds of organisational situations which can be specified and realised.

3.2 The problems of governing a virtual organisation

Crowston and Short [9] follow Mohr [10] in arguing that since organisations are too multi-dimensional to make generalisations about, processes are a useful sub-unit for analysing and reasoning about organisations.

Organisational governance is the subject surrounding the set of mechanisms which govern an organisation. This should be distinguished from management, which is the subject of temporarily assigned roles and responsibilities in an organisation. A management mechanism can always be changed, removed, or replaced by governance mechanisms. In this sense management mechanisms are subordinate to governance mechanisms which provide the highest level of authority in any organisation. Since every decision cannot be taken through the governance mechanism, it is necessary to delegate roles and responsibilities.

Conventionally, governance is prescribed by the legal form of the organisation. For example, a limited company has a set of shareholders who meet at an AGM and appoint the board of directors, as well as top management. A majority of shareholders always has the right to change the management or veto actions. In a cooperative or a partnership this is somewhat different. For instance, in cooperatives there is democratic governance by the members. Of course such democratic governance can take various of forms, so in the case of a cooperative, there is a need for explicit decisions to be made about the fundamentals of governance.

Virtual organisations often take the form of partnerships, whether or not they are legally constituted as such. Managing partnerships successfully is well known to be a minefield. Part of the problem is establishing *who* has the authority to make decisions. This can lead to disputes, and disjuncture in the partnership. By properly establishing governance mechanisms before embarking on a partnership such problems can be mitigated. However specifying governance mechanisms is easier said than done. The problems are twofold. Firstly, most people have very little understanding of organisational governance mechanisms, and there is little guidance available. Secondly, in virtual organisations, there are few tools which allow the flexible specification of governance mechanisms. Many virtual organisations use technologies which do not facilitate sufficiently structured communication. For instance, many Open Source projects, use IRC and wikis as their main mode of communication (including the Open Coop). Since organisations are most usefully understood through analysis of processes, and governance is no exception, such unstructured forms of communication are insufficient. As a result governance mechanisms often become hidden, and informal, implicit in the social structure.

This project aims to create a system in which:

virtual organisations can flexibly and explicitly define and enact their governance processes in a structured way.

3.3 The problem of establishing trust in virtual organisations

Often, custom-built platforms for virtual organisation are intended for facilitation of already established organisational structures. In the case of the Open Coop it is expected that organisations will be created through a *virtual* platform. For this reason users will not have *a priori* knowledge of each other's identities. Establishing the trust necessary to create and dynamically govern organisations (or ad-hoc groups) *virtually*, creates a special set of problems regarding authenticating users, allocating roles for accessing services and resources, and setting permissions for participation in the organisation.

In order to deal with this problem, Keoh, Lupu and Sloman [11] suggest using *user-role assignment policies* which are rules for acquiring the roles. In the context of this project, this will be achieved by defining the organisational structure in terms of roles which are assigned to groups. Then in order to specify policies for assigning roles to members of the organisations, we will need to define criteria for joining and leaving the group. Since in our approach we will be focusing on processes, the *user-role assignment policies* will take the form of processes for joining and leaving the group. Once a user is a member of a group she will acquire all roles associated with that group.

By being transparent and explicit about group admission processes we build trust in the organisation, because behaviour is expected to be consistent with the admission criteria.

3.4 Modelling Processes – levels of decomposition

In the field of business process modelling, and business process re-engineering, processes are considered to consist of *activities*.[12] However, often processes contain so many activities, that the activities are usefully grouped into nested sub-processes. Varying approaches have been suggested for defining such process hierarchies. [12],[13].   For instance, Davenport and Short [13] group activities into process types on the basis of the entities or sub-units involved, the types of objects manipulated, and the type of work activities taking place.

However, having many nested processes can become too complicated for the task at hand, and the question arises, to what level should we specify the detail of processes. Put differently, what is the level of atomicity of an *activity*. As Crowston and Short [9] point out, the appropriate level at which we should specify the process is usually determined by the purpose of the analysis. We should specify process types to the level of detail where the salient differences (according to the purpose of the analysis) between such types are expressible. This is the principle of *generativity* of a specification or model, as explained below.

Crowston and Short [9] argue that rather than grouping activities into processes, we should group activities into higher-levels of activities. For instance, most workflow applications specify processes to the level of detail of individual *user-tasks*. Since there are likely to be literally millions of possible activities at this level, there is little possibility for standardisation of reusable components representing higher-level activities [14].

Crowston and Short [9] suggest that conciseness and generativity are two key properties necessary for a process model or specification. The benefits of conciseness are clear. Generativity is the property which holds when the representation of a type of process not only specifies the current type of process but also suggests possible alternative types of processes which could be created. This is an important property for allowing experimentation, comparison, and improvement of processes.

So, for instance, if we were deciding which governance process best suits a purpose and a part of the governance process included voting, we would not be interested in the order in which votes are cast, which specific users get to vote, or even how the action of voting occurs at all, we will be interested in the voting algorithm itself.

An example voting algorithm might look like this:

“people in groups X, Y, and Z can vote “yes” or “no”, and once more than 60% of people have voted, if more than 50% voted “yes” then we will implement the policy, otherwise the policy will be revised, and we will restart the process.”

Similarly in a debate *activity,* we may want to specify different types of debate such as single threaded, or multi-threaded, synchronous or asynchronous, but not the detail of how each individual process will be threaded. This lower level would be more appropriate to, for instance, a psychological study of how people react to one another's comments, but is not relevant when considering the goals of processes at an organisational level.

Shi, Lee and Kuruku [14] have argued that business process automation, has succeeded through the practice of re-engineering business processes and standardising them to fit “best practice”. This has resulted in models that lack flexibility and applicability for non-standardised industries. In addition there have been found to be major barriers to organisational change, causing great cost where the “best practice” processes differ greatly from the existing processes. This includes resistance from employees who are reluctant to change the existing processes in their organisations. Shi et al, [14] follow Zhuge [15] in suggesting developing a “component-based workflow” system. A *workflow component* is defined as a complete *business process unit* and is analagous to *higher-level activities.*

Most workflow solutions have tended to distribute processes to the level of individual user-tasks. Since there are so many possible variations at this level there is little possibility of useful standardisation.

3.5 Processes and Activities as Means of Achieving Goals

Processes can be conceptualized as ways to accomplish *goals*. Indeed, processes are often named after the goals they accomplish. For instance, in the context of governance, we might have a process to “create a subgroup”, “join a group”, “make a policy decision”. Such goals can be considered subsets of states which satisfy a condition [16]. Goals can therefore be effectively represented as *constraints* [16],[17], [18]. This is useful for comparing the results of different types of processes. For instance we may want to compare the types of *policy decisions* that get made when different *voting* algorithms are used. The *goal* also provides a heuristic for assessing which component activities are likely to be relevant.

It is also useful to express *higher-level activities* as goals and represent those goals as constraints. For instance the “Vote” component might have a constraint that the percentage of people who have voted should be greater than 50%. Once such constraints are fulfilled, the activity (goal) is complete, and the transition to the next activity can take place

4 Use-case Scenarios and Specification

The goal of this project is to develop an application, in which individuals or organisations can form and govern virtual organisations or virtual teams. The application will also contain a framework within which virtual organisations can flexibly specify and enact their governance processes.

The application will be known as the *Virtual Open Organisational Platform* or VOOP.

The Open Coop aims to use VOOP in order to map existing and prospective member organisations into a system of interrelated groups. In such interrelated groups, all participants can *see* the relations between the various groups, the members of the groups, the governance processes of the groups which are in progress, and those governance processes which are completed and no longer active. In this framework organisations will be "open organisations" in as much as they will have:

1. Transparency in the roles of participants
2. Clearly defined, and well documented, governance processes
3. Clearly defined rules surrounding participation in various sub-groups.

Such organisations or teams will be represented in VOOP as 'groups'. The interactions involved in governing the group will take the form of a set of processes.

Group creation and development is expected to reflect either the structure of an existing organisation, or a new organisation coming into existence in the virtual realm.

Since organisations are usefully analysed in terms of their processes [9][10], VOOP will facilitate the creation of, and participation in, governance processes. In particular the project will focus on the flexible specification and enactment of governance processes.

The following are use-case scenarios identified by the Open Coop which VOOP should enable. The first is a hypothetical use-case whilst the second is a use-case of an Open Coop project which is currently being developed by Dr Gary Alexander, Senior Lecturer in ICT at the Open University[7] .

4.1 A Hypothetical Open Source Project

Much research exists concerning the social structure of Free Software and Open Source projects [19] as well as Open Source projects as virtual organisations. Crowston and Howison [19] follow many authors - Cox[20], Gacek et al. [21], Mockus et al. [22], in characterising FLOSS[8] projects as an "onion ring" diagram (as depicted in Figure 4.1). Moving from the centre of the diagram to the outer layers there are likely to be more participants and a lower level of involvement. At the centre are the few core developers who contribute the majority of the code and make the decisions about the evolution of the project. In the next layer come the co-developers who tend to contribute bug-fixes and make minor alterations to the code. Then come the active users who contribute bug reports and feature requests and can be quite large in number for some projects.

[7] http://sustainability.open.ac.uk/gary/

[8] FLOSS – Free Libre and Open Source Software

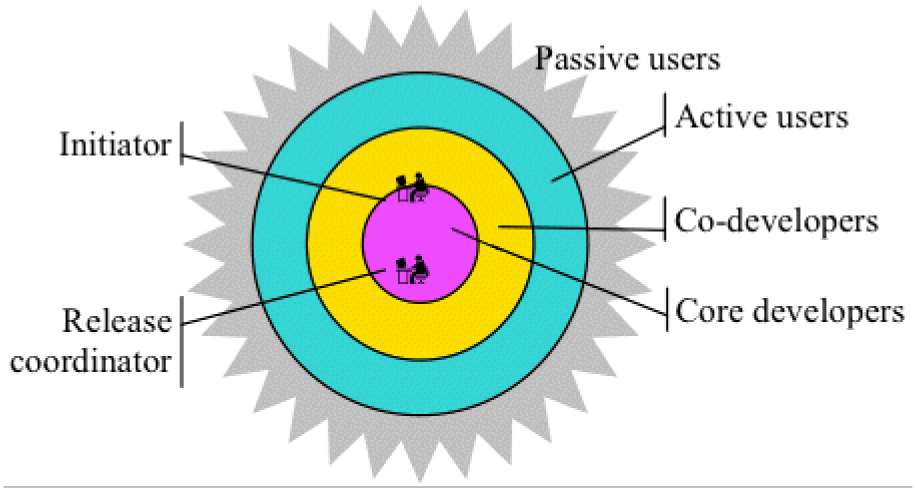


Figure 4.1: A synthesised FLOSS development team structure (*extract from Crowiston and Howison* [19])

For a hypothetical example of a FLOSS project, with formally defined governance procedures, corresponding to the structure outlined above, the process of mapping the project into VOOP is outlined below. It should be stressed that many FLOSS projects do *not* currently have a formal governance structure. However according to the principles of open organisation, these projects are *not* open organisations since they do not have a written charter. It might be the case that were such an organisation to formalise its existing social structures, in order to become an open organisation, that processes similar to those specified below would result.

Formalising organisational processes should help organisations to ensure their organisational transparency. This is a key factor in simplifying participation in a core organising group for newcomers. Further, it should ensure that processes occur in a way which corresponds to the intentions of participants. Since unstructured interactions don't necessarily lead to the processes which were envisaged by participants, organisational governance/management is the activity of choosing processes [3].

4.1.1 Defining Organisational Processes

A core developer could set up a group on VOOP. She would first define a process relating to the high-level governance of the organisation. This might be considered the ultimate source of authority in the group/organisation. An example would be:

"*policy proposals are debated for a week after which there is vote requiring the support of a majority of core developers. Also policy decision can be reversed with the agreement of more than 70% of the core developers.*"

Once a high level governance process like this is defined, the group is able to propose and approve *lower level* governance and management processes. At this stage the core developer creating the group might move on to the next stage of group creation, creating sub-groups and roles in the organisation. Alternatively they might choose to define some lower level processes which are already well established in the organisation. So for instance she might now define the following processes:

> i) Proposals for *developing a new branch of code*. This might look like this:
>
> "*debate will continue for 1 day, then the proposal will be rated between +2 to -2 within the next 2 days. If the average rating is greater than 1 then the project will be taken to the implementation phase in which tasks, milestones, and dependencies will be mapped out and the developers will be assigned/assign themselves to tasks. After 3 months the implementation will be reviewed*".

ii) Another process might be "bugtracking". Here the process could be defined as:

"*any* "*active user*" *can submit a bug document, then a core or co-developer attends to it, and a core developer can archive it once it is fixed.*"

iii) In addition creating a feature request process would be a useful way of interfacing with the potential market for modifications to FLOSS software.

4.1.2 Creating Groups and Associated Roles in Processes

The group creator would then create three sub-groups with roles relative to the governance processes.

First, the Core Developer role would be the only role/group with permissions to take part in high-level governance process in any way. Second, they might be the only ones to be able to *rate* lower-level proposal for *developing a new branch of code*.
Additionally more specific roles such as "release coordinator" might be defined for individual members of the core group.

The co-developer role would have the permissions to contribute to processes relating to *developing a new branch of code*. However they would not be able to vote on proposals in this category.

Active users would be able to contribute to discussions and submit bug reports and feature requests as part of a code branch. If they so choose they should be able to attach bounties to feature requests and make them open for donations/bounties from others (this part will be beyond the scope of the project).

The group creator should send invitations to and/or create accounts for each of the core developers, co-developers and active users, to join VOOP. Each developer should then map themselves into the system by creating a personal profile with various attributes which should be configurable by group or by role.

4.1.3 Processes of Entry and Exit to the groups

- **Entry and Exit to the Core Developers Group**
  Once a Core Developers group is set up, entry to the group could require the approval of a certain percentage of the existing core developers. Other criteria of joining the group could be expressed informally, and implemented socially within the system. For instance if the group requires that to be a core developer an individual must have contributed more than 10,000 lines of code, then it will be the responsibility of each core developer not to approve an individual who has contributed less than this. It is normal in Open Source projects that since there are no contracts involved, developers can exit the project at will.

- **Entry and Exit to the Co-Developers Group**
  In order to become a co-developer an individual must get the approval of at least one existing co-developer or core developer.

- **Entry to the Active Users Group**
  Active users, should be able to register themselves without any approval, They should be able to create process instances under the classifications of, feature proposals, or bug reports.

- **Benefits of using VOOP**
  FLOSS projects have a tendency to have unclear and informal mechanisms of governance and decision making. By mapping their members and processes into VOOP they will formalize their governance mechanisms, and thereby increase their transparency, making it clear how to get involved and how decisions have been and will be made.

4.2 The Open Food Co-op[9]

Open Food Co-op, is an initiative of the Open Coop and is in the early stages of development. This is likely to be one of the first projects to pilot VOOP. The idea is to create a collaborative partnership between local organic food producers, shops/agents, distributors, and consumers.

4.2.1 Defining Organisational Processes

Once a group is set up on VOOP the high-level policy process will have to be defined. In this case it is expected that a Co-ordinating group will make policy for the whole group on a consensus basis. The process will look like this:

i) An issue is raised
ii) Coordinating group has a discussion.
ii) Coordinating group produces a proposal for a process which is approved by consensus (vote with single member veto).
iii) Proposal put to all members of the Open Food Coop for comment and approval by majority vote.
iv) If the proposal is approved by the Open Food Coop, it is implemented. Otherwise the Coordinating group reviews and revises the proposal, and

[9] Open Food Co-op - http://open.coop/tiki-index.php?page=Open+Food+Co-ops

either presents a new version or rejects the proposal.

The group creator could then go on to define further organisational processes before creating the group itself. Alternatively further processes could be defined from within the group using the above policy process. For instance a process may be defined corresponding to the process of getting food from producer to the consumer. This might look like this :

i) Food producers enter their produce for the next month or week into online forms.
ii) Consumers view the listings of food offered, select what they want, and select the shop/depot to which it is to be delivered. (In many cases, the consumers agents will do this on behalf of the consumers.)
iii) A logistics module lists the origins and destinations of produce needing to be delivered.
iv) The distributors view the listings and indicate which routes they will be making.
v) The distributors then transfer produce from the producer to the agents/shop.
vi) The produce and service is rated by the consumer.

N.B Its not expected that the necessary components to realise the logistics aspects of the process will be implemented as part of this project.

4.2.2 Creating Groups and Associated Roles in Processes

Five sub-groups with roles relative to the processes will need to be defined. Each member may be a member of more than one sub-group and thereby have more than one role.

i) Coordinating group
The coordinating group plays the main role in high level policy. That is they respond to issues raised by proposing policies. They also actively recruit for sub-groups such as producers, agents and distributor.

ii) Producer group
This group includes farmers, gardeners, and food processors (i.e. cheese makers, cooks) They are able to vote on policies proposed by the Coordinating group. They are able to put up produce on the consumer-producer process. Like most other groups they would elect a representative onto the coordinating group in order to represent their interests.

There might be different sub-groups within the producer group having slightly different roles. For instance it is anticipated that there will be both a farmers' group and a gardeners' group.

The producer group might also have an internal governance process aimed at optimising collective production to best match consumer demand.

iii) Distribution group
This group will include farmers and shops, and others who already do deliveries, to which will be added people willing to take on regular and irregular deliveries. The main function of this group will be to rationalise and co-ordinate the desired deliveries. They carry out stage v) delivering

the produce from the farmer to the depots/shops. They will also elect a representative to the coordinating group.

iv) Agents

Agents act on behalf of consumers or producers who do not wish to participate in the computer network, entering transactions on their behalf. They will thus fulfill the function of actively increasing the consumer base (marketing). Agents may choose to work together to market produce under a collective brand. This would entail having an internal governance process, as well as defining various processes around the creation of brand.

v) Consumers

Receive info on what is available/coming soon from producers and events organisers. Many consumers may not interact with the Food Coop online but rather buy from an agent in the real world. Feedback from consumer will be actively pursued by the Food Coop, in order to feed into the Coordinating group's work. This would entail a process in which consumers report their experience and any major trends would be reported to the coordinating group. Consumers will also elect a representative to the Coordinating group.

4.2.3 Processes of Entry and Exit to the groups

- **Entry and Exit to the Coordinating Group**

  The Coordinating group will consist of one representative from each of the 5 sub-groups. These representatives will be elected by each subgroup, through a process of continual debate, and voting, which will occur every 6 months. There will be 1 week of debate by the end of which sub-group members will be expected to have voted.

- **Entry and Exit to the producer and distribution groups**
  This will require approval by the coordinating group. Approval will be determined by the explicit approval of at least one member of the Coordinating group but can be vetoed by any member of the coordinating group.

- **Entry and Exit to the producer and agents/shops groups**
  Anyone can choose to join this group, but those found to be misrepresenting the Food Coop or flaunting rules can be banned by the Coordinating group.

- **Entry and Exit to the consumer group**
  Anyone can join this group, however agents should actively seek to involve consumers with relations to them, in order to gain a visible online reputation.

4.3 VOOP Specification : System requirements

Here I will layout a functional specification for the implementation of the application. The features here are a distillation and to some extent a rationalisation of the requirements laid out in the use-case scenarios, as well as a few more general points from the Background and Context section.

4.3.1 Groups

G1 Hierarchical group structure

Organisations or teams will be represented as groups. Where an organisation consists of a number of teams or departments, the group representing the organisation will have a number of sub-groups. Therefore groups should be structured hierarchically.

So, for example, within the Open Coop group there might be a Open Food Coop group, and a Software Development group. Within the Open Food Coop group there would again be multiple groups, such as the Producers, Consumers, and Distributors etc.

System requirement: **Hierarchically nested groups**

G2 A Seed Group

To reflect the fact that the Open Coop is a membership organisation in which many groups operate, an initial *seed* group will have to be created from which users will then be able to navigate to and join the various groups. The seed group is also the place where organisations new to the Open Coop can create a group to represent themselves. In the deployment of the software for the Open

Coop, this seed group will represent the Open Coop itself.

Since the system is intended to be an open-access system for creating virtual teams and organisations, any web-user will be able to create a group within the seed group. In fact its expected that the seed groups functionality will be restricted to having members join and create groups. That is, there will be no process for creating new process types in the seed group. This is intended to allow the seed group to be initially a place of experimentation where users can test the effects of different governance processes on the evolution of groups.

It is important to realise that the above configuration should be only one possible deployment of the software. For instance, the permission to create a new group in the *seed group*, could be allocated to a different group of *trusted users* with relatively stringent entry criteria (e.g. approval of existing members). Indeed its likely that in the virtual organisations which exist below the *seed group* that sub-group creation would be restricted.

System requirement: **a seed group should be created from which all other groups are created and navigated to.**

G3 Creating groups

When creating a new group, a user will first be expected to set the *purpose* and *principles* of the group. These are simple text entries which give potential members or collaborators useful information about the group.

The user will also be expected to define some key *types of processes,* as well as assigning roles within them. These are the types of processes which the group will have on creation. It is important to realise that once the group is created, new types of processes can only be created through existing processes. The system requirements for process definitions and roles are discussed in detail later.

In order to have full functionality, groups should normally be created with at least a minimal set of types of processes which allow:

1. creation of new types of processes,
2. joining or leaving the group,
3. creating a subgroup within the group.

It is perfectly possible that a group would not have all three of these processes on creation. However since they are normally expected, the user should be prompted to create each of these types whilst creating a new group.

System requirement: **Flexible creation of types of processes when creating a group. Prompt the user to create core process types, as well as *purpose* and *principles.***

G4 Group functionality

Once a group is created, the group is the place from which new processes are initiated. Most processes will in some way modify some aspect of the group itself. For instance, when the process of a new user joining the group is completed, the user will be added to the group's user list. The user should also be able to browse from group to group, and browse the users of a group.

System requirement: **The user should be able to initiate processes from within the group (assuming the user has the necessary roles), browse to other groups, or browse the users of the group.**

4.3.2 The User

U1 Identity Creation & Login

Since the Open Coop is an open organisation any web visitor will be able to create a secure identity and begin using VOOP. Once an identity is created the user will be able to login using that identity, at this stage the user will not be a member of any groups, but will be able to browse the contents of groups.

System Requirement: **Creation of secure identities.**

U2 User profiles

The user must have a set of *user-attributes* which constitute a profile. A *user-attribute* will be a label associated with a typed value, for instance the label might be “age” and the value might be an integer of value 10. It should be possible for any individual to change their own *user-attributes*. Users should also be able to view each other's profiles.

System Requirement: **User-editable profiles.**

U3 Joining Groups and Acquiring Roles

Assuming a user has the roles which are necessary to apply to join a particular group, on requesting to join the group the user will have to go through the associated process. It is likely that this process will require the user to submit some details which are specific to the group. Once a user is a member of a group, the user will acquire the roles which have been allocated to the group.

System Requirement: **Users acquire roles from their groups**

4.3.3 Processes Types

Processes are the key mechanism for *change* in the system. The process types associated with a group, define the ways in which groups can be changed. By assigning roles in process types to other groups, we define the ways in which groups can interact.

In order to demonstrate the effectiveness of the underlying system, it will be necessary to develop a minimal set of components which can be composed into process types. Below I have grouped the elements by the process types they are intended for creating. There are also some extra components which might be used in many possible types of processes.

Creating new types of processes

In order that the governance of groups can evolve its necessary to have a process type which can be used to create *new* process types. P1 to P3 are the components its envisaged which will be required.

P1 Process Type Definition

A flexible way of formally specifying governance processes is essential to this project, since it is envisaged that many “organisational forms” will map themselves into the Open Coop system. The Process Type Definition component will allow the user to compose a process type definition. The process type definition specifies the order of components in the process and the conditions (constraints) on which a component activity is considered complete.

System Requirement: ***component for flexibly defining the order of components in a new process type***

A process needs to be able to execute different *branches* of components depending on whether a certain precondition is met at a certain point in the process.

For instance in a policy-making process, a *vote* component might be an activity which forms part of a “make a policy decision” process which looks like this:

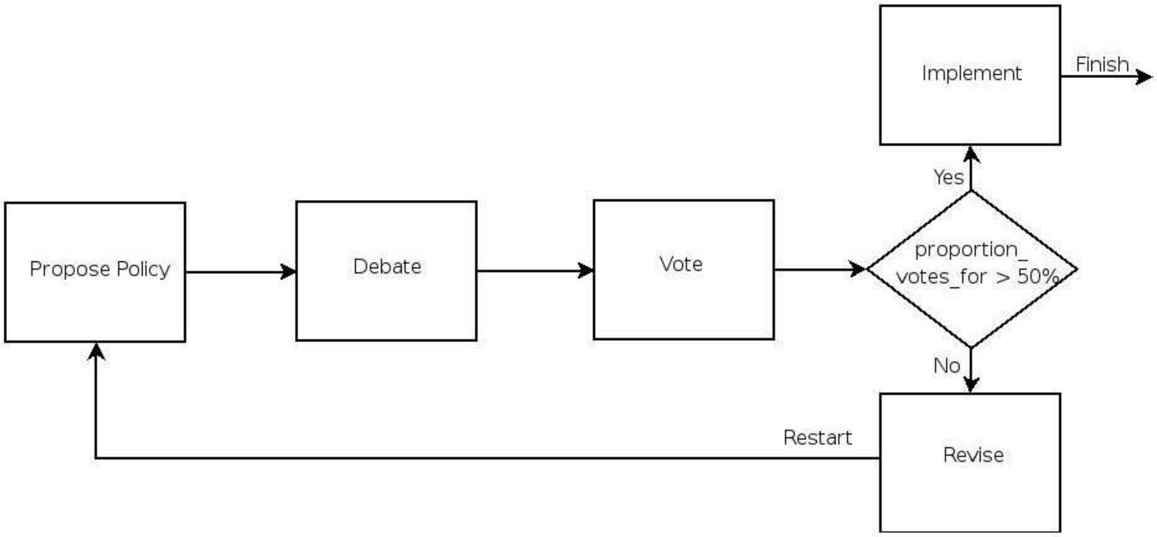


Fig 4.2 Branching in a Workflow on a Precondition

Depending on whether the precondition ("proportion_votes_for > 50%" in this case) is satisfied we execute either the Implement and Archive components or the Revise and Restart components.

When the "goal" (constraint) of the vote component is fulfilled the vote activity is complete and the policy process moves onto the next activity. What the next activity is, depends on the outcome of the vote with reference to a precondition ("proportion_votes_for > 50%") for moving to the the next process.

System Requirement: ***need for branching in the process-based on preconditions***

The process definition should allow customisation of certain stages in the process. For instance the voting stage could be configurable to facilitate majority decision or consensus with single member veto. While it is hoped that the process definition will be configurable through a simple web interface, this is probably beyond the scope of this project.

System Requirement: ***configuration of components as part of process definition***

P2 Creating Roles - Assigning Permissions to Groups

Every component in a process has an interface which is made up of a number of *method signatures*. Each method signature can be assigned to any number of groups. The set of method signatures which any group has assigned to it for a process type is considered their role in the process.

This component allows roles in a process to be created thus assigning access to methods to members of the relevant groups. This component is of course dependent on P1, the *process type definition* component, since it is not possible to assign *method signatures,* until we know which components constitute the process.

System requirements: **A component for assigning permissions in a process type to a group, thus creating roles in a process type each associated with a group.**

P3 Create Process Type

Once a process type definition, and the roles in the process type, are fully specified, the process type will have to be created and added to the list of processes which can be initiated from the group. This needs to be a separate component so that it is possible for users to propose process type definitions and roles, which will only be realised under certain conditions. For instance, we may insert a *vote component* between P3 and P2 and then execute a different branch of the process definition depending on the result of the vote.   P3 is dependent on both P1 and P2 having occurred earlier in the process.

System Requirement : **Component to create previously defined process types**

Joining and Leaving Group

J1 to J4 are the components that will be required in order for groups to have processes with which users can join or leave.

J1 Apply

This component should provide the necessary actions for the user to *apply to join* the group. The component should determine any user-attributes which are required by the group, but not already included in the user's profile. It should then request that the applying user enters the required details.

For example the main Open Coop group might require the fields: *username, password,* and *email address*. While the Open Food Group might require the fields *username, password*, *telephone*, and *vegetarian*. When a user who is already a member of the Open Coop applies to the Food group, they should be requested for their telephone number, and whether or not they are vegetarian.

System requirement: **The component needs to be able to identify which of the required attributes are already fulfilled and which still need to be added to the users profile.**

N.B By assigning the permission for initiating a group's, *apply to join,* process we can control who can apply to join a group. For example, if we assigned the permission to the *Open Coop* group, it would effectively be possible for every registered user to join the group. However if we assign the role to some other group X, then it will become necessary to join X before joining the original group.

J2 Approve application

In this component someone with the role to do so, can approve the application which was entered in J1. J2 is dependent on J1. When J2 is complete, the user who applied in J1 will be a member of the group.

If a group wanted to have some kind of vote on whether or not they should approve an application for membership, this could be done by inserting a vote component in the *join* process, between J1 and J2.

System requirement: **Component for a user to approve an application, and add a user to the group**

J3 Request to leave

If a user is already part of a group, they should be able to ask to leave the group using this component. It might also be possible for other users to suggest that a user is excluded from a group. This could use the same component, with a slightly different configuration.

System requirements: **Component for a user to leave the group**

J4 Approve Request to leave

This will remove a user from the group, and is dependent on J3. Of course, there may be other aspects of this process, for instance if a user is under contract. If the user has been asked to leave the group by another user there may be a debate and/or vote involved. Such other components should be placed between J3 and J4 in order to influence whether or not J4 will be executed.

System requirements: **Component for a user to be removed from the group**

Creating Sub-Groups

Users should be able to start new groups within existing groups by initiating the process to “create a new sub-group” (assuming they have the necessary roles). This process constitutes a group design exercise in which the stages of mapping the corresponding virtual organisation into the VOOP are executed. These stages will include defining the purpose and principles of the group, as well as designing the groups process types. Since creating new process types makes use of components P1 and P2 this is a good example of reuse of component interfaces.

S1 Begin creating a Group

Initially a 'purpose' and a set of 'principles' should be prompted for. This can be specified in text and act as guidelines for participants to think about the organisation. The component must also prompt the user to create the types of process that the group will be initiated with.

System Requirement: **Component should prompt for purpose, principles, and process type creation**

The next stages will involve defining governance processes for the group. Therefore it will be possible to use the components P1 and P2 for defining processes and creating roles.

S2 Finish Creating the Group

Once the required processes have been added to the group, this component creates the new group. S2 is dependent on S1. Once this activity is completed the group will have a new subgroup, with the corresponding processes created by components P1 and P2.

System Requirement: **To be able to create a new group, as a subgroup of the group where the process was initiated.**

Policy Document Management Components

In order to be properly considered a governance system it will also be necessary to have components for managing policy creation. These would include: *proposing policy documents, debating, revising, and voting/rating*.

System Requirment: **A simple set of components for managing policy documents**

4.3.4 Distributed Features

Ideally this project would be implemented as a distributed system of communicating web servers. Each server would host groups and would communicate in order to create seamless services. Anticipated services requiring distributed communication between instances of VOOP include:

1. Creating relationships between groups on different servers. For instance a group on one server may have a role in a process in a group on another server.
2. Creating topological graphs, for instance drawing graphs where the nodes are groups, and the arcs connecting the nodes are roles in the processes of other groups.

System requirement: **An architecture that supports future development of distributed services. Due to time constraints it is not expected that any distributed services will be implemented as part of this project.**

5 Architectural Design

The previous section specified the required features of the VOOP application. This section will outline considerations on how to best design the VOOP system in order to realise those features.

5.1 Component-based Workflow

Software components are reusable sections of code which can operate independently of each other. A component has an **interface** which documents the methods which are available to the outside world. An interface makes it possible to *use* the component without understanding *how* it works. The concept of an interface maps well onto the idea of a user-interface. Indeed, in this paper, where an interface is referred to, its methods are user-actions through which users can interact with a component, thereby accomplishing the *task* of the component.

The idea of a component in the context of workflow, is analogous to the concept of a *higher-level activity* [9], in the context of the study of processes. That is, for any process with a particular purpose there are a limited amount of components, or higher-level activities, that are likely to be involved. For instance, there are many different processes for buying a meal in a restaurant, but they are all likely to include the components: *order, pay, serve, eat.* The same is true in the field of organisational governance, so for instance the process of making an organisational policy is likely to include the components: *propose policy, debate, vote/rate, revise, and implement.*   Thus VOOP will be implemented as a component-based workflow system.

As noted earlier, a higher-level activity can be usefully considered as a *goal.* So it would be useful to be able to specify when a higher-level process is complete, using a representation of the goal. A goal can be effectively represented by a *post-condition* constraint. When the constraint is fulfilled a *transition* will occur from the current component to the next. Therefore a simple example of a component-based workflow should look like this:

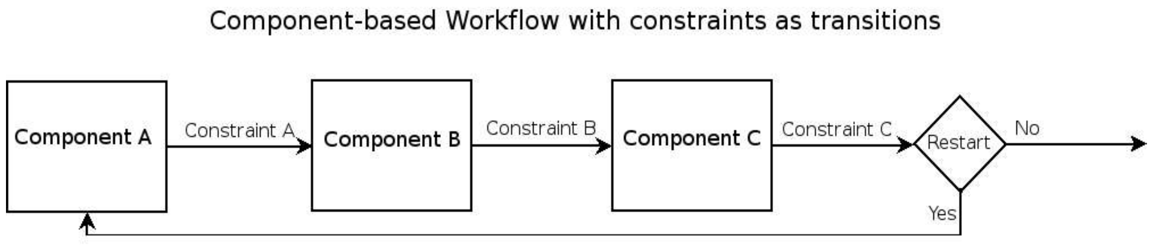


Fig 5.1 Component-based Workflow with Constraints

The *post-condition* constraints (A, B, and C above) can be used to configure the component to behave in various ways. For instance, in the *debate* component in a *policy-making* workflow it should be possible to set the constraint on any relevant *property*[10] . So the constraint could be set on the *time since the debate started,* or alternatively the *amount of participants* involved in the debate.

It is also important to consider what happens to the workflow instance once the workflow definition has been completed. In some cases we may want to restart the workflow (whilst archiving the data from the workflow *cycle* that has just been run). Alternatively, we may want to simply finish the workflow instance and archive the whole workflow instance. Finally, we may want to restart the

[10] A property here refers to some aspect of the state of the workflow. In the implementation instead of using attributes to maintain state properties are used since they are more fundamentally relevant to constraints. A property in this context is a key-value pair where the key can be any type of object.

workflow instance with the existing data in place.

As explained in *section 4.3.3* it is also necessary for *if-else* branching in the process based on preconditions. This means that different *branches* of components need to be executed dependent on a condition. This will look like this:

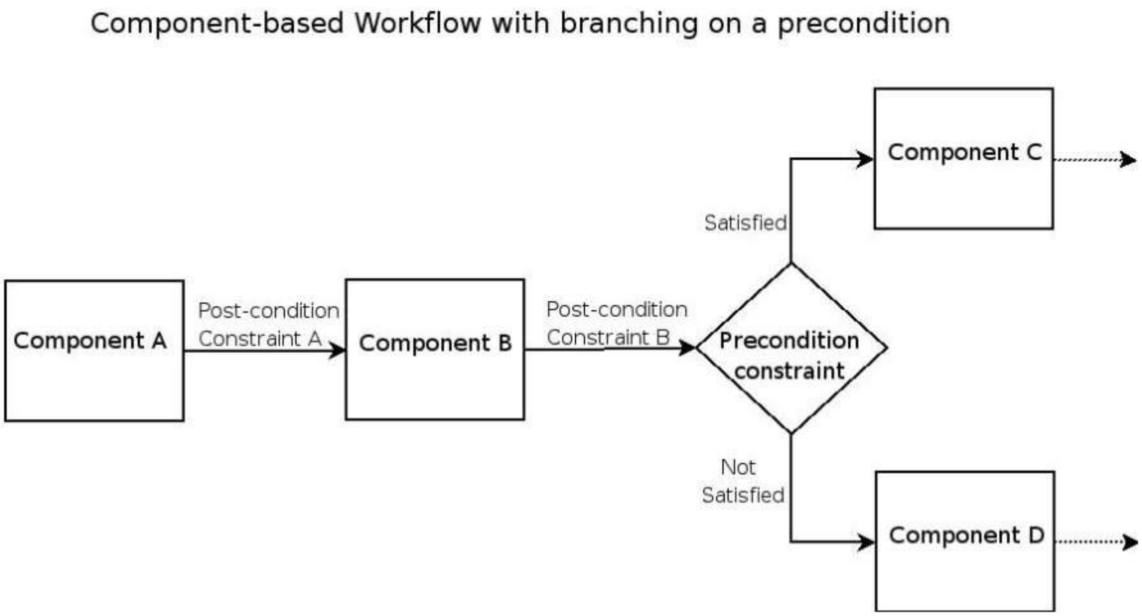

Component-based Workflow with branching on a precondition
Component A
Post-condition Constraint A
Component B
Post-condition Constraint B
Precondition constraint
Satisfied
Component C
Not Satisfied
Component D

Fig 5.2 Component-based Workflow with Branching

It is important to note that this type of branching (in Fig 5.2) is distinct from the more general type of branching in traditional activity-based workflow, which allows representation of parallel task execution. The branching on the *pre-condition* in VOOP is only used for XOR (exclusive OR) branching. Whereas traditional workflow engines allow OR (inclusive OR branching), AND branching, and a similar set of constraints for re-joining the workflow branches. Since each of the components in VOOP facilitates multi-user concurrent interaction, this type of branching for parallelisation of user/automated tasks will not be necessary. It may however be necessary to add this in the future so that a workflow instance can implement interfaces to multiple *components* in parallel.

5.2 Why NOT use an existing workflow engine

Most existing workflow engines are designed around a state-machine with states which are user-tasks (or automated tasks), and transitions between states on completion of a user-task. Tasks are normally distributed to the level of detail of individual user-actions. This is known as activity-based workflow. The components in VOOP are intended to facilitate *higher-level activities* which are likely to be multi-user in character and define a type of interaction between users (e.g. debate). Thus the detail in traditional workflow specifications is too low-level for this purpose, and would therefore be too restrictive and expensive to customise. Indeed, Shi, Lee and Kuruku [14] state that most process automation today is achieved through standardisation to “best practice” processes which can be costly to re-engineer.

Most attempts to create component-based workflow integrate traditional activity-based workflow management systems with components[11][23] by considering the component as a *task* in the workflow definition. This approach has the disadvantage that the specification of the workflow is not inherently component-oriented. That is the specification language is not written in terms of interfaces and configuration parameters for the corresponding components. Thus the flexibility of component configuration is limited in this approach.

In addition implementation of this type of standard is complicated by having two layers of abstraction rather than one. In the OMG Workflow Management Facility [23] there is both a CORBA based component system and the traditional activity-based workflow specification standard of the WFMC[12] [24]. Integrating these two standards into an implementation seems unnecessarily complicated when it is possible to have both systems in a single layer of

[11] OMG integration of WFMC with CORBA to make Workflow Management Facility

[12] Workflow Management Coalition (WFMC) – Workflow Reference Model

abstraction. Of course since CORBA is a distributed object system this has the advantage of facilitating *distributed* components. However as we will see in section 6.7 there are also possible solutions to the problem of distributed component workflow in the VOOP architecture.

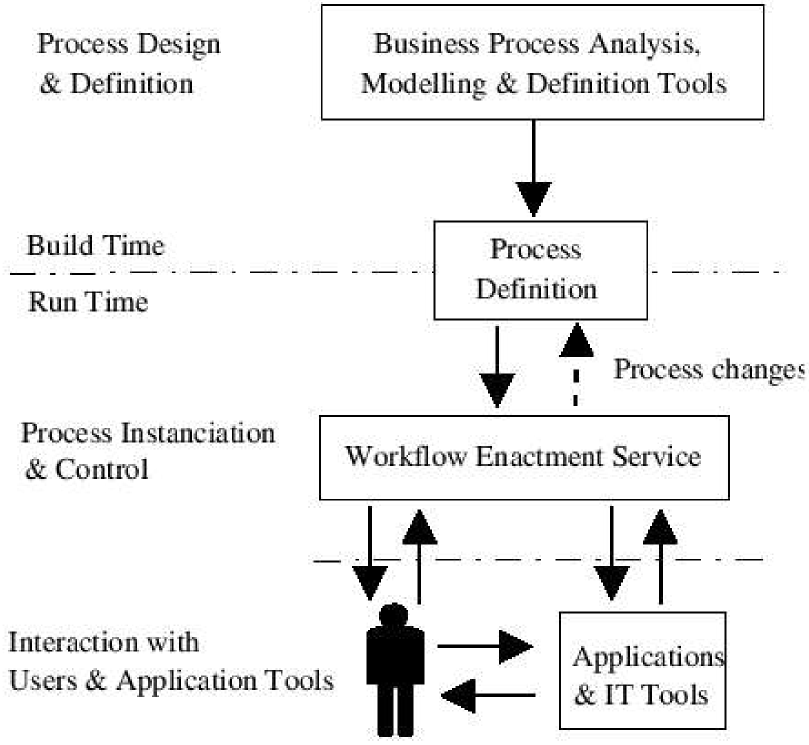


Fig. 5.3 Workflow System Characteristic (extract from Workflow Management Facility V1.2)[23]

Figure 3.2 shows the archetypal activity-based workflow management system. Note the separation between the layers: *process definition, workflow enactment service* and the *applications and IT Tools*. The VOOP system is designed in such a way that the workflow enactment layer and the application layer are one single layer. In VOOP the workflow definition is specified directly in terms of interfaces to components. Therefore the process definition *directly references* the applications layer (in this case the components). In VOOP the *enactment layer* for an instance consists only of the instance object which has a workflow definition. On parsing the definition, the workflow instance acquires the required behavior, by using the interface to find the corresponding component (explained in section 5.3).

Some more innovative approaches to component-based workflow suggest using an events-based programming model for communication between components [25] . Zhuge [15] and Shi et al [14] have suggested developing inherently component-based workflow systems, where the workflow definition is written in terms of components. These approaches are related to the approach taken in designing VOOP, and indeed indicate some of the possible future extensions of VOOP. However, the free availability of any related implementation to date, is currently not known by the author.

5.3 The VOOP Architecture

VOOP is a *component-based* workflow system as outlined in section 5.1. VOOP consists of a number of key building blocks which are outlined below:

A **Group** is the fundamental *context* within which everything else exists.

A group may contain, and provides the *context* for:

1. Members which are **users** who have joined the group, or who created the group.
2. **Workflow types**, which define the types of processes that can be instantiated in the group.
3. **Roles:** Groups may have roles in the process types of other groups as well as their own.
4. **Workflow instances:** Most types of workflow instances change aspects of the group. Such process instances effectively modify their own *context.*
5. **Sub-groups:** A group may have other groups nested within it. In fact, the only way of creating a group, is from within an existing group. The system of groups is hierarchically structured, with a single *seed* group, within which multiple sub-groups may be defined. The *seed* group is created when the *application* is launched without any groups in the database.

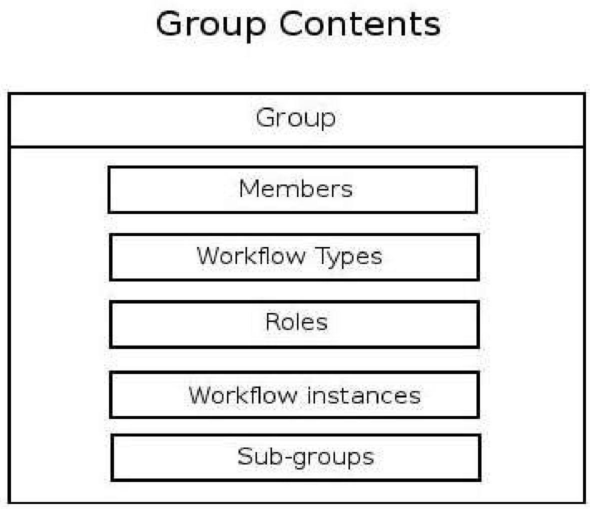


Fig 5.3 Group Contents

A **workflow type** is a class which acts as a *factory* for workflow instances (it produces workflow instances). In VOOP, a workflow type is configured with:

1. A **workflow definition,** which is an ordered list of interfaces. The interfaces are augmented with constraints, and configurations parameters. Such constraints and configurations are applied to the components corresponding to the interface on which they are defined..

2. **Roles,** which are sets of permissions in a process type, which are assigned to groups. Each method in an interface corresponds to a single permission which can be assigned to any number of groups.

A workflow type creates workflow instances with the *workflow definition* and the *roles* it is configured with (as seen in figure 5.4).

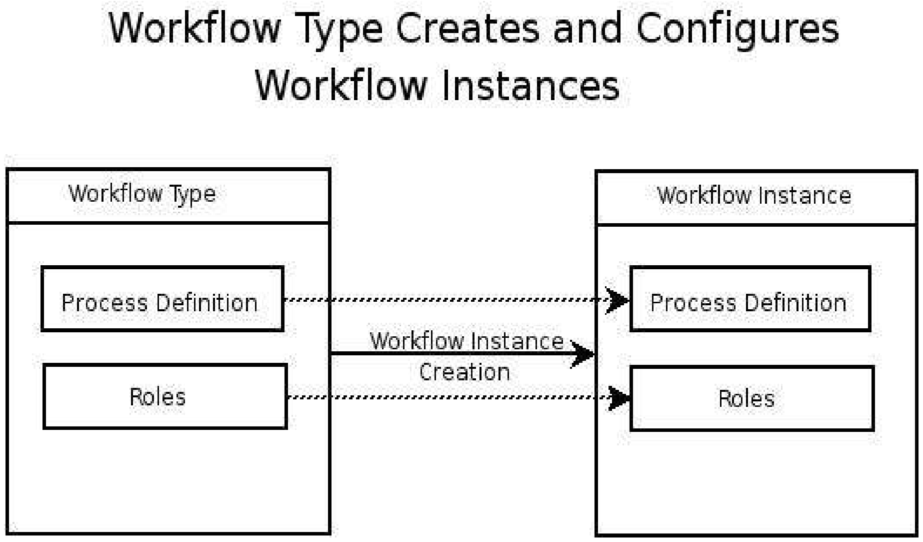

Workflow Type Creates and Configures Workflow Instances
Workflow Type
Process Definition
Roles
Workflow Instance Creation
Workflow Instance
Process Definition
Roles

Fig 5.4 Workflow Type Instantiation

A workflow instance is a *state container* which represents the *state* of an organisational process. There are no *methods* intrinsic to workflow instances which operate on its *state.*

The only methods with which workflow instances are created, are those relating to generic functions, such as parsing the workflow *definition*, and retrieving the corresponding workflow components which implement the interfaces in the *workflow definition*.

A special aspect of the *state* of a workflow instance is the interface(s) which it currently implements. This corresponds to the current stage in the *workflow definition*.

The *workflow definition* tells the workflow instance which interface is required for the next *higher-level activity*. The system must ensure that the behaviour of the interface is implemented by the workflow instance. This will require adding the functionality of a component which implements the interface to the workflow instance. For example, when a workflow is in the *vote* component, it will provide a user-interface for voting, and the voting functionality will be provided by the workflow instance through the *vote* component. When the workflow has progressed, it is important that the functionality of this component is no longer available. When a workflow instance progresses through its workflow definition, it must *acquire* the necessary functionality to *provide* (implement at run-time) the next interface. The *methods* which are *acquired* act as the workflow instance's methods. Therefore they can affect its state.

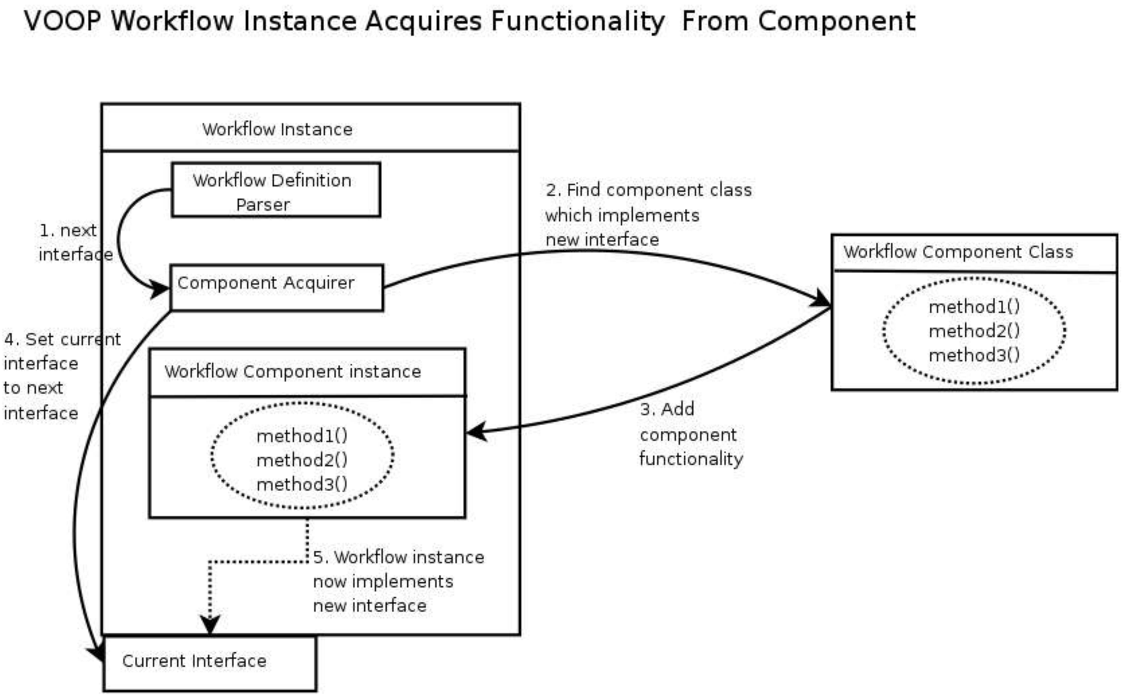

VOOP Workflow Instance Acquires Functionality From Component
Workflow Instance
Workflow Definition Parser
1. next interface
Component Acquirer
2. Find component class which implements new interface
Workflow Component Class
method1()
method2()
method3()
4. Set current interface to next interface
Workflow Component instance
method1()
method2()
method3()
3. Add component functionality
5. Workflow instance now implements new interface
Current Interface

Fig 5.5 Acquiring Component Functionality

The acquisition of interface functionality has five stages as depicted in fig.5.5:

1. The workflow definition parser finds the next interface and passes it to the Component Acquirer.

2. The Component Acquirer *somehow* (this is explained in the Implementation chapter 6.1.3) finds a component with the necessary functionality (methods) to implement the interface.

3. The component's functionality is added to the workflow instance.

4. The Component Acquirer sets up the new interface to the workflow instance.

5. The workflow instance now has the necessary functionality to implement the new interface.

N.B This acquisition of functionality is achieved in the implementation through a python specific programming concept known as component adaptation. This is explained in the Implementation section.

Once a workflow instance has acquired the functionality of a workflow component, users (with appropriate roles) may execute the methods of the new interface, and thereby operate on the *state* of a workflow instance (or on the *state* of the *group* which is the context in which the workflow exists).

When the workflow instance acquires a component, a *constraint* is set by the *workflow definition* on the component. Constraints are checked against the state (properties) of the workflow instance (and/or the state of the *group*). Once a constraint is fulfilled the component functionality is removed, and a new component is *acquired*.

A *component method* is any method added to a workflow instance through the acquisition of a component. When a *component method* is executed it may modify the state of the workflow instance.   After a *component method* is executed, the component checks whether its postcondition constraint is satisfied by the workflow instance's state. Here the question is: “*has the goal of the higher-level activity been achieved?*”. If the postcondition is satisfied by the *state* of the workflow instance, then the workflow instance should execute a transition which will involve changing its interface and therefore changing its functionality. The subsequent interface is determined by parsing the process definition. This process is depicted in fig. 5.6.

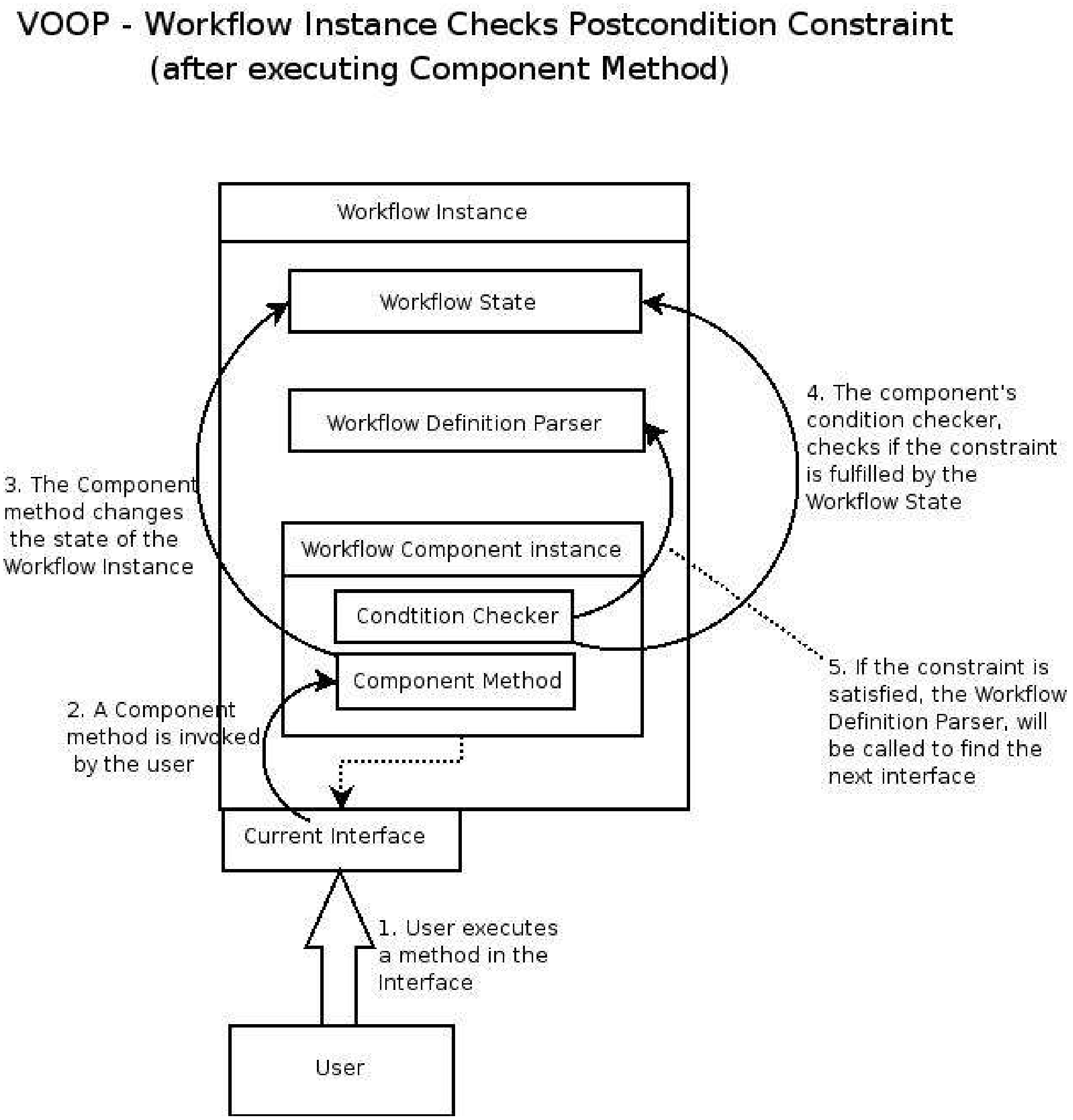

VOOP - Workflow Instance Checks Postcondition Constraint
(after executing Component Method)
Workflow Instance
Workflow State
Workflow Definition Parser
Workflow Component instance
Condtition Checker
Component Method
Current Interface
User
1. User executes a method in the Interface
2. A Component method is invoked by the user
3. The Component method changes the state of the Workflow Instance
4. The component's condition checker, checks if the constraint is fulfilled by the Workflow State
5. If the constraint is satisfied, the Workflow Definition Parser, will be called to find the next interface

Fig 5.6 Checking Component Constraints

Precondition constraints may also be checked against the state of the workflow instance. These are checked in order to create branches of the if-else type in the process definition. Preconditions for branching in the workflow instance are tested after a postcondition for the previous component has been satisfied (as depicted in figure 5.2).

A **role** in a workflow type is a set of permissions projected onto the *method signatures* of the interfaces in a workflow type definition. Each role in a workflow type is associated with a group. There maybe multiple roles onto a single process type.

A **user** may acquire a role by becoming a member of the group. Equally the user may relinquish a role by leaving the group. A user can execute *component methods* which are made available. Which methods are *available* to a workflow instance is determined by a combination of the current interface of the workflow instance, and the users roles relating to the workflow instance.

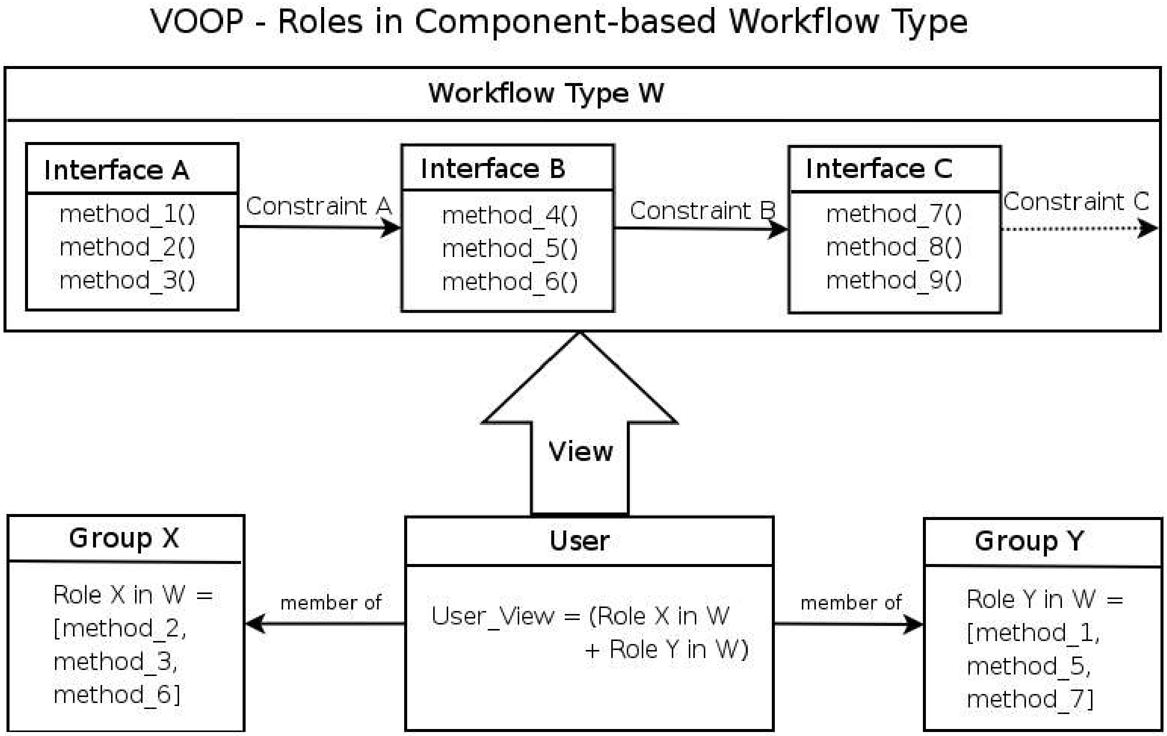


Fig 5.7 Roles and the User View

In fig. 5.7 the user is a member of groups X and Y. Both X and Y have roles which correspond to sets of methods in the various interfaces in the workflow type, W. The user acquires both of these roles since it is a member of both groups. The combination of these roles give the user a *view* onto the workflow type.

Of course at any one time there is only one interface to a workflow instance available. So, if for example, a workflow instance, of type W, was in the stage corresponding to Interface B, then since the *user view* contains methods 5 and 6 but not method 4, the user would see an interface where it could execute methods 5 and 6 only.

6 Implementation

In Chapter 5, I outlined the overall architecture of the VOOP system. In this chapter, I outline some key implementation decisions. I briefly describe the tools which I used and why.

6.1 Choice of implementation language

The language in which VOOP will be implemented is Python. Below I highlight some reasons why, as well as explaining some key language features which are used in the VOOP implementation.

Python is a dynamically typed, *interpreted, interactive, object-oriented* programming language. It is highly productive for many reasons amongst which are:

1. There is no compilation step required in the development cycle;
2. It has some useful high-level built-in types such as lists and dictionaries;
3. It is very flexible, allowing easy combination of object-oriented and functional styles of programming.

6.1.1 Checking Constraints using Metaclasses

Python has some useful features for modifying its own semantic structures. In particular *metaclasses* are the factories which make *classes*. In most programming languages, these are hidden from the programmer. In Python, by modifying the *metaclass* from which a class is derived, we can modify the behaviour of the class itself. In VOOP the *workflow components* checks the postcondition after the execution of each *workflow method*. This is achieved by modifying the *metaclass* of the *component classes* so that every time a a method is executed on an instance of a component, the postcondition is also checked.

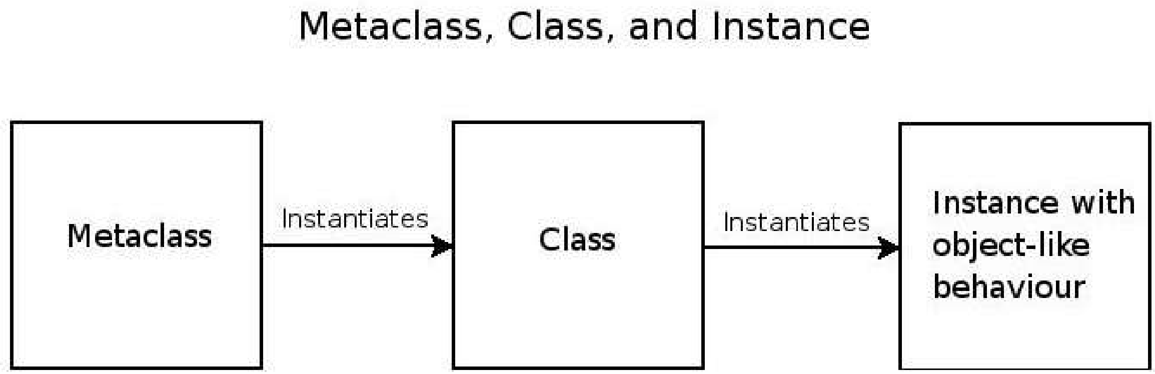

Metaclass, Class, and Instance
Metaclass
Instantiates
Class
Instantiates
Instance with object-like behaviour

Fig 6.1 Metaclasses, Classes, and Instances

6.1.2 Creating Workflow Definition Notation

The language libraries which are used by the python *interpreter* are available from within Python, making it easy to construct new languages, including declarative mini-languages. These are used in VOOP to create the constraint-based workflow definition notation with which we define workflow types (see chapter 7).

6.1.3 Concept of Adaptation in Python

Adaptation, is a programming construct which exists in Python largely due to the pioneering work of Phillip J. Eby on the PyProtocols package [26]. Adaptation is used in VOOP to achieve the implementation of interfaces by workflow instances, dynamically at run-time, as described in section 5.3. This is the mechanism through which *component* functionality is added to the workflow instance.

There are two slightly varying implementations of adaptation in Python, these are: the PyProtocols package, Zope Interfaces. However both have some basic conceptual aspects in common, and there is the suggestion that adaptation should be included as a fundamental programming construct in Python [27].

An adapter changes the behaviour of an object so that it can implement an interface which it might previously not have implemented. That is, an adapter allows us to add functionality to an object at run-time.

During the process of adaptation the fundamental question we are asking is: can I find a way of making X do Y?   So we can ask the programming environment to return a version of an object X which implements an interface Y. Behind the scenes the object X is asked, "do you know how to wrap yourself to provide this interface Y"? If the answer is no, the interface Y is asked, "does the object X provide or implement you? Or do you know how to wrap it in order to do so"

In practice, the most simple case of *adaptation* is: if an object of a particular class, does not implement or provide an interface, it may be able to adapt to that interface if it has a registered adapter for doing so.

6.2 Python-based Tools

There are a wide variety of high quality libraries available for Python. Apart from using the standard libraries of Python this project has made use of two well established Python projects. These are Twisted [28], and Zope [29]. Both are highly componentised frameworks so it is possible to pick and choose components and integrate them.

6.3 Authentication with Twisted Cred

The authentication mechanism in VOOP utilises the *Twisted Cred* package. *Cred* separates the process of authentication into two constituent parts, the *credentials checker*, and the *realm* (see figure 6.2). These components must be built by the application programmer. When an identity is created in VOOP, it adds a username and encrypted password to a file. When a request for login occurs (in VOOP this will usually be over HTTP) the portal uses the credentials checker, which is provided with the same encryption function (hash) and password file, to find the associated *avatar id* (assuming there is one). An avatar is a business logic object for a particular user, in other words it can be considered to be the *user* object, or a user's *view* into the system. The portal then takes the *avatar id* and sends it to a realm. The realm returns the avatar to the Portal, which sends the avatar on to the user via the protocol (HTTP). In the case of the VOOP implementation, the avatar is the web application returned to a user logging into the site.

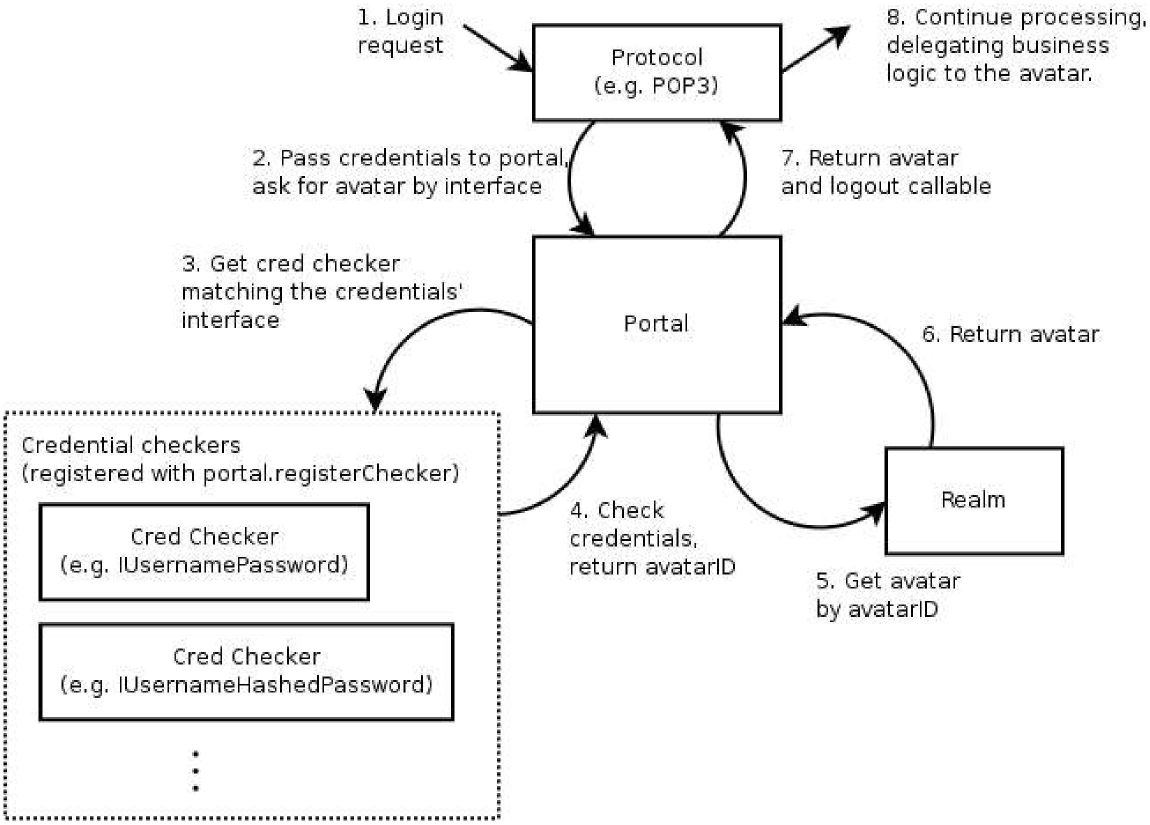

1. Login request
Protocol (e.g. POP3)
8. Continue processing, delegating business logic to the avatar.
2. Pass credentials to portal, ask for avatar by interface
7. Return avatar and logout callable
3. Get cred checker matching the credentials' interface
Portal
6. Return avatar
Credential checkers (registered with portal.registerChecker)
Cred Checker (e.g. IUsernamePassword)
Cred Checker (e.g. IUsernameHashedPassword)
4. Check credentials, return avatarID
Realm
5. Get avatar by avatarID

Fig 6.2 Authentication with Twisted Cred – from [30]

6.4 Components, Interfaces and Adapters

6.4.1 Interfaces

An *interface* (in Python) is simply a class that:

1. Acts as a marker

2. Documents a group of methods.

In *Zope Interfaces,* which are used in VOOP, an interface is created by writing a class and sub-classing the class “Interface” (as seen in figure 6.3).

```
from zope.interface import Interface

class IVote(Interface):
  def cast_vote(self, vote):
   “””Do you approve?
   ”””

  def change_vote(self, new_vote):
    “””Change your Vote
    “””
```

Fig 6.3 Zope Interfaces Example

Note that the methods in the interface do not have method bodies.

6.4.2 Components

Object-oriented programming allows programmers to reuse code through the design pattern of *inheritance*. However multiple inheritance can become difficult to manage. *Component-based* design increases usability of code through the design pattern of d*elegation,* and the object-oriented technique of *composition.*

Twisted Components, is a module for creating components which is used in VOOP

In VOOP *workflow components* can be declared to *implement* or *provide* (implement at *run-time*) an interface. For instance, the component in figure 6.4 implements the interface specified in figure 6.3. The implementing component must provide methods corresponding to all the *method signatures* in the interface.

```
Class Vote_Component:
 implements(IVote)

 def def cast_vote(self, vote):
  #method body goes here

 def change_vote(self, new_vote):
  #method body goes here
```

Fig 6.4 Component Implements Interface

6.4.3 Adapters

Since the *workflow instance* will need to provide the behaviour required by the “IVote” interface, the *workflow instance* will need to *use* the functionality of the “Vote” *component*. Therefore we need to register the “Vote” component as an *adapter* for *workflow instances.* To do this we use the Twisted components registry as shown in figure 6.5*.*

```
from twisted.python import components

components.registerAdapter(Workflow, Vote, IVote)
```

Fig 6.5 Registering an Adapter

The adapter (Vote component) can then be used to adapt the *workflow instance* to *implement* the interface IVote. We may then ask a *workflow instance* to adapt to implement the interface like this:

```
workflow_instance = Workflow()   #create a workflow instance
IVote(workflow_instance)         #then adapt the instance
                                 #to implement IVote
```

Fig 6.6 Component Adaptation

When the code in figure 6.6 is executed, the interface looks for an *adapter* in the *adapter registry* which will allow it to implement the interface “IVote”. If it finds one, it constructs an instance of the “Vote” class, passing the *workflow instance* to the constructor. The “Vote” class is said to *wrap* the *workflow instance.* Now the workflow instance *implements* the “IVote” interface.

6.5 Web Templating Using Nevow

Nevow is a web templating system, which separates business logic and display logic. The *Nevow* project is connected to the *Twisted* project, and Nevow is dependent on the Twisted code base. Nevow integrates seamlessly with twisted servers, providing a web-interface, to *twisted applications*. *Nevow* has a number of modules which are used in VOOP.

6.5.1 Rendering Interfaces with Formless

Nevow's *formless* module allows us to *expose* interfaces method signatures to the web. Formless automatically renders web forms and coerces typed input. This is done by modifying interfaces and turning them into *typed* web-interfaces. These *typed interfaces* validate and coerce string input, ensuring it is of the correct type to call the methods. This allows us to dynamically generate web-interfaces to workflow components. As shown in figure 6.6, this is achieved by making minor modifying the Zope interfaces from figure 6.4. There are two differences between the two interface definitions:

1. The interface subclasses a class called f*ormless.annotate.TypedInterface* instead of *zope.interface.Interface*

2. The arguments in the method signature are *typed* – so that for example the vote argument must be a *boolean.* Interface *typing* is useful in order to expose Python methods to the web, since python is dynamically typed, so otherwise the user could enter any type into functions which were not intended to be used in that way.

```
from formless.annotate import TypedInterface, Boolean

class IVote(TypedInterface):
    def cast_vote(self, vote = annotate.Boolean()):
      “””Do you approve?
      ”””

    def change_vote(self, new_vote = annotate.Boolean()):
       “””Change your Vote
       ”””
```

Fig 6.6 Typed User Interfaces

Once we have specified our interfaces in this way, its possible to *expose* the interfaces to the client. The interface then acts as a *user-interface*.

6.5.2 STAN

STAN is a pure Python Document Object Model which is significantly “lighter” then the W3C DOM. STAN documents consist of a XML tree structure built from nested sequences of Python types. STAN generates XHTML. STAN allow Python programmers to leave “hooks” at certain points in the document, in order to permit subsequent manipulation.

6.5.3 LivePage

LivePage is an AJAX (Asynchronous Javascript over XML-HTTP) implementation. This allows the browser to respond to server events without the refreshing the whole page. When combined with the twisted events model on the server-side this make an effective events model for web-application development. In particular this makes the goal of real-time services in the browser possible. So for instance web-based *chat* clients applications, need not *poll* the server for new comments by refreshing the page, since events can be routed directly into the browser.

6.6 Choice of Database

6.6.1 Why an object database?

The decision to use an object database, or more precisely a “persistent object store” was taken mainly due to the heterogeneity of the objects which will be persisted (stored) in the application. In particular a *workflow instance* may use multiple different components during its life-cycle, which modify and add attributes (properties) to it. This creates many possible combinations of attributes (properties).

Since the process objects will have different attributes according to different workflow definitions, using a relational database would be cumbersome. In SQL, or any other relational database, all rows of a database table must contain a set of attributes which are determined by the columns of the table. Since process objects of each type may have different attributes, and those attributes may be of different types, this would involve creating at a minimum one table per process type. However due to the fact that individual process objects will have different fields according to the branches that are taken in an individual process, even this would lead to having some large tables in which the majority of the fields are often empty. *In short, relational databases is not well suited to objects which are heterogeneous in their attributes.*

Whilst having other drawbacks, object databases, make having objects with heterogenous attributes very easy because object databases make objects persistent by serialising them and storing the serialised version.

However, one major disadvantage of using an object database is that the objects are queried and retrieved only by a single key through which they are stored. Using SQL the data querying method is intrinsic to the DBMS. In object databases, this is sometimes overcome by creating indexes of the objects stored and then querying those indexes to retrieve the object ID before querying the database for the object with that ID.

Another important feature of Object Databases, is that stored objects, are not usually accessible by other languages since they are serialised versions of objects in the programming language. They are also often created using a serialiser which is usually language specific. This is clearly a potential problem, if we are later likely to want to access the data from another language. This could be solved by serialising the objects into an external data representation. However there are advantages of having the data stored in the programming language of the application. In particular, since querying the database is done independently from storage, querying and indexing of data is done in the language of the application programmer. This is seen by some as an advantage, since the programmer need not learn or use, another language, such as SQL, in order to understand or construct code relating to data retrieval.

6.6.2 Why ZODB?

ZODB (Zope Object Database) is probably the best established object database written in Python. It was originally part of the Zope project, which is a web applications development framework, and one of the biggest and best known Python projects. As part of the release of Zope 3 many of the previously monolithic parts of Zope have been modularised. Therefore now it is possible to use ZODB, without using the rest of Zope.

ZODB has many advantages when compared to other object databases in Python. These include:

1. **Syntax doesn't interfere with application logic –** it is very simple to make the instances of a class persistent using ZODB. Simply sub-classing the class “persistent” will have the desired effect.

```
class Process(persistent):
   ...
```

Fig 6.7 Making an Object Persistent in ZODB

ZODB also requires that transactions are *committed* by the application programmer, and that if *container* data types (e.g. lists or mappings) are used, that the application programmer sets the “dirty bit[13]” manually when an element in the *container* is changed. The level to which *committing* interferes with application code is dependent largely on the granularity of transactions. However in the case of most web applications, a single web request is usually considered a transaction, so it is possible to just commit at the end of each request.

2. **Scalability** - ZODB can be scaled to support heavy loads, and many objects. This is achieved through two main mechanisms. First, the programmer can choose the type of data structure in which to store the objects. So for instance, the programmer can choose to store objects in a *btree*, which is highly efficient for querying (btrees are used in VOOP). Second, through using ZEO (Zope Enterprise Objects), ZODB allows multiple storage servers, and processes, to be integrated to provide a single *transactional object database* for an application. Whilst this is not used in VOOP, it is possible to add at a later date with virtually no modification to the

[13] Database terminology – when the data in the cache has been modified but not written back to disk the “dirty bit” should be set.

application code.

6.7 The Potential for Distributed Services in VOOP

As mentioned in the specification, whilst no distributed services are to be developed as part of this project, it is essential that VOOP is designed to support easy extension with distributed services. So, for example, a group which is on an instance of VOOP on one server may have a role in a workflow in a group on another server. To the user, we would want this to be a seamless process. So when the user navigates to the group it should appear to the user as 'just another group'. There are two *Twisted* services which together make such *distributed services* relatively trivial to achieve. These are *twisted cred* as outlined in *section 6.3* and *perspective broker.*

The use of *twisted cred,* and particularly the separation between the *protocol* through which a request comes, and the *portal*, has a key advantage. A request may be made from another server over a distributed object *protocol* (i.e. a remote method invocation protocol), and can be returned via the same protocol. By sending the *avatar id* in the request, we can get the same user object as would be provided to the user. Alternatively, a *service* may login instead of a user. For example, a *node mapping service* which maps the relation between distributed groups which have roles in each others' workflow types. Similarly customisable results can be returned according to the *avatar id* of the *service.*

Perspective broker is a distributed object protocol and is part of the twisted set of tools. *Perspective broker,* consists of serialisation and remote method invocation. In order to invoke a method on a server, the client needs to have a remote object reference. By logging in through *twisted cred* an avatar is returned to the client. The *avatar* represents the user's abilities to do things on the server. The avatar gives the client appropriate object references which become the user's *perspective*. The *perspective* allows users logged into one server to make method invocation on another server.

7 Constraint-based Workflow Definition Notation

In Chapter 5 the concepts of a *workflow type*, *a workflow type definition*, and *roles* relating to a workflow type, were introduced. It was explained that a workflow definition and roles are constituent parts of the workflow type which are transferred to the workflow instance on instantiation. The process of execution of the *interface* and *constraint* constituent parts of a *workflow definition* was also explained. This section describes in more detail how workflow type definition, and role definition takes place in VOOP and elucidates the syntax and features of the notation used.

7.1 Basic Workflow Definition

The syntax of the most basic statement in the workflow definition notation looks like this:

```
[Interface1 : post_condition1 | Interface2: post_condition2]
```

Fig 7.1 Workflow Definition Syntax

The above workflow definition is all enclosed in square brackets, which means that this is a single branch of a component-based workflow. The definition says that a workflow instance of this type should provide the behaviour of *Interface1* until *post_condition1* is fulfilled. The pipe ('|') tells the workflow instance that there is another stage to the workflow remaining, and that the workflow should try to *adapt* to provide the behaviour of the next interface. The post-condition constraint should be set on a *property* which some aspect of the corresponding component can modify. Once this constraint is satisfied the workflow instance moves to the next interface. The most basic building block of a constraint is a statement about the value of a property. This takes the form:

```
property1 == value1
```

Fig 7.2 Atomic Constraint Syntax

The “==” operator can be replaced with any standard inequality operator (“>=”, “<=”, “>”, “<”). In addition composite constraints can be built by stringing together these atomic constraints with boolean operators OR and AND.

```
property1 == value1 OR property2 == value2
```

Fig 7.3 Composite Constraint Syntax

7.2. Branching on Preconditions

Preconditions are used for choosing *branches* of the workflow based on the *state* of a workflow instance at the point of testing the precondition.

```
 [Interface1 : post_condition1 | Interface2: post_condition2]
if{pre_condition}->[Inteface3 : post_condition 3]
else->[Inteface4 : post_condition 4]
```

Fig 7.4 Branching Workflow Syntax

In the above definition, each *branch* is enclosed in square brackets. Once the first branch is complete, the process will come to the *if* statement. The precondition is enclosed in curly brackets ("{}"). If the precondition holds, then the branch containing Inteface3 is executed. Otherwise the branch containing Interface4 is executed.

7.3 Component Configuration

In a *workflow definition* an interface represents a stage in the process. The workflow instance provides the interface at a particular stage using a *component.* We can configure components to behave differently to some degree using the post-condition constraint. So, for instance if the interface is *IVote* and the constraint is *"percentage of members who have voted must be greater than 60%"*, then the IVote component will behave differently to if the constraint was "time is after 10pm on 9th September 2005". However there are some other aspects of the operation of a Vote component which we may want to configure differently in different workflow types. For instance, we may want to configure the “Vote” component to do “rating 1 to 10” rather than “yes or no” vote. Therefore there is a mechanism to pass configuration parameters to the workflow component from the workflow definition.

```
IVote<vote_type =['yes', 'no']> : percent_voted>=40
```

Fig 7.4 Component Configuration Syntax

In the above case we pass a list of option values to the vote component. In this case the list of options are 'yes' or 'no', however we may choose instead to pass any list of options, for example: [1, 2, 3, 4, 5], would create a rate 1 to 5 interface.

7.4 Role Creation

The role definition notation takes advantage of two Python built-in types – the dictionary, and the list. A dictionary is a mapping of key-value pairs indicated by curly brackets (“{}”). In this case the key is the name of the group, and the value is a list of methods available. A role definition is part of the workflow type and thus the same role definition applies to every workflow instance.

```
{group1:[method1, method2],
 group2:[method2, method3]}
```

Fig 7.5 Role Definition Syntax

In the above role definition, any member of *group1* would have the necessary permissions to executed *method1* and *method2,* while members of *group2* would have permissions to execute *method2* and *method3.* However, at any particular time, only those methods which are in the interface *currently provided* by the workflow can be executed.

7.5 User Interface Issues

This notation whilst basic in its current form could evolve into something quite powerful. However the challenge, more than making the notation more expressive is to present it to users in a form which is easy to understand. This is very important to the project, since it is not expected that groups will be created only by those with technical abilities.

The long-term goal is to have a drag-and-drop user-interface, in which workflow definitions can be composed intuitively and visually. The user would need to be prompted for each step. Some forms of “intelligent” prompting could also be added quite easily. For instance, if *component A* is dependent on *component B, component A* should not be available in a *workflow definition composer* until after an interface to component B has been specified.

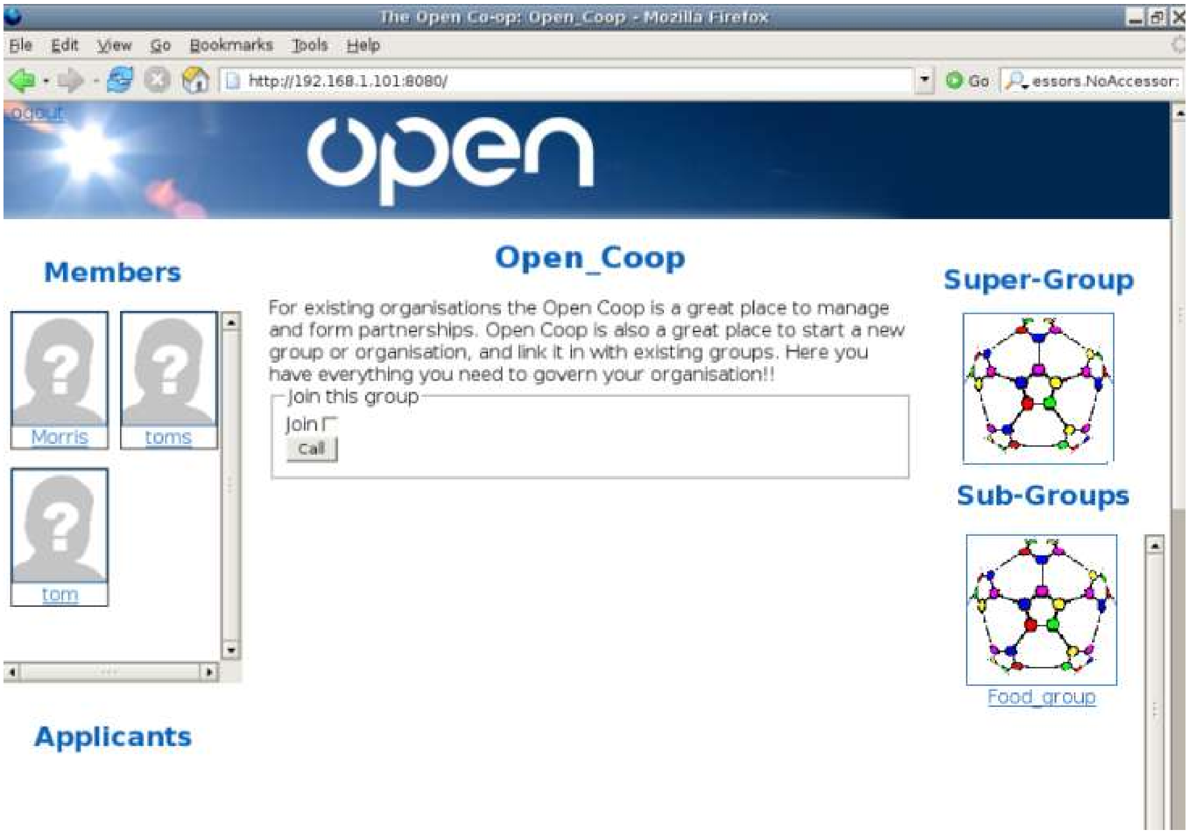
Unfortunately there has been insufficient time to develop this in the initial implementation. However, the VOOP system does allow users to construct workflow definitions, as well as define roles, using a web user-interface with drop-down boxes (see the Evaluation chapter).

8 Evaluation and Testing

8.1 Testing the application

In this section I outline the workings of the user-interface as currently implemented, with some screen-shots for illustration.

Figure 8.1 is a group home page as seen by a user who is *not* a member of this group. In fact, this is the Open Coop seed group, however it has the same layout and functions as other groups. On the left are the members of this group.


The Open Co-op: Open_Coop - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://192.168.1.101:8080/
Go
essors.NoAccessor:
logout
open
Members
Morris
toms
tom
Applicants
Open_Coop
Super-Group
Sub-Groups
Food_group

For existing organisations the Open Coop is a great place to manage and form partnerships. Open Coop is also a great place to start a new group or organisation, and link it in with existing groups. Here you have everything you need to govern your organisation!!

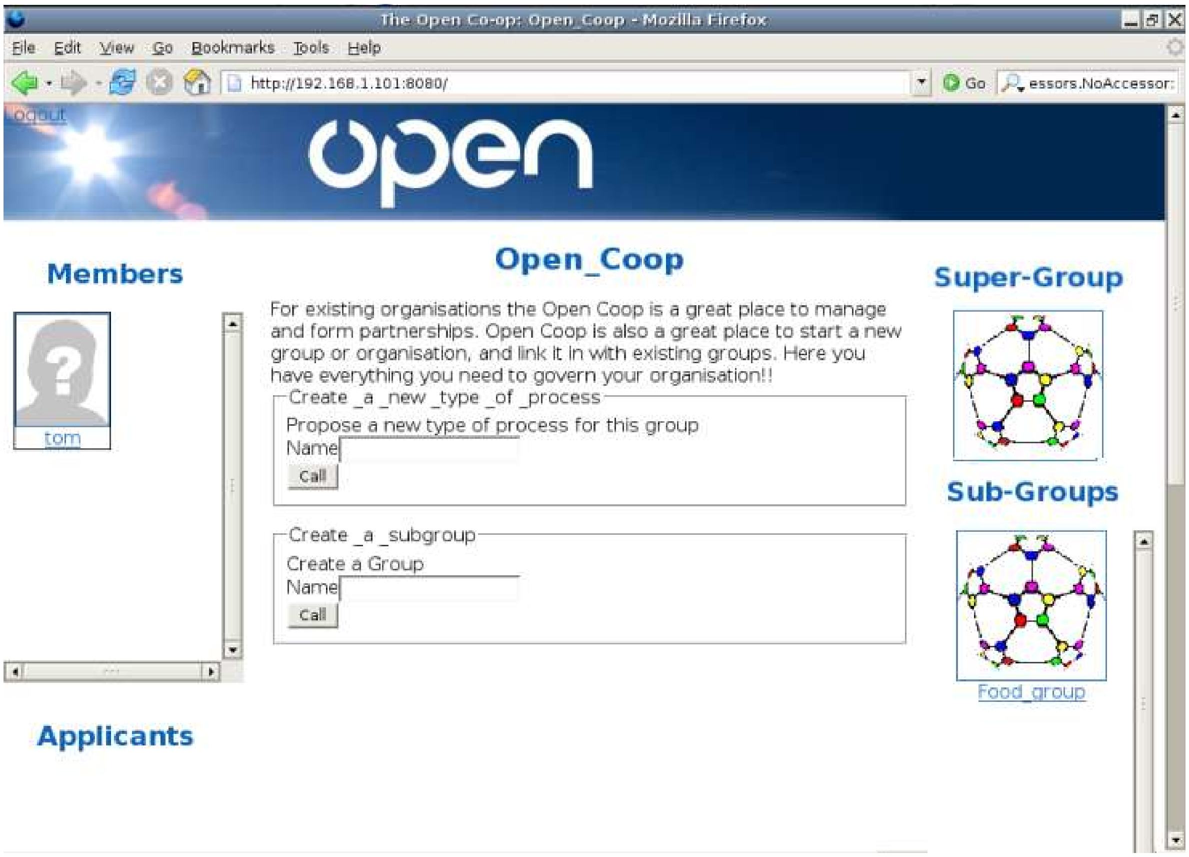
Join this group

Join □

Call

Fig. 8.1 A group page as seen by a non-member

A user can browse to each user page by clicking on the image links. Similarly, the user can navigate the group tree by clicking on the image links on the right. Unsurprisingly, sub-groups, are contained within the *current* group. and the super-group is the *parent* of the *current* group*.* Figure 8.1 also shows a *check box* for *joining the group.* When this is submitted it creates a new process which will constitute an *apply* component and an *approve* component (it could contain extra components too e.g. Vote). Once (if) the approve component is completed, the user who applied will become a member of the group and will appear in the left hand column.



The Open Co-op: Open_Coop - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://192.168.1.101:8080/
Go
essors.NoAccessor:
Logout
open

Members

tom

Open_Coop

For existing organisations the Open Coop is a great place to manage and form partnerships. Open Coop is also a great place to start a new group or organisation, and link it in with existing groups. Here you have everything you need to govern your organisation!!

Create _a _new _type _of _process

Propose a new type of process for this group

Name

Call

Create _a _subgroup

Create a Group

Name

Call

Super-Group

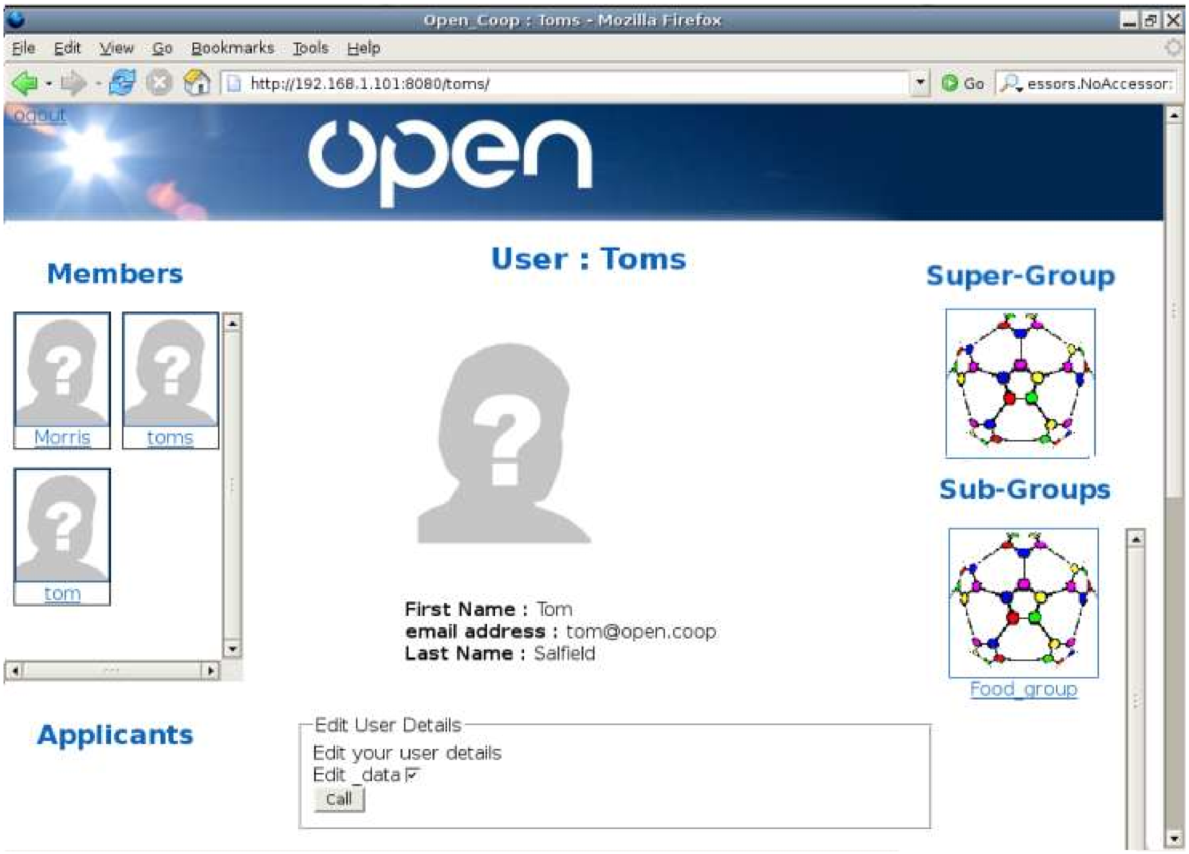
Sub-Groups

Food_group

Applicants

Fig. 8.2 A group page as seen by a member

In, figure 8.2 the user has joined the group, and hence does not see the join process but rather an *interface* (form) to create instances of each of the process types which he has the roles necessary to create. In this case, the user has the ability to begin a process to either, “*create a sub-group*” of this group, or “*create a new type of process*” for this group.


Open_Coop : Toms - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://192.168.1.101:8080/toms/
Go
essors.NoAccessor:
logout
open
Members
Morris
toms
tom
Applicants
Super-Group
Sub-Groups
Food_group
Call

User : Toms

First Name : Tom
email address : tom@open.coop
Last Name : Salfield

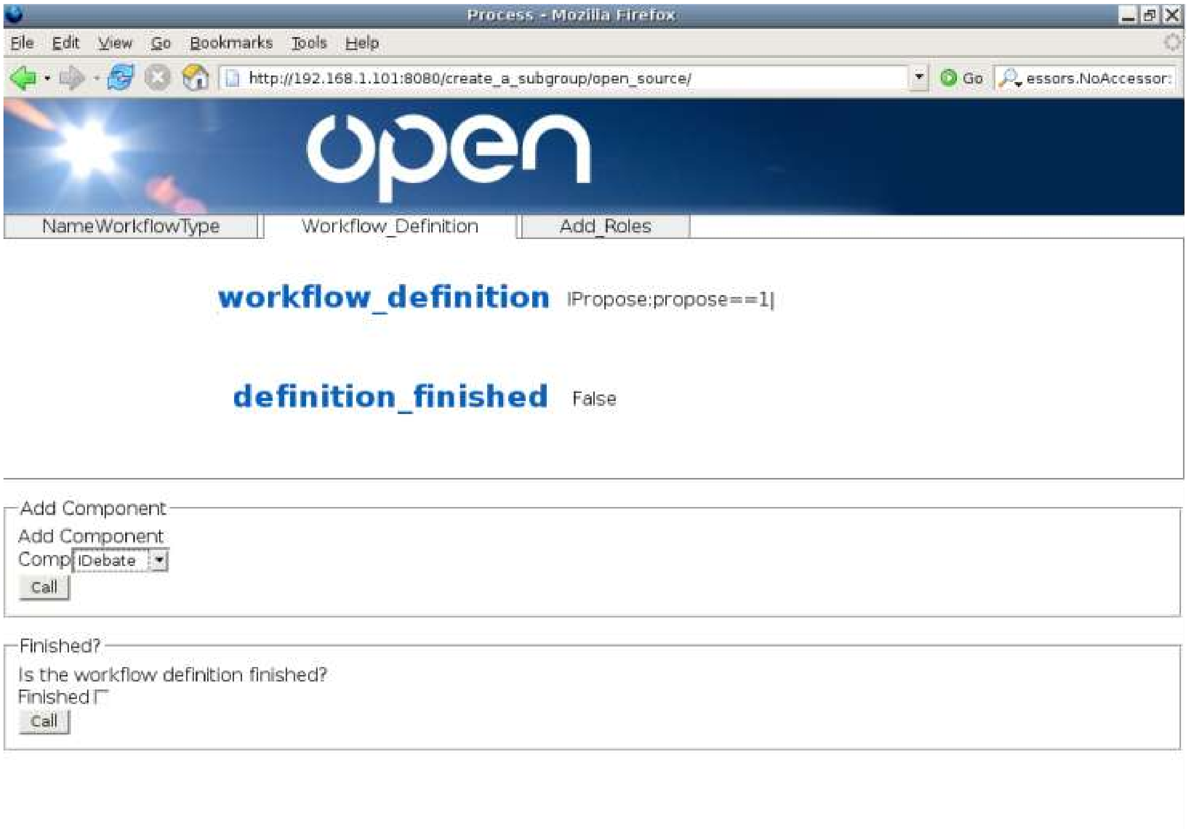
Edit User Details
Edit your user details
Edit _data ☑

Fig. 8.3 A User Page

Figure 8.3 shows the layout of a user page. Users will be able to view the profiles of all other users. When, as in this case, the user is looking at his own page he will also see the button to edit his user details. This profile is flexible and can contain any number of different fields.

In figure 8.4 we see, for the first time the layout of a process in VOOP. The process has components which are displayed in *tabs* from left to right, corresponding to the order they appear in the workflow definition. In this figure, we see that the current component is *“workflow definition”.* Inside the active tab is the data corresponding to the current component. The tabs before the current one will also contain data which is derived from their prior execution in the process instance.

At the bottom of the process instance page, is the user-interface to the current component. In the case of figure 8.4 we are in the *“workflow definition”* component and have already added an interface for creating a proposal to the new process type we are constructing. As can be seen, we are now considering adding a debate component and a corresponding constraint. When the workflow definition is complete we will check and submit the “Finished ” check box.



http://192.168.1.101:8080/create_a_subgroup/open_source/

workflow_definition IPropose:propose==1|

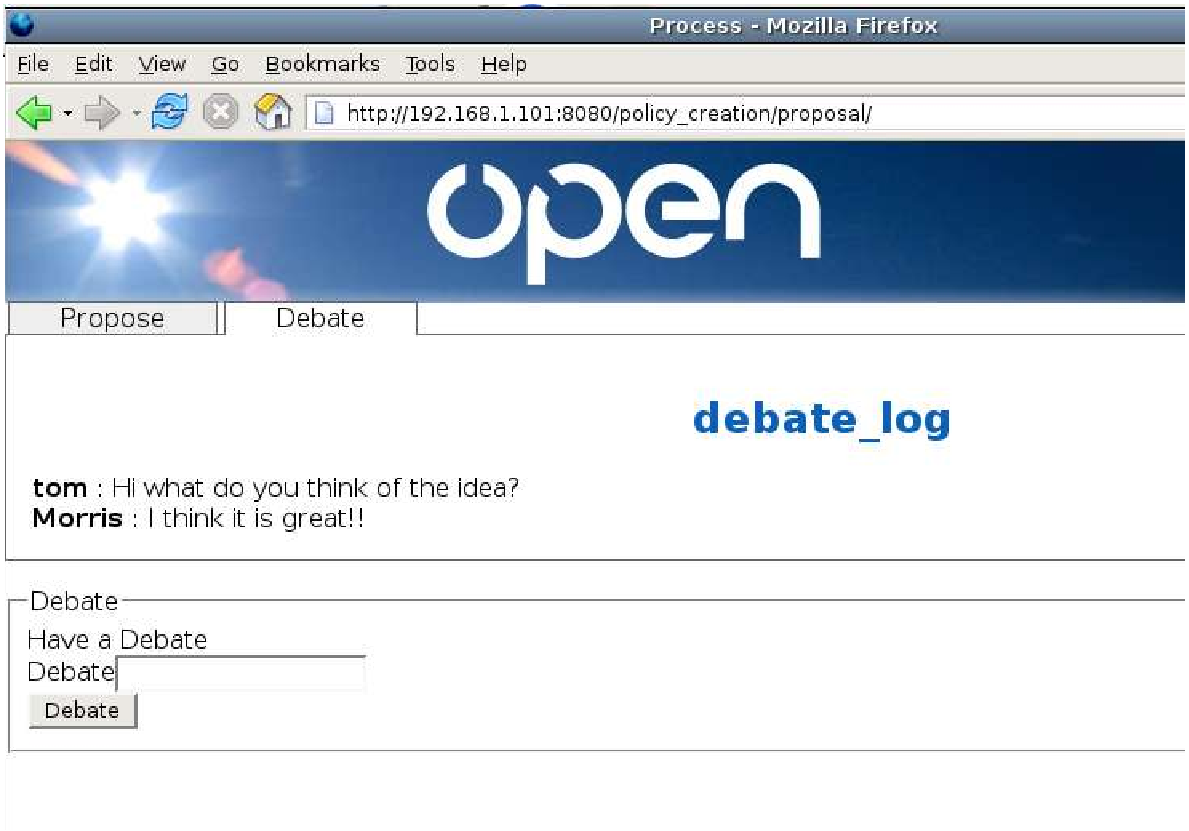
definition_finished False

Add Component
Add Component
Comp IDebate

Finished?
Is the workflow definition finished?
Finished □

Fig. 8.4 A process for creating a new process type

Once the *“Finished”* button is submitted, a constraint will be satisfied (definition_finished = = True), and the workflow instance will move on to the Add Roles component.


Process - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://192.168.1.101:8080/policy_creation/proposal/
open
Propose
Debate
debate_log

tom : Hi what do you think of the idea?
Morris : I think it is great!!

Debate
Have a Debate
Debate
Debate

Fig. 8.5 An interactive component in a process (the debate component)

Finally, in figure 8.5 we have another process type. This process contains only the components *propose* and *debate*. This figure illustrates a component which is designed to be multi-user in character. A proposal, has already been submitted, the text of which can be seen by “clicking” the “*Propose”* tab. In the debate component, users can discuss the proposal which have been made. By adding a comment in the debate form at the bottom, a user can contribute to the debate.

8.2 Achievements and Limitations

During this project, I have designed and created an application which provides a framework for specifying and enacting organisational governance processes using component-based workflow. I have created an innovative mechanism for enacting workflow instances using the mechanism of component adaptation. I have also created a basic notation for specifying *component-constraint* based workflow types.

In the system It is possible to form groups, representing virtual organisations, and govern them in the framework. Since its possible to create process types that create new process types, such organisational governance mechanisms can be dynamically evolved. Further the permissions relating to members of any particular group can be dynamically evolved over time, as new process types are created.

However, the VOOP system, in its current state, is limited in a number of ways including:

1. While the components built are sufficient to demonstrate the overall architecture of the system, and its potential, they are currently quite limited in their functionality and user-interface integration. This is largely due to time constraints and the large number of components required to demonstrate governance effectively.

2. The components for creating new process types, and assigning roles in them, are currently too complicated and technical for a user to understand how to use, without some training. They are also currently unable to create some of the types of workflow definition which are possible in the *notation*. Again this is largely due to time limitations. If the VOOP system is to meet the goal of evolvable governance of virtual organisations, it will be necessary for this interface to be much clearer and simpler to use.

8.3 Possible Usage

8.3.1 Groups, Trust, and Experimentation with Processes

VOOP is not yet developed enough to allow normal users to form and dynamically govern virtual organisations. However this is mainly due to limitation of the user-interface for creating processes. While the long-term goal of VOOP is to make this possible, there are possible uses of the platform in its current state. It is the view of the author that the best way to further develop VOOP, is in close collaboration with a number of *trained* users, who understand the process specification language.   Those users could use the platform to experiment with forming organisations and developing governance processes. In other words, VOOP could be used a platform for experimenting with the effects of different types of governance processes, on organisational development. Keser [31] has suggested that virtual organisation research could benefit from an experimental economics laboratory to examine certain design issues for trust and reputation. This has been referred to as a "collaboratory". Since the area of the effects of governance processes on the development of virtual organisations is similarly poorly understood, VOOP could be considered in a similar vein a "collaboratory" for experimenting with governance processes.

Experimenting with the effects of creating seed groups as well as sub-groups with different governance processes is expected to yield results of great interest for the academic pursuit of understanding the dynamics of virtual organisations.

This project does not explicitly deal with the issues of trust and reputation in virtual organisations, though it could be extended to do so. However issues of trust are to some extent implicitly dealt with through the mechanics of group membership. Since each group is able to define the process through which membership is attained, and groups have roles in the processes of other groups, a certain level of trust will be conferred on users on the basis of their membership of a group. VOOP could potentially also be extended to include explicit trust mechanisms. However, this would require extensions to the ontological framework of VOOP.

.

8.3.2 Dynamics of Groups

The types of processes available in the seed group, as well as the roles in those processes will have significant implications for how the system of groups can evolve. For example, if there is no process for creating sub-groups of the seed group, there will never be any more groups. Perhaps a more interesting example, the seed group may or may not have a *type of process* which facilitates the creation of *new* types of processes. If it does not, then the *seed group* will be "stuck" with the types of processes it is created with for the rest of its existence. This can be an advantage if the group wants to be relatively politically neutral. For example the *seed group* in the initial deployment of VOOP will act as a "container" for creating new groups. However a disadvantage might be that the seed group will be less dynamic, so for instance it would not be possible for the members to change the process of creating sub-groups at a later date. Creating the seed group with a type of process which creates new types of processes will allow it to evolve more flexibly, but will make it into a much more politically-driven entity.

Particular issues occur when creating sub-groups, if a sub-group represents a *role* within an *organisation*, such as Consumers or Agents in the Open Food Coop example. In this case, the organisation will define the types of processes initially available to the sub-group and whether or not the group can evolve its own new process types. However it is likely that in some situations, the organisation will want to retain the ability to add extra process types to the sub-group. Here the question is of the relative autonomy of sub-groups. This issue has not yet been properly addressed in VOOP. The question on an organisational level, "do we give a sub-group the ability to evolve its own processes?” is decided on creation of the group. However in a hierarchical organisational form, it is common that a group cannot create new processes for itself, but its *super-group* can. This is not yet possible in VOOP but would require only minor modifications to make possible.

8.4 Future Developments and Research

Since this project is for the Open Coop it will not end with the completion of this paper. Rather, it is hoped that this is the first step in a much more ambitious project.

As already noted, VOOP will need much improvement in its user-interface in order to make the process of governance of virtual organisations understandable to the end-user. However it is not yet known what will be necessary in order to make component-based composition of processes understandable to end-users [14].

Component-based workflow, where the workflow is specified in terms of interfaces and constraints, seems to have some promise. During this project, the author, has experienced firsthand the amount of work involved in creating individual components capable of facilitating governance activities. In order to draw in more resources from the Open Source community it would be a sensible next step to create an API[14] for third-parties to develop their own components. This would both simplify and standardise the process of developing components, making it possible to create a distributed effort to develop suitable components. A repository of components could then be kept, allowing users and groups to pick and choose from available components.

As explained in section 6.7, VOOP has an architecture which can quite easily be extended to create a distributed system, using the Twisted modules, *Perspective Broker* and *Twisted Cred.* Creating a seamless user-interface to groups hosted on multiple servers is a necessary development for scalability of the system.

One additional feature, the need for which has been requested of the author numerous times during this project, is the ability to visually *map* the relation

[14] Application Programming Interface

between groups. This could be done in a distributed fashion with a *mapping service* as briefly mentioned in section 6.7.

Finally, VOOP could benefit significantly from a richer semantic framework, on which to base constraints. Currently VOOP sets constraints against *properties,* which constitute key-value pairs. The same properties are modified by the component. While this has some mileage as an approach, the system would be much more powerful (and arguably more understandable to the user) if there were more advanced ontologies available to the constraints and components. The simplest implementation of this might include an RDF[15] framework of “triplets”. However there are many, more advanced, and interesting ontological frameworks currently under development. In this way VOOP could contribute to the growing research area of ontology-extended workflow [32].

[15] Resource Description Framework

8.5 Conclusion

There are many complex issues surrounding the governance of virtual organisations. The issues are complicated by the need to form organisations in the virtual realm, and the consequent need for dynamic evolution of governance. Analysis of governance in terms of processes seems to be a promising approach and consequently workflow is the appropriate technology with which to approach the problem from a technical perspective. Due to the interactive nature of most governance activities, component-based workflow is the most appropriate type of workflow. However the field of component-based workflow is still relatively young and is currently evolving very quickly. A great deal of research is still required if we are to meet the goal of workflow components which can be understood and composed into processes by a normal user. As outlined in the previous section there are many avenues for further development. The VOOP project will be taken forward by the Open Coop and will continue to evolve and improve.

9 References

[1] N. Szabo, “Formalizing and Securing Relationships on Public Networks,” in *First Monday, Peer Reviewed Journal on the Internet,* [online] Vol.2 No.9 - September 1 1997.

http://www.firstmonday.org/issues/issue2_9/szabo/index.html (Accessed 25 July 2005)

[2] J. Davies-Coates, “Open Organisations,”(Open Coop), [online] 2002, http://open.coop/open+organisation (Accessed 21 June 2005)

[3] The Open Organizations Project, [online] 2004,
http://www.open-organizations.org/ (Accessed 1 June 2005)

[4] International Cooperative Alliance, [online] 2003,
http:// www.ica.coop (Accessed 23 June 2005)

[5] D2 ID 7.1.1 State-of-the-art evaluation, *Trust Com (EU 2004)* TrustCom, Sixth Framework Programme, Networked Businesses and Governments, INTEGRATED PROJECT, EU, P40-42, P438-454 [online] (April 2004)

http://www.eu-trustcom.com/index.php?page=Documentation (Accessed 8 July 2005)

[6] Bultje, René, van Wijk, Jacoliene: Taxonomy of Virtual Organisations, based on definitions, characteristics and typology., in: *VoNet: The Newsletter* [online] Vol. 2 (3) p. 9 1998

http://www.virtual-organization.net/ (Accessed 1 July 2005)

[7] C. Handy, “Trust and the Virtual Organization,” *Harvard Business Review.* 73 (3). May-June 1995

[8] L.M. Camarinha-Matos and A. Abreu, “Towards a foundation for virtual organizations,” in *Proceedings of Business Excellence 2003 – 1st Int. Conference on Performance measures, Benchmarking, and Best Practices in New Economy*, Guimarães, Portugal, 10-13 Jun 2003.

[9] K. Crowston and J.E. Short, *Understanding processes in organizations*. Unpublished manuscript., [online] (1998)

http://crowston.syr.edu/papers/understanding-processes.pdf (Accessed July 18 2005)

[10] L. B. Mohr, *Explaining Organizational Behavior: The Limits and Possibilities of Theory and Research.* San Francisco: Jossey-Bass, 1982.

[11] S.-L.Keoh, E. Lupu and M. Sloman, “PEACE : A Policy-based Establishment of Ad-hoc Communities,” in the *Proceedings of the 20th Annual Computer Security Applications Conference* (ACSAC), Tucson, Arizona, USA, © IEEE Computer Society, pages 386 - 395, December 6 - 10, 2004. http://www.acsac.org/2004/papers/76.pdf (Accessed 19 July 2005)

[12] H. J. Harrington, *Business Process Improvement: The Breakthrough Strategy for Total Quality, Productivity, and Competitiveness.* New York: McGraw-Hill, 1991.

[13] T.H. Davenport, & J.E. Short, “The new industrial engineering: Information technology and business process redesign,” *Sloan Management Review,* 31(4): 11-27, 1990

[14] J. Shi, D. Lee, and E. Kuruku, "Task-Based Modeling Method for Business Process Automation." submitted to *ASCE Journal of Construction Engineering and Management* (2004)

[15] Zhuge, “Component-based workflow systems development Source”, *Decision Support Systems*, Volume 35 , Issue 4, July 2003. Pages: 517 - 53

[16] P. Soffer and Y. Wand, "Goal-driven Analysis of Process Model Validity", *Advanced Information Systems Engineering* (CAiSE'04) (LNCS 3084), p. 521-535, 2004

[17] R.M. Cyert, & J.G. March, J. G., *A Behavioral Theory of the Firm*. New Jersey: Prentice-Hall, 1963

[18] H. A. Simon, “On the concept of organizational goal,” *Administrative Sciences Quarterly*, 9(1):1-22, 1964

[19] K. Crowston and J. Howison, “The social structure of Free and Open Source software development,” *First Monday*, [online] volume 10, number 2 (February 2005), http://firstmonday.org/issues/issue10_2/crowston/index.html (Accessed 1 August 2005)

[20] A. Cox, "Cathedrals, Bazaars and the Town Council," *Slashdot* (13 October) 1998

http://slashdot.org/features/98/10/13/1423253.shtml (Accessed 24 October 2004

[21] C. Gacek, T. Lawrie, and B, Arief, “ The Many Meanings of open source,” *Technical Report 1, DIRC – Interdisciplinary Research Collaboration in Dependability,* 2001.

http://www.dirc.org.uk/publications/techreports/papers/1.pdf (Accessed 24 October 2004)

[22] A. Mockus, R. Fielding, and J. Herbsleb, "Two Case Studies Of Open Source Software Development: Apache And Mozilla," *ACM Transactions on Software Engineering and Methodology*, volume 11, number 3, pp. 309–346. 2002

[23] Workflow Management Facility V1.2, Object Management Group. [online] 2000
http://www.omg.org/docs/formal/00-05-02.pdf (Accessed 4 July 2005)

[24] WFMC Workflow Reference Model, Workflow Management Coalition. [online] 1995
http://www.wfmc.org/standards/docs/tc003v11.pdf (Accessed 31 June 2005)

[25] P. J. Kammer, “Building the Process: Component Based Workflow Architectures in a Distributed World”, *CSCW 2000 Workshop: Beyond Workflow Management: Supporting Dynamic Organizational Process*, Philadelphia, PA, December, 2000.

[26] PyProtocols, Python Enterprise Application Toolkit. [online] 2003
http://peak.telecommunity.com/PyProtocols.html (Accessed 22 July 2005)

[27] Python Enhancement Proposal 246, Python. [online] 2001
http://www.python.org/peps/pep-0246.html (Accessed 25 July 2005)

[28] Twisted, Twisted Matrix Laboratories. [online] 2005
http://twistedmatrix.com/ (Accessed 10 July 2005)

[29] Zope, Zope Community. [online] 2005
http://www.zope.org/ (Accessed 12 July 2005)

[30] Twisted Cred how-to, Twisted Matrix Laboratories. [online] 2005
http://twistedmatrix.com/projects/core/documentation/howto/cred.html

[31] Keser, C., 2000, Strategically planned behavior in public goods experiments, Working Paper, CIRANO, Scientific Series 2000s-35.

[32] Pathak, J., Caragea, D., and Honavar, V. (2004). Ontology-Extended Component-Based Workflows: A Framework for Constructing Complex Workflows from Semantically Heterogeneous Software Components. In: Proceedings of the Workshop on Semantic Web and Databases (SWDB-04). Springer-Verlag Lecture Notes in Computer Science. In press.