ARCHITECTURAL SUPPORT FOR SOCIO-TECHNICAL SYSTEMS

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Osama E. S. Elhassan, M.Sc., D.I.C. (Imperial)

Department of Computer Science

University of Leicester

November 2008

This thesis is dedicated to my parents.

Thanks for always supporting me, no matter which way I choose…

Author’s Declaration

I herby declare that this submission is my own work and that it is the result of work done mainly during the period of registration. To the best of my knowledge, it contains no material previously published or written by another person nor material which to as substantial extent has been accepted for the award of ant other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

Parts of this submission appeared in the following conjoint publications, to each of which I have made substantial contributions:

- El-Hassan, O. and J.L. Fiadeiro, Role-based Architectural Modelling of Socio-Technical Systems, in *Proc. of the 3rd International Workshop on Coordination and Organisation (CoOrg'06)*, Bologna, Italy: ENTCS, 2006, p. 5-17.
- El-Hassan, O., J.L. Fiadeiro, and R. Heckel, Managing Socio-technical Interactions in Healthcare Systems, in *BPM 2007 Workshops, the 1st International Workshop on Process-oriented Information Systems in Healthcare*, H.M. ter Hofstede, B. Benatallah, and H.-Y. Paik, (eds.), *LNCS*, Vol. 4928, Springer: Berlin / Heidelberg, 2008, p. 347-358.

Abstract

Software development paradigms are increasingly stretching their scope from the core technical implementation of required functionalities to include processes and people who interact with the implemented systems. Socio-technical systems reflect such a trend as they incorporate the interactions and processes of their social participants, by treating them not as users but as integral players who enact well-defined roles. However, developers of these systems struggle with their complexity and weak architectural support. The challenge is that existing toolboxes for modelling and implementing complex software systems do not take into account interactions that are not causal, but only biddable (i.e. whose execution cannot be ensured by software). Therefore, models and implementations generated by these toolboxes cannot detect and respond to situations in which the system participants deviate from prescribed behaviour and fail to play the role that they have been assigned as entities of the system.

The research focus is on how a norm-based architectural framework can promote the externalisation of the social dimension that arises in software-intensive systems which exhibit interactions between social components (i.e. people or groups of people) and technical components (devices, computer-based systems and so on) that are critical for the domain in which they operate. This includes building normative models for evolvable and adaptable socio-technical systems to target such interactions in a way that ensures that the required global properties emerge.

The proposed architectural framework is based on a new class of architectural connectors (social laws) that provide mechanisms through which the biddability of human interactions can be taken into account, and the sub-ideal situations that result from the violation of organisational norms can be modelled and acted upon by self-adapting the socio-technical systems.

The framework is equipped with a new method underpinned by a coherent body of concepts and supported by a graph-based formalism in which roles present the structural semantics of the configuration, while the laws have operational semantics given by the graph transformations rules. Guiding methodological steps are given to support the identification of critical social interactions and the implementation of the proposed method.

Case studies derive the evaluation of the approach to demonstrate its generality, applicability, flexibility and maintainability.

Table of Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, all praises and thanks go to *Allah,* who took me through this mission and help me to accomplish it. I also thank all people without whom this thesis would not be possible. My sincere thanks go to my supervisor, Professor José Luiz Fiadeiro, for his guidance throughout my Ph.D. trajectory and for his wise and acute observations on how to improve my work. Those many hours we spent discussing my research topic are unforgettable. My warmest thanks also go to Professor Reiko Heckel for his *visually* compelling and *mathematically* rigorous co-supervision that inspired deep discussions throughout my graduate study. I express special thanks to all the members of the Computer Science Department at the University of Leicester, notably Professor Rick Thomas, Dr. Roy Crole, Dr. Mohamed El-Ramly, Dr. Stephan Reiff-Marganiec and to the PhD tutor Dr. Fer-Jan Vries. I gratefully acknowledge thoughtful comments and encouragements from Dr. Michel Wermelinger and Dr. Jon Hall from the Open University. I convey my gratitude to my PhD colleagues, Cristóvão Oliveira, O'Neil Delpratt and João Abreu.

I owe my parents an ocean of gratefulness for their continuous moral support and prayers. Definitely, I could not have finished this work without their help. I would like to express my appreciation to many special people who contributed to my work with their continuous encouragement. Among them, I wish to thank Professor Izzeldin Osman, Dr. Mohamed Khidir, Dr. Ali Abbas, Khalid Abdul-Aziz, Dr. Mohamed Taye, Dr. Ali Al-Bayati, Dr. Ajlan Al-Ajlan, Dr. Hassan Al-Marzuoki, Dr. Mohamed Manan, Mohamed Al-Otaibi, Nasir Al-Alwan, Yosif Al-Saeed, Ali Al-Qahtani and finally Dr. Ahmed Al-Ghamdi who deserves special thanks. Without question and beyond any doubt, their moral support and suggestions were of great value.

Chapter 1

Introduction

"We can start with the obvious statement that engineering is a problem solving activity".
Walter G. Vincenti

1.1 Engineering Socio-technical Systems

In spite of the increasing degree of automation across all sectors of the economy, people will and must remain as integral players in all sorts of large-scale heterogeneous systems that control critical infrastructures (defence, energy, health, telecommunication, transport, etc.) or ensure services that are essential for the functioning of the society (e-government, e-learning, etc.). To emphasise the fact that the behaviour of such systems depends on interactions between humans and technical components, the term "socio-technical systems" was coined (Emery & Trist 1960). The term has been extended to refer to systems that incorporate a "social" dimension in the sense that people (or groups of people) need to be considered, not as external users, but as another class of components, together with software and devices (Brier, Rapanotti & Hall 2004, Bryle & Giorgini 2006). As such, software is used to ensure that both technical components (e.g. devices, software applications and artefacts) and social components act and interact jointly within ad hoc and changing configurations in ways that are sensitive to the needs of the society or the economy.

One of the problems that need to overcome to support the development of socio-technical systems is the fact that current software engineering methods and techniques create a boundary around the technical components and place humans outside that boundary as users, not as players on an equal par with the technical components. This is particularly important as these boundaries between social and software components may vary throughout a system's lifetime; tasks performed by humans can be partially replaced or even shared with software applications, depending on the context of execution. Therefore, interactions between people (as social components) and technical components need to be brought inside the system, which is being made easier thanks to recent advances in monitoring techniques and context-awareness technologies such as sensor network applications (Heinzelman, Murphy *et al.* 2004), wearable systems (Drugge, Hallberg *et al.* 2006) and RFID technologies (Holzinger, Schwaberger & Weitlaner 2005).

One of the challenges that need to be overcome for addressing this social dimension of systems is the fact that interactions involving humans are not necessarily causal, i.e. people cannot be guaranteed to bring about changes that may be required to ensure the correct behaviour of the whole system. In other words, social components cannot be designed, as software and mechanical/hardware entities can, to comply with system rules; instead, they constitute what Michael Jackson calls biddable domains: "they can be enjoined to adhere to a certain behaviour, but may or may not obey the injunction". That is, social components may deviate from prescribed behaviour or *codes of norm,* and perform interactions that lead to sub-optimal (or sub-ideal) states. Such deviations are not necessarily "faults" in the sense that they are deliberate or malicious, but they may arise from the fact that the context in which a system is operating changes, which may imply that the humans involved in the system may need to operate outside the role that they have been ascribed and, as a consequence, violate a number of norms.

In such circumstances, software cannot force the social components to change their behaviour, but the system should be able to reconfigure itself in order to adapt to a new operating context. For instance, under normal circumstances, the software that is controlling a routine check-up will prevent a nurse from operating some kinds of devices but, if an emergency is detected, the software should adapt to the new role that the nurse is required to perform in, say, a life-critical operation, by withdrawing some of those restrictions and providing information that a doctor would normally know (or have access to) like an allergy to penicillin.

In summary, there is a need for new methods and techniques that can support the engineering of systems in which interactions can be causal or biddable and that can respond to situations in which system participants deviate from prescribed behaviour and/or act (possibly by necessity) outside the role that they have been assigned.

1.2 Aims & Objectives

The aim of this research is to put forward an engineering method for socio-technical systems that addresses social entities not as external users but as integral role players whose interactions with technical components (i.e. software and devices), although governed by organisational rules and policies, may affect the whole system behaviour in ways that cannot be totally predicted, let alone programmed. This method should be able to handle situations in which social components act outside their permitted role scope so as to allow the system to reconfigure itself in order to adapt to a new operating context in ways that ensure agreed, possibly minimal, levels of service.

Towards this aim, the following objectives are pursued:

- **Concepts**: modelling primitives need to be introduced through which normative concepts (e.g. permissions obligations and power) and organisational concepts (e.g. roles, tasks and operations) can be expressed and socio-technical protocols (i.e. protocols governing interactions between social and technical components) can be modelled.
- **Conceptual models**: a new framework is required to capture the structural and behavioural aspects of the introduced concepts. Conceptual models are patterns of interconnected structural elements. The framework needs to separate social aspects from the technical ones and capture normative (or ideal) as well as sub-ideal situations.
- **Meta model**: a new generic reconfiguration language is required through which the new modelling primitives can be used and a new social interaction-aware level of reconfigurability (i.e. norm-based self-adaptivity) can be supported. A meta model needs to address the interconnections between the technical and social levels of the conceptual model at a high level of abstraction.
- **Tools**: the new reconfiguration language needs to be supported by tools that can help modellers specify and animate models of socio-technical protocols and the way they self-adapt to handle violations and sub-ideal situations.

1.3 Methodology & Approach

1.3.1 Research Methodology

This thesis is not application-driven but rather directed to foundational issues of a relatively new area of research that needs to be equipped with new concepts, abstractions and mechanisms. Several research paradigms are related or relevant to its aims, though none is capable of addressing their full extent. Although a more exhaustive review of the state of the art is left for other chapters, it is useful to mention three particular areas as a justification for the research methodology that I have adopted.

One of the most prominent class of approaches in this broad area of research adopts agent-oriented methodologies, which have been extensively used for modelling social and organisational structures (e.g. Yao, Moody & Bacon 2001, Dignum, Meyer *et al.* 2002, Nickles, Rovatsos & Weiß 2002, Zambonelli, Jennings *et al.* 2003, Dignum, Vázquez-Salceda *et al.* 2005). However, as further discussed in Chapter 2, such approaches are not really appropriate to meet the objectives of the thesis because they focus mainly on modelling and capturing the *autonomy* of agents that act as owners of roles in order to pursue their own goals (individually or in cooperation with others). Thus, within agent-oriented methodologies, the way agents collaborate with each other and self-adapt to environment changes is hardwired in the agent code. This is why I decided to lean more towards software engineering methodology, namely recent modelling techniques for software architecture that support the design and implementation of an interaction-centric approach in a more explicit and less intrusive way.

A related software engineering approach that I considered is Jackson's Problem Frames, which supports problem analysis and decomposition, i.e. the identification of components, their interconnections, their assumptions about each other and the way they relate to the problem domain. However, the aims that I am pursuing target a kind of dynamic requirements that are different from those captured in Problem Frames: they prevail only when sub-ideal situations and violating behaviours of social components are detected. Therefore, socio-technical systems require analysis and decomposition techniques that support these different levels of ideality directly.

Another relevant area of software engineering is requirements modelling. However, methodologies such as the *i** framework for enterprise modelling (Yu & Mylopoulos 1997) and the Soft System methodology (Checkland 1984) consider social components as entities (users) outside the boundaries of the system, i.e. as part of the environment, which does not make them really suitable for my purpose.

1.3.2 The Approach

As discussed above, I ended up adopting elements of architectural modelling approaches within software engineering as a means of handling non-normative or sub-ideal situations in socio-technical systems as first-class concerns. The main reason for this is that, in order to achieve the levels of adaptability motivated above, procedures dealing with these situations cannot be buried in the code of technical components. Otherwise, it would be impossible to figure out, within the code of a component, what is implementing its functionality and what is handling violation of organisational norms. This separation is essential because recovery procedures that handle sub-ideal situations always depend on the role of the social entities that play roles within the larger system. Therefore, these aspects need to be modelled and controlled separately.

In summary, in order to meet the objectives of this thesis, I bring to bear a number of concepts, formalisms and modelling techniques developed in different areas of computer science and software engineering. In particular, the research work builds on:

- Software architecture methodology, namely the 3Cs business architectural approach (Wermelinger 1999, Andrade, Fiadeiro & Wermelinger 2001, Andrade, Fiadeiro *et al.* 2002, Andrade & Fiadeiro 2003). The 3Cs stands for the core concept of separation between *Computation, Coordination* and *Configuration* concerns. The 3Cs architecture provides the required technical level of the proposed model.
- The graph-based approach to model-based transformation of (Karsai & Sztipanovits 1999, Engels, Heckel & Sauer 2000, Wermelinger, Lopes & Fiadeiro 2001), which I adopted for achieving two major purposes:
    - The integration of the 3Cs causal modelling primitives with the dynamic reconfiguration techniques that support adaptation;
    - The provision of a visual semantics of the proposed reconfiguration language through graph transformation rules that can manipulate instances of structural elements (i.e. roles, tasks and technical components) defined in the architectural framework.

- Social studies and theories of collaborative agency such as speech acts (Searle 2002), normative positions (Sergot 1999), behavioural implicit communication (BIC) (Castelfranchi & Giardini 2003) and the overhelp concept of adjustable autonomy (Falcone & Castelfranchi 2000).
- Jackson's Problem Frames approach to problem analysis and decomposition (Jackson 2001) as a means of capturing patterns of real-world problems and normative or ideal behaviour that enforces a requirement. In particular, I build on the extension that has already been defined for socio-technical systems (Brier, Rapanotti *et al.* 2004).
- Deontic formalisms such as the ones developed by M. Sergot and colleagues for normative positions and analysis (Sergot 1998, Liu, Sun *et al.* 2001a, Lomuscio & Sergot 2003b) as a means of addressing notions of sub-ideality.
- Organisation structures and policies, e.g. Role Based Access Control (RBAC) (Sandhu, Coyne et al. 1996b, Moffett & Lupu 1999) and the Ponder policy language (Damianou, Dulay *et al.* 2001).
- The Human Interaction Management method (HIM) as a starting point for methodological steps towards modelling interactions of human-driven processes within organisations (Harrison-Broninski 2005).
- Agent-based approaches for modelling collaborations within social and organisational structures (Finin, Labrou & Mayfield 1995, Barbuceanu, Gray & Mankovski 1999, Weiß, Rovatsos & Nickles 2003, Zambonelli, Jennings & Wooldridge 2003, Dignum, Vázquez-Salceda & Dignum 2005).

1.4 Contributions

The contribution made by the work reported in the thesis can be summarised as follows.

1. The thesis puts forward an architectural method together with modelling primitives that captures socio-technical protocols. The method integrates techniques imported from architectural description languages (namely the 3Cs conceptual model) for the technical side of systems and newly introduced concepts that support the separation of control between *causal interactions* management and *social interaction* management. Based on this

conceptual differentiation, a new reconfigurability mechanism has been put forward to respond to biddable human interactions within a socio-technical protocol. Within the proposed architectural method, biddable social interactions are taken into account as they are:

   a. anchored on *social roles*, which model tasks, permission and capabilities.
   b. captured by *social laws* (an extension of architectural connectors that can handle system response to unexpected interactions or contextual changes), which take into consideration capabilities and permissions as well as the context ideality;
   c. governed through reconfiguration rules that ensure self-adaptivity.

2. The proposed method is supported by graph-based formalisation of reconfiguration operations in which the social roles have structural semantics, while the laws have operational semantics given by the graph transformations. These transformations are based on a meta model. Modellers can exploit domain-independent abstract syntax and transformation rules presented in this thesis, under the AGG tool support, to model and animate social-technical protocol as a means of supporting self-adaptation specifications.
3. The proposed method is supported by methodological steps to locate parts of the socio-technical system that may involve sub-ideal situations, which are then dealt with through social laws.
4. A new level of self-adaptivity is achieved that ensures that once a violation or sub-ideal situation is detected, the system will adapt to enable the congruence[1] of the enacted task (i.e. the required recovery task) to the current operation condition (i.e. available roles and technical resources), or to impose sanctions on the role player who disobeys the obliged task enactment.
5. A generic characterisation of a self-adaptivity manager (the harmoniser), which extends the 3Cs configuration manager to support collaborative aspects between social and technical components. An harmoniser is a

[1] The degree of congruence corresponds to the degree of “fit” between organisational structures and properties of the task at hand and/or environment (Donaldson 2001). Donaldson, L., The Contingency Theory of Organisations, Foundations for Organisational Science, Sage, 2001.

software application that monitors the balance between unexpected interactions and current role entitlements. It interprets the operational semantics of triggered social laws (through the graph-based model) in order to effectuate the empowerment aspects of social interactions or the imposition of sanctions when appropriate.

6. An extension to the work of Brier at al. (Brier, Rapanotti & Hall 2006) is also promoted by this thesis. The original work successfully incorporates Problem Frames in representing human knowledge and guiding development of real-world socio-technical system. The proposed extension introduces a uniform model through which designers can represent capability-based role concepts and their technology-oriented view of tasks.

The proposed approach has been demonstrated and evaluated through peer-reviewing in different research communities e.g. (El-Hassan & Fiadeiro, 2007) and (El-Hassan et al., 2008). Case studies have been developed to illustrate how the new primitives address sub-ideal situations, manage human biddability and guide the system reconfigurations to self-adapt to changes of context.

1.5 Organisation of the Thesis

Due to the multidisciplinary nature of this research work, I do not present a dedicated chapter for literature review. Alternatively, I decided to dedicate a literature review section in certain chapters for a background review. Moreover, an introductory section is dedicated in certain chapters were background knowledge is relevant. An introductory section is included to specify how they relate to the mainstream of the thesis. The organisation of the rest of this thesis is depicted in Figure 1.1 and Figure 1.2 as follows.

Chapter 2 surveys throughout different paradigms of information systems to correlate research work in different disciplines and to illustrate the terminology related to human interactions in various domains. Chapter 3 introduces the software architecture paradigm and thoroughly describes the 3Cs approach upon which this approach relies to build the proposed architectural primitives and reconfiguration mechanisms. Chapter 4 describes in detail the approach adopted for managing and reasoning about biddable interactions: the ternary of norms, roles and reconfigurations. The chapter focuses in particular on the notion of role across several crosscutting concerns, either non-technical (organisation theory, communication theory and social sciences) or technical (object-oriented modelling, agents modelling, and access based policies). I also present a model that aims at providing a method to model biddability with organisational contexts while prescribing a norm-based control method to preserve the overall good behaviour of the system, particularly in sub-ideal situations.

Chapter 5 explains our methodological approach and high-level concepts that have been advocated through this thesis particularly the abstractions related to biddable interactions and norms analysis. Chapter 6 includes the graph-based mathematical formulation of the proposed concepts such as providing operational semantics for reconfiguration operations and role transitions. Additionally, it touches on practical implementation issues such as tool support. Chapter 7 is dedicated to evaluating the proposed architectural approach and its features against the objectives of this thesis using a number of case studies.

Finally, Chapter 8 summarises the lessons learned from and identifies potential issues to be further developed. The glossary at the end of the thesis provides key of the terminology used throughout the thesis. The Appendices include a list of abbreviations and a mapping from the proposed extension to the 3Cs architectural approach to the conceptual framework of the planned behaviour theory (PBT).

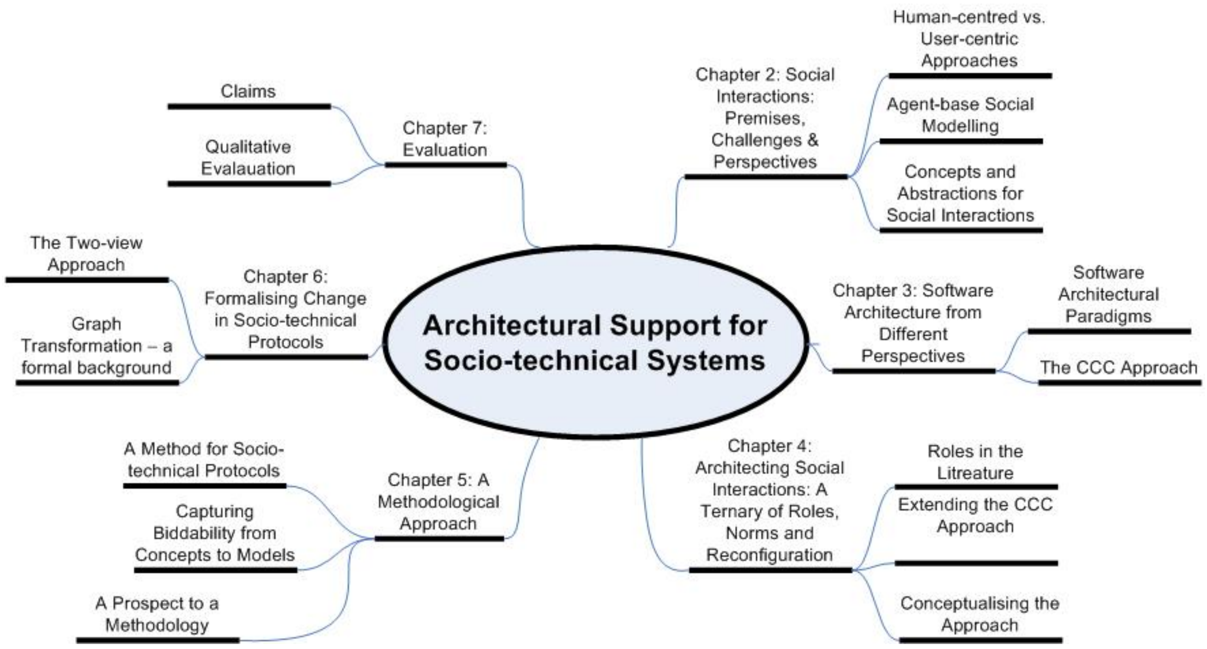
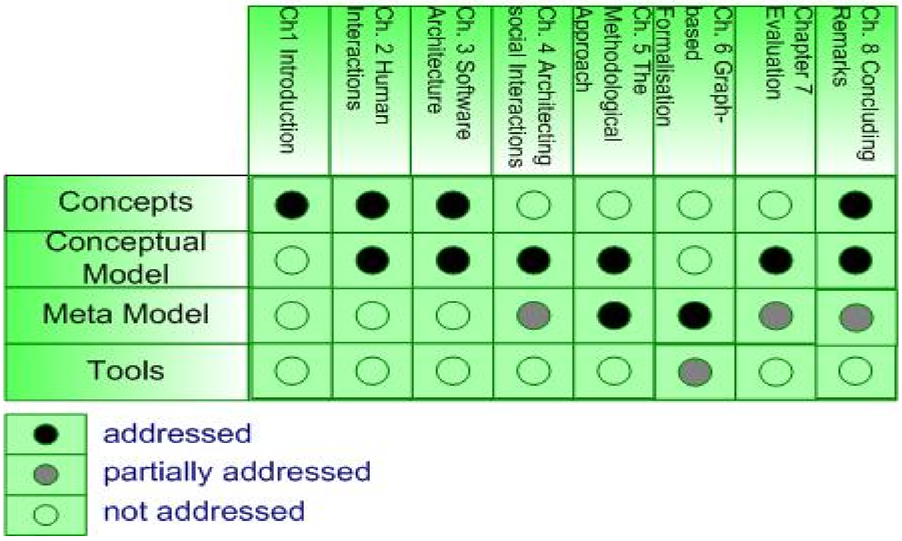

Architectural Support for Socio-technical Systems
Chapter 2: Social Interactions: Premises, Challenges & Perspectives
Human-centred vs. User-centric Approaches
Agent-base Social Modelling
Concepts and Abstractions for Social Interactions
Chapter 3: Software Architecture from Different Perspectives
Software Architectural Paradigms
The CCC Approach
Chapter 4: Architecting Social Interactions: A Ternary of Roles, Norms and Reconfiguration
Roles in the Litreature
Extending the CCC Approach
Conceptualising the Approach
Chapter 5: A Methodological Approach
A Method for Socio-technical Protocols
Capturing Biddability from Concepts to Models
A Prospect to a Methodology
Chapter 6: Formalising Change in Socio-technical Protocols
The Two-view Approach
Graph Transformation – a formal background
Chapter 7: Evaluation
Claims
Qualitative Evalauation

Figure 1.1 Topics covered in the thesis chapters



	Ch1 Introduction	Ch. 2 Human Interactions	Ch. 3 Software Architecture	Ch. 4 Architecting social Interactions	Ch. 5 The Methodological Approach	Ch. 6 Graph-based Formalisation	Chapter 7 Evaluation	Ch. 8 Concluding Remarks
Concepts	●	●	●	○	○	○	○	●
Conceptual Model	○	●	●	●	●	○	●	●
Meta Model	○	○	○	◍	●	●	◍	◍
Tools	○	○	○	○	○	◍	○	○

●	addressed
◍	partially addressed
○	not addressed

Figure 1.2 Modelling and meta-modelling the conceptual framework

Chapter 2

Social Interactions: Premises, Challenges and Perspectives

2.1 Overview

I address a multi-perspective view of social interactions within "socio-technical systems": systems that include a "social dimension" in the sense that people (or groups of people) need to be considered not as external users but as another class of components that, together with software and devices, perform roles that are vital for the "good" behaviour of the system. In order to bring human interaction within the boundaries of systems, normative concepts such as permissions, obligations and powers should be referred to, in order to model violations that can take place so as processes and underlying software can be reconfigured to react to non-normative situations in ways that ensure agreed, possibly minimal, levels of service.

2.1.1 Objectives

The goal of this chapter is to review the engineering principles, social concepts and computer science underpinning collaborations, including the interactions of human components. This review highlights the concepts and abstractions that contribute to a "dual-view" model, which balances coordinated technical interactions with the biddable nature of social entities on which the success of the system depend, particularly at sub-ideal contexts. Therefore, human interactions are required to be incorporated within the system's boundaries.

In this vision, humans are partners of a system, and therefore, what is required is neither addressing the *user-centric* view of social components (i.e. improving the way humans interact with software), nor designing technical artefacts to fit around models of human intentions, skills and creativity (i.e. *human-centred view*), but rather defining ways through which software can *orchestrate* interactions between humans and technical components, so as to guarantee optimal or sub-optimal responses to dynamically changing environments.

2.1.2 Social Interactions

The key role of social interactions in achieving the dependability and evolvability of socio-technical systems has been emphasised by several recent research articles, e.g. (Felici 2003, Hall & Rapanotti 2005, Bryle & Giorgini 2006). They exhibit the social dimension as a first-class concern, which can be perceived as coupled systems whose performance depends on the *interactions* of humans together with technical information systems. One of the main sources of difficulty in socio-technical systems, which are found in various application domains, is the fact that human participants may deviate from prescribed social or organisational *norms*. Such deviations are not "errors" but, rather, result from the fact that situations may arise in which humans may need to interact with machines in "sub-ideal" states.

Sheridan summarized the above mentioned fact in (Sheridan, Corker & Nadler 2006): "Technology has become much more capable of performing sensing, decision, communication and action functions in comparison to humans. Humans are slower, less accurate and less predictable. Yet under off-normal and unanticipated circumstances, machines can look stupid and humans are invaluable in perceiving complex patterns of information, making complex decisions based on probabilistic data and value judgments, and improvising to recover from otherwise disastrous situations".

Analogous findings have already been identified and expressed in distinct yet related research areas, e.g. ad hoc changes in medical business processes (Lenz & Reichert 2007), workflow changes (van der Aalst & Jablonski 2000), and ad hoc resource management (Russel, van der Aalst *et al.* 2005). An agent-based framework introduces *patterns overhelp* — identifying the way that an *agent* can help to solve problems—to support reasoning about such situations (Falcone & Castelfranchi 2000). Additionally, a new paradigm in the business process community has recently emerged, namely *human-driven processes* (Harrison-Broninski 2005) in which these processes are distinguished from *mechanistic* ones. This is a sort of distinction that is similar to my view in the sense of the differentiation between causal and non-causal interactions in terms of models and control mechanisms.

The common factor between these research studies is the call for some sort of freedom of reaction to be ascribed to human participants, thereby contributing to system flexibility and dependability. Freedom to (re)act matches perfectly the characteristics of *biddable domains* as identified by Jackson (Jackson 2001): "[people] can be enjoined to adhere to certain behaviour, but may or may not obey the injunction". The problem, as highlighted in the quote above, is how to endow systems with a degree of flexibility that allows them to adapt dynamically to changes from ideal to sub-ideal states.

Methodological approaches available for modelling interactions between software and other technical components do not generalise to social components. This is because interactions with technical components are causal in the sense that technical components perform designated actions in reaction to triggers issued by software components, whereas interactions with social components are only biddable.

The argument extends to techniques that are used for modelling business processes and workflows in organisations; most of the time they are based on causal models and fail to take into account the fact that humans reaction(s) cannot be programmed or hardwired. Workflows tend to be implemented in ways that are too rigid to sustain interactions with people, leading to fatal incompatibilities with human forms of interaction, which themselves derive from more relaxed and non-causal if not opportunistic behaviour. These types of behaviour are of great importance should they be called upon to support the recovery of a system from sub-ideal situations. Software developers are in need of modelling social components which in turn require a new set of behavioural models for their collaborations with both software and hardware components (Fiadeiro 2007).

Humans participating in systems, unlike agents, are not fully independent "agents" as they are expected to follow norms, and their set of possible interactions are *projected* to the set of monitored actions with regards to three important factors:

- The existing and available configurable technical elements: software, hardware and business entities.

- The set of existing rules that causally governs interaction between technical entities.
- Responsibilities — in terms of permissions, obligations and interdictions — that are conferred upon human participants.

For example, human participants who are both willing and capable of over-helping a system — in the sense of (Falcone & Castelfranchi 2000) — may or should be exploited to alleviate sub-ideal situations. A doctor should be empowered to violate his/her assigned role by accessing a patient's medical record and performing operations—even if s(he) is not authorised to do so —when this patient is in a life-threatening situation.

2.1.3 The Chapter's Structure

This thesis commences by introducing in Section 2.2 a survey on existing approaches towards incorporating social interactions within systems whether user-centric or human-centred. Section 2.3 reviews the research in social interaction modelling within the field of agent-based modelling. Section 2.4 presents concepts and abstractions that are deemed to be valid for capturing the particularities of social interactions for the sake of reasoning about them and supporting system participants in different contexts. Section 2.5 examines the capacity of existing approaches with regards to contextualising interactions.

2.2 User-centric and Human-centred Approaches

Tackling the issue of the dependable design and implementation of socio-technical systems that operate within organisational environments requires novel ways of thinking about the interconnections and interactions between social systems and technical ones. Martin and Somerville argued that despite the fact that the structures of these systems cannot perfectly be composed into a single form in a model (or series of models), a structural approach still allows us to create intuitive, more fundamental connections between them (Martin & Somerville 2006).

The meditation between the field of study of social structures in general and human interactions in particular, on the one hand, and the conceptual and practical modelling of technical systems, on the other hand, still gains the interest of researchers in social studies (Akrich 1995, Sutcliffe 2000, Hall & Rapanotti 2005, Dobson & Martin 2006, Coiera 2007), in software engineering (Liu 2000, Ghezzi & Picco 2002, Cebulla 2004, Lock 2004, Hall & Rapanotti 2005, Bryle & Giorgini 2006, El-Hassan & Fiadeiro 2006), and in knowledge domains (Coakes 2002, Reddy, Pratt *et al.* 2003). Most of these studies were either biased to putting the human at the centre of the modelling approach — forcing the design and adaptation of technological aspects to their needs — (e.g. *human-centred* approaches), or *technology-centric* approaches that give priority to the design of technology or take it for granted, and constrain user interactions accordingly. In the latter approach users are considered extrinsic entities whose impact is overlooked; this is often revealed as ill-conceived by usability models.

Human-centredness is a design approach to information systems (IS) that gives the main concern to humans and their interactions with systems over any technological aspects. (Gill 1991) defines human-centredness as "a new technological tradition, which places human need, skill, creativity and potentiality at the centre of the activities of technological systems". The human-centred approach to the design of technology emerged as an answer to the identified deficiency of traditional approaches to software development which *deskill* technology users and fail to take into account the rich human qualities of working environments (Gill 1991).

The perspective in this approach is to develop a language of software interactions that lies hidden behind the boundaries of "user" interactions with computers. A design that is built around how users overtly interact with systems is limited due to the focus on the technology rather than underpinning how that technology supports the system users in their work. The key concept in this approach is how to enrich *interaction design*: a paradigm that addresses the ways in which people collaborate with a technical artefact, and designs artefacts in a way that reflects the purposes of these collaborations. A definition of the interaction design term was coined by (Winograd 1994): "My own perspective is that we need to develop a language of software interaction—a way of framing problems and making distinctions that can orient the designer [...]. There is an emerging body of concepts and distinctions that can be used to transcend the specifics of any interface and reveal the *space of possibilities* in which it represents one point."

Conversely, within user-centric systems, user interactions appear to be limited by the tradition of Human-Computer Interactions (HCI) techniques (Jacko & Sears 2003), which investigate how a single user might use a *predefined* technical artefact (i.e. software or hardware component) to determine how to design the artefact to be usable. These techniques are inadequate for designers considering aspects of role-dynamicity, context-ideality and norms significance that would make the system more "human-centred."

Human-centredness thus constitutes a feasible solution to the weaknesses of user-centric approaches and their underlying usability-driven HCI techniques. This research aims for a departure from the trend of user-centric approaches as it reduces "the world of possibilities" ascribed to a human participant merely to interactions with a computer-based system, putting aside interactions with machines and hardware components that could be well-monitored by the system itself. This limitation is a result of the IS-orientation of both user-centric and human-centred approaches.

The IS perspective invokes a view of human *agency*—the capacity for human beings to make choices and to impose those choices on the world—that reduces human-centredness to only those considerations required to model individual interactions with a computer-system. This avoids considerations relating to violations, over-helping, and most importantly the role of IT configuration in enabling or constraining organisational processes.

These aspects are the ones that realise the dimension of collaboration that the presented approach aims for, and thus, it should be able to provided abstraction and mechanisms at runtime to allow system participants to take initiatives and exert their biddability when called upon to respond to emergencies. This sort of opportunistic behaviour can be achieved through adjusting their normative state (i.e. permissions and obligations conferred to them at runtime) and providing smart monitoring mechanisms that manage the system configuration and guarantee both context-awareness and norm-based interaction management. The next section will shed light on the approaches, frameworks and techniques that have emerged in the area of agent-based contextualised social interactions.

2.3 Agent-Based Social Modelling

The agent-oriented research to represent social models has shifted towards role-based collaborative frameworks instead of agent-based ones. Castelfranchi (Castelfranchi 2003) justified this move by arguing that despite the alleged autonomy of agents, they are restricted by responsibilities and obligations, which can be perceived as norms. Additionally, social activities that determine role requirements are relatively stable whereas the enactment of agents to roles may change rapidly.

Pacheco et al. (Pacheco & Carmo 2003) provided a norm-based view of organisational modelling that promotes many important concepts e.g. collective agency and role autonomy. Their view of deontic logic is standard in terms of operator's inter-definability but lacks the flexibility of describing norms deviation.

The OperA Framework, developed by (Dignum 2003, Dignum, Meyer *et al.* 2003), devises a concrete approach to support the specification of a multi-agent system. This approach distinguishes between the various mechanisms through which the structure and the global behaviour of the model is described and coordinated. This conceptual model provides both formal semantics that make verification possible and a methodology for domain directed development. The major achievement of this approach, which I intend to incorporate into the promoted methodological approach to capture human interaction within organisational settings, is the provision of an explicit separation between the design of the organisational components (i.e. the roles) and the active entities that animate those components (i.e. the human participants or role players).

(Broersen, Dignum *et al.* 2004) extend this framework to model time-sensitive obligations (i.e. deadlines), by proposing a combinatory form of contracts that uses multi-modal logic with dynamic, temporal and deontic operators. This extension facilitates representing deontic forms with temporal operators (e.g. *O(p) until q)*. The combination of dynamic and deontic logic, the considerations of adding temporal operators, and the development of a formal action language were discussed earlier in (Meyer 1988, Dignum & Kuiper 1997).

A similar approach has been proposed by (Weiß, Rovatsos *et al.* 2003), namely, RNS approach which stands for Roles, Norms and Sanctions. It provides a formal schema for specifying boundaries of autonomous agent behaviour, which consist of roles, norms and sanctions. This approach has several interesting parallels with the promoted understanding of norms and how to utilise them since they centred their idea around agent autonomy and norm deviation management by providing roles space and positive and negative sanctions respectively. Moreover, this approach provides a GUI to support dynamic norms validation. The key distinctions between the proposed approach and the RNS are summarised in the following table (Table 2.1).

Issue	**RNS**	**The proposed architectural approach**
First-class citizen(s)	Agents, roles, norms and sanctions	Social norms, laws and tasks
Purpose	Modelling and capturing the *autonomy* of agents, who act as owner of roles in order to puruse goals, and its impact on the agency of agent behaviour	Modelling collaboration dependencies by capturing the consequences of biddable social interactions putting into consideration the capabilities and the qualification of the system participants and the context in which these interactions take place.
Accessibility	Stakeholders during development and XML enabled computational agents at runtime	Stakeholder and system specifiers during the development and the 3Cs configuration manager at runtime
Role structure	No explicit role-role relationships	Explicit hierarchical role-role relationships
Context sensitivity	Included as preconditions for norm activation and the feedback of targeted agent's obedience to the activated norm	Explicit speech-act-like coupling of monitored interaction and the captured sub-ideal context which can be hierarchically organised
Inter-norm relationships	Chaining is allowed	Chaining is not allowed
Normative impact of events	Request events incur obligations to be activated	Protocol type specification that allow contextualising the captured violations

Table 2.1 RNS vs. the proposed architectural approach

Ricci (Ricci 2002, Ricci 2004) has anchored his TuCSon agent coordination framework on the role enactment operations, which provides contextualized agent-role enactment procedures. His architecture devises a separate construct named (ACC) *Agent*

Coordination Context that allows the agent to perceive the space where they act and interact. ACC is a protocol to allow engineers to encapsulate rules for governing applications built as agents systems, and to mediate the interactions amongst agents. The result is a systematic means of changing the global application behaviour.

2.4 Concepts and Abstractions for Social Interactions

The focus on the proposed approach is on concepts and models that can capture collaborative and cooperative phenomena in which elements self-adapt to each other and the environment. Such adaptations should be *purposive* with regard to the well defined tasks and roles of an organisation, as opposed to *emergent* or ad hoc adaptation processes, and this should allow us to reason about emergent properties resulting from their interactions and interconnections. The focus herein is to model exactly *when* and *how* the participants will go about their assigned tasks because this is a difficult task; sometimes their interactions depend on factors that cannot be captured by modelling. What the approach aims for is to know whether their participants' interactions match their ascribed set of permissions, inherited capabilities and imposed directed obligations, or not. Reaching this goal is vital for reasoning about them and thus allowing the system to respond in a way that preserves the overall good behaviour of the system.

I represent herein a context-capturing framework (Problem Frames), a formalism for specifying normative relations (deontic logic), and a communicative framework (Behavioural Implicit Communication).

2.4.1 Problem Frames

Problem Frames are generic problem types that capture structures and relationships between various types of domains and system elements. Together they constitute a problem-decomposition approach that has received a great deal of attention in software engineering research as it excels in problem analysis, requirements decomposition and specification. In a similar vein, Michael Jackson exploited his understanding of the philosophy of phenomenology to relate intuitively requirements together with both *domains* and *software machines*, which interact in a certain context to achieve certain requirements (Jackson 1995, Jackson 2001).

Software machines are computations that usually reside in some hardware medium and interact with a set of domains, which are wrapped all together in a Problem Frame *context*. A problem context provides us with a view of collaboration rather than presenting a function. The requirements of this collaboration are expressed in terms of the context rather than the machine itself. The machine should be connected to each presented domain, posing some shared (connection) domain in which both the machine and the domain are involved (i.e. shared phenomena).

A simple problem frame, as depicted in Figure 1, is represented typically by a context diagram showing one machine, one domain, and the shared phenomena between them. The Problem Domain is that part of the world in which those effects are perceived by the customer. The Requirements are the properties that the customer wants to observe in the Problem Domain, through the shared phenomena *b*, as a result of the effects brought about by the software as it executes and interacts with the domain via the shared phenomena *a*.

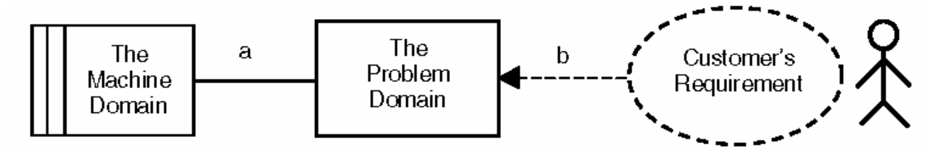


Figure 2.1. A simple Problem Frame

Problem domains can be classified into three categories according to their inherent nature of dynamicity:

- Inert — no action can be generated on its own accord
- Active — actions can be performed on their own accord
    - Autonomous: actions are uncontrollable
    - Programmable: actions can be enforced
    - Biddable: actions can be suggested
- Reactive — actions can be generated in response to external stimuli

This research shows interests in biddable domains as identified in (Jackson 2001): "[people] can be enjoined to adhere to certain behaviour, but may or may not obey the injunction". The reason for this is that their characteristics match social entities' freedom to re(act) within the premises of organisational norms (i.e. instructions, manuals, processes, etc.) to respond to a changing environment or normative state (i.e. obligations and permissions).

However, the original Problem Frames requirement analysis model has kept the traditional view of the separation between the machine *M* and the general description of the environment *W*, burying humans' interactions into the presupposed specification of the system, which contains HCI specifications among others. Figure 2.2, which is borrowed from (Hall & Rapanotti 2005), illustrates a two-ellipse requirement analysis model that contains: the environment description *W*; the statement of requirements *R*; the specification *S* that maintains the fitting interface between the problem and its solution; and the program *P* that resides in the machine and implements the specification *S*.

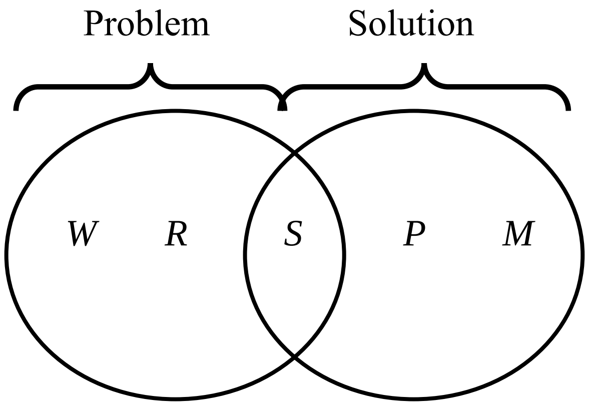


Figure 2.2 Problem Frames' Requirements Analysis Model

Hall and Rapanotti developed an extension that caters for complex socio-technical systems, and they introduced their three-ellipse model as shown in Figure 2.3. They allow for separately describing the instructions that humans are supposed to obey in order to achieve the ultimate goals of a system in general. They introduced a modelling ellipse for human modelling, consisting of *H* and his/her knowledge *K*, which in turn allow other important areas to emerge *UI* and *I*, user interface and instructions, respectively. Hall and his colleagues (Brier, Rapanotti *et al.* 2004, Hall & Rapanotti 2005) concretised their approach by introducing a knowledge domain to be presented in the solution space, thereby maximising *K* — to bring in domain knowledge as design — and minimising *H* (i.e. making a departure from *agentifing* human representation). Their objective was to separate the description of the world from the social components that are subject of design.

I argue that their inclination to *agentify* human models agents, as found in (Yao, Moody & Bacon 2001, Dignum, Meyer *et al.* 2002, Nickles, Rovatsos & Weiß 2002, Zambonelli, Jennings *et al.* 2003, Dignum, Vázquez-Salceda *et al.* 2005) is not appropriate for modelling viable software methodology to implement reliably evolvable software systems. It invokes Agent-based frameworks, which are usually domain-specific, accompanied by tools and methods that are difficult to implement for most software architects and engineers. It can be claimed that the question-marked area in the three-ellipse diagram demonstrates exactly the dynamicity of collaboration between humans and technical components, namely the criteria for monitoring and validation, particularly with machines that are beyond the HCI capacity. These criteria influence the changes that are required for handling both sub-ideal situations and unexpected human behaviours.

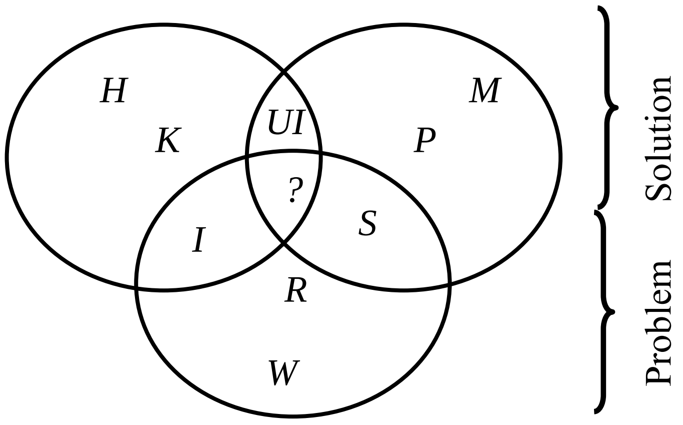


Figure 2.3 Problem Frames' three-ellipse model of requirements

Throughout this thesis, the approach continues to conform to software architecture principles by externalising and governing human interactions only. However, the key issue herein is to compensate the human knowledge *K*, which is a subject of design, by providing design constructs (e.g. roles, tasks and social laws) to support generic representations, as explained in Chapter 4, and a methodological approach elucidated in Chapter 5.

2.4.2 Speech Acts and Behavioural Implicit Communication

Austin proposed the Speech Act theory in (Austin 1962) and it was carried forward by (Searle 1969, Searle 2002). The rationale behind the theory is that a language is not limited to stating the affairs of the world but also has the capability to bring about new states of affairs. The utterance of specific language sentences constitutes acts, which they refer to as *performative* of speech acts (e.g. "I apologise"). Searle classified speech acts according to one of five fundamental points: assertive, directive, commissive, expressive and declarative (Searle 2002). If one takes a sample sentence: "I promise to meet you tomorrow", the main parts are called illocutionary points of an utterance, which contain illocutionary force, (“I promise” as a force indictor) and propositional content (“meet you tomorrow” as an asserted proposition).

Conversation of Actions is a generic sequence of related speech acts proposed by (Winograd & Flores 1986), which allowed modelling human-agent interactions and encouraged agent communication languages such as KQML (Finin, Labrou *et al.* 1995) and FIPA ACL language (FIPA 2000) .

Behavioural Implicit Communication (BIC) is another communicative approach that adopts a more intuitive way to achieve collaboration without explicit communication (Castelfranchi & Giardini 2003). The BIC approach does not require *special* or *specialised* behavioural signals to be added to the set of purposeful interactions between entities but rather exploits genuine and purposeful interactions as communicative vehicles to improve coordination. However, one of the main deficiencies of speech acts as a model of communication is that it requires direct communication via a formalised language that might be suitable for agents but not for humans. Very often indirect communication is not only common, but also more effective. BIC allows effective communication between the configuration manager and participants via the alteration of software and hardware components around them. Such alterations would be taken more seriously than a notice appearing in a user interface. In other words, what a participant does as a purposeful interaction to bring about a new state of affairs on a certain occasion cannot always be uttered or asserted, in parallel, into a communication media or protocol e.g. CSCW.

Putting into consideration that collaboration cannot be realised without communication, the communicative element pertained to these actions should be represented as a physical indicator (i.e. certain signified behaviours to be contextually used as massages that can be perceived by other collaborators to act upon. A common case is when a participant enjoins a required behaviour that is usually prohibited in ideal contexts signalling a role or a coordination violation. Unfortunately, this is often overlooked at design time by system specifiers who are responsible for encoding scheduled processes and routine practices.

Putting forward actions for communication also coincides with the essence of workflow management systems. These are recognised among other collaboration-based technologies, such as Computer Supported Cooperative Work (CSCW), precisely for the way they bring task allocations to a first-class entity for collaboration rather than any other communication protocol. Well-defined workflows, with clearly defined initiation actions, termination actions and final goals, are necessary for overcoming the difficulties of extracting intentions and goals of the performing agent, provided that these actions are performed intentionally.

2.4.3 Social Interactions and Control: the Deontic Way

The inclusion of a "social dimension" means that people (or groups of people) need to be considered not as external users but as another class of components that, together with software and devices, perform roles that are vital for the "good" behaviour of the system. This requires a modelling approach that incorporates normative concepts such as permissions, obligations and power, and that models the violations that can take place so that processes and underlying software can be reconfigured to react to non-normative situations in ways that ensure agreed, possibly minimal, levels of service.

The usefulness of *deontic logic* for modelling the behavioural aspects of systems was identified by logicians and computer scientists three decades ago. Deontic-based frameworks highlight crucial aspects of modelling biddable interactions of social entities through normative relations which include formalising the logic of obligations and permission (i.e. deontic logic), capturing *sub-ideal* situations and imposing obligations on their presence (i.e. *contrary-to-duty obligations*), and thereby prompting alternative control mechanisms to handle *violations of norms* committed by these entities (i.e. *normative positions*).

2.4.3.1 Deontic Logic

Deontic logic is a branch of philosophical logic that provides a formal framework for modelling and reasoning about permissions, prohibitions and duties or obligations. What is appealing in deontic based approaches to modelling behaviour, from the point of view of social interactions, is the clear separation that is achieved at the formal level between the *indicative* (how things are) and the *optative* (how things should be) modes of system specification[2]. The distinct advantage in using deontic concepts is that they allow definitions of concrete normative system behaviour, but at the same time, they sustaining the possibility of capturing behavioural forms that do not comply with those norms. The first standardised system of deontic logic was proposed by Von Wright in the fifties (von Wright 1951) and it has been followed by many variants and variations: e.g. dyadic deontic

[2] Specification modes are borrowed and adapted to be used in a slightly different context from (Jackson 1995). Jackson, M., *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices*, 1995.

logic (von Wright 1964) and deontic dynamic logic (Meyer 1988). The extensive research in this area gained no successes in unifying these approaches as most of them are still paradoxical (Hansen, Pigozzi & van der Torre 2007) and semantically problematic in relating obligations to permissions (Boella & van der Torre 2003).

Deontic-based frameworks do not insist that social participants adhere only to normative behaviour; they allow for the exploitation of possible violations as sources of information, before deciding upon which remedial actions must be taken, or which sanctions to apply on biddable participants when capturing non-compliant behaviour.

In this context, the thesis focuses mainly on the advantages of using deontic concepts for modelling autonomous interactions as previously established in the following domains of interest: formal organisational societies (Dignum, Meyer *et al.* 2002, Dignum, Vázquez-Salceda *et al.* 2005), multi-agent coordination (Barbuceanu, Gray *et al.* 1999), business processes (Padmanabhan, Governatori *et al.* 2005), and speech acts (Johannesson & Wohed 1999). The argument for using these concepts is that organisational models need to capture social behavioural patterns within business processes operating in open environments. Thus, this architectural approach requires mechanisms to systemise, support, and recommend efficacious or "good" behavioural patterns on the one hand, and impose sanctions on "bad" ones on the other hand. These mechanisms provide reliability and trust to the overall system behaviour.

The essential element of incorporating such concepts to business processes is to provide monitoring mechanisms for both expected (requested) and the entitled (empowered) aspects of actions, in order to deal with possible violations by detection and sanctions. The concept of normative positions—among other deontic concepts—enables establishing formal frameworks over which system modellers can reason about the different degrees of compliance, entitlement and capabilities of social entities.

2.4.3.2 Ideal and Sub-ideal Worlds

The idea of *sub-ideality* has been developed as a response to a criticism related to derived obligations, namely obligations that are not conditional but arise under certain factual circumstances (e.g. obligations that are imposed due to the detection of a violation of another obligation). Von Wright introduced the first notion of the context-based form of Standard Deontic Logic (SDL), represented by the following combination of symbols *P(p/q)*. It is interpreted as follows: "it is *permitted* that p *given that* (on condition that) q"

(Lomuscio & Sergot 2003a). This idea of sub-ideality has been extended in several research papers (Prakken & Sergot 1996, Carmo & Jones 2002) to provide Contrary-to-Duty obligations (CTD) that are put into actions when primary obligations are *violated.* A well-known example of primary (a) and CTD obligations (b) is:

a) You should not kill Mr. X

b) But if you kill Mr. X, you should do it gently

The CTD obligation is put into action when the primary obligation is violated which brings about a sub-ideal situation. CTD formalism has several problems:

- There is no provision for controlling the overall behaviour when the primary and/or the CTD obligation are violated.
- An inherent problem of SDL that it cannot model different levels of sub-ideality: Modellers can only model a flat range of sub-ideal worlds (Carmo & Jones 2002).
- The formal relationship between permission and obligation is still subtle and pose some logical paradoxes as described in (Boella & van der Torre 2003).

2.4.3.3 Normative Positions

Normative position is a method for outlining the space of all logically possible "positions" of some specified entity. It was originally established by (Kanger & Kanger 1966, Kanger 1972, Lindahl 1977) and extended by (Sergot 1999, Sergot 2001). It can be viewed as a combination between deontic logic and the logic of action-agencies resulting in a formal representation of normative concepts like duties, rights, authentication, etc.

Kanger's and Lindahl's theory has a normative component for reasoning about what kinds of actions are ethically or morally desirable or even permissible. It consists of a normative position together with a corrective action that influences how well a particular action performs in correcting a sub-ideal situation (Sergot 2001). Such a normative component is complementary to other "causal rules" in the sense that it provides a guide for corrective actions to recover — from a sub-ideal situation — to a stable state of a system.

2.5 Contextualising Social Interactions

The presented research asserts that human interactions are part of the human communication and social systems. More specifically, monitored human interactions are constituents of social systems that enable architectural reconfiguration mechanisms to interpret reason about and provide the required changes across global configuration protocols.

In other words reasoning about human interactions as a constituent part of the monitored contexts can be used as a vehicle to extract these behaviours from their *agentfied* aspects e.g. BDI. These can then be exploited as architectural building blocks for a constructive approach to modelling norm-based socio-technical systems.

In hospital settings for example, system goals are a little blurred; even indicators like customer satisfaction are quite difficult to measure — at least over the short term. Also the relations between the participants are complex and the context elements are almost beyond the cognitive capabilities of existing monitoring systems. Even if the context elements are successfully captured, normally the doctor's interpretation and judgment prevails over statically prescribed rules.

Similarly, these interaction-centric social concepts can be well-extended to cover different domain of interests with respect to socio-technical systems. Comparing the above-mentioned medical-oriented role relationships with their counterparts in other domains (e.g. flight-cockpit interaction modelling), I have perceived more formal relations between the pilots' roles with regards to normative positions. I also noticed that, unlike healthcare settings, the cognitive power gap between system participants (i.e. pilots) and the cockpit monitoring system is almost negligible in normal situations. Flight cockpit studies usually concentrate on display concepts, workload and situation-awareness.

Additionally, the intertwining between the auto-pilot system, which includes monitoring and actuator agents that interact with the environment in a causal way, and the human pilots, who may take the initiative at any time and override decisions that were made during automation, clearly defines the borders between causality and biddability.

Nevertheless, the problem is still there: what are the relevant properties for a sub-ideal situation? The main problems with the context-driven adaptations are:

a) Defining the "minimal set" of the context parameters: as an example heart-beat rates, whether for an athlete or an 80 year old man.

b) Guiding humans when there are several alternatives that crosscut a single defined “sub-ideality”.

These problems are exacerbated when handling intensive software systems that operate in a technology-rich environment where software components mingle with social and technical ones. Monitoring mechanisms for the context, and thus interaction management, are highly dependent on the definition of the constituent context elements. For example, (Summers, Jansen *et al.* 1997) shows how tricky and controversial the standardisation of the minimal data-set in an ICU bed is.

2.6 Discussion

Deontic logics and related formalisms have been the focus of this chapter. Due to the severe problems that deontic logic poses, this thesis is incorporating its essence in an engineering context (i.e. configuration-oriented). The modelling of biddable social interaction dynamics has, however, not attracted widespread interest. In the first place the impact of such dynamics on system well-being has not been widely recognised. The level of expertise, data and resources required to build and calibrate such models did not appear justified; there is a perception that the technological means for tracking these interactions are insufficient in the face of continuously changing contextual information.

Additionally, in my view, sub-ideality is not only a property of the environment (e.g. there is a patient in a critical situation) but also include the way the protocol is behaving in a particular context (an unqualified medical staff manipulating a ventilator machine that is attached to the patient) or a qualified one insisting upon adjusting the ventilator when normative conditions are not met.

In order to respond to potentially disastrous situations, every piece of equipment has to be interconnected with social interactions using *social wires*: connections that govern these interactions should exploit normative positions. These wires have to address the following issues:

- Does the social interaction modelling (and the reconfiguration adaptations that incur) relate significantly to organisational settings, software and available resources, and if yes, how?
- How can a faithful representation of people, roles instances, role players and the context level of ideality be maintained?

- Can the model be systematically updated to remain an accurate representation of the possible technical configurations, on the one hand, and the organisational structure on the other?

The RNS approach (Weiß, Rovatsos *et al.* 2003) mentioned in Section 2.3 shows several interesting parallels with my work since the research team centred their idea around agent autonomy and norm-deviation management by providing roles space and positive and negative sanctions respectively. Moreover, the approach provides a GUI to support norms validation dynamically. Normative actions equate to process change, which potentially moves the running process from one process to another.

In summary, the presented literature review showed how agent-oriented approaches to normative agency can be used in the domain of process modelling/management and how rules can define who should be empowered through rights assignment/alteration. Such power assignment rules (or competence conferring norms in the sense of (Spaak 2003) are often represented using a *counts-as* structure, e.g. (Jones & Sergot 1996), to denote the context in which they operate.

The main aim of this chapter was to illustrate the kinds of nuances and distinctions that can be ascribed to biddable human interactions compared with causal ones, and to pave the way for the subsequent chapters — namely 3, 4 and 5 — to indicate how these interactions can arise in practical "architectural" settings.

.

Chapter 3

Software Architecture from Different Perspectives

3.1 Overview

In Software Engineering, interaction-centric approaches can be supported by "architectural techniques". In the last two decades, architectural modelling approaches have promoted interconnections to first-class citizens by extracting the code that, in the components, is responsible for interactions, into connectors (Shaw & Garlan 1996, Allen & Garlan 1997a, Bass, Clements & Kazman 1998). Architectural concepts and technologies provide high-level standards and open interfaces for the delivery of high quality software systems that are maintainable and evolvable (i.e. *agile software systems*).

This chapter reviews the terminology and the concepts of software architecture based on an examination of existing research literature together with my own insight, and therefore included the interactions of non-technical elements (i.e. social/human components) within architectural models. Among other architectural concerns, dynamic evolution of software system has emerged as a crucial feature; it is required by software business clients to handle changes in system requirements or the environment they operate in.

3.1.1 Software Architecture on the Move

"Software Architecture" as a software engineering discipline intuitively borrows "architecture" as term which refers to architecture in buildings and urban planning (Alexander, Ishikawa *et al.* 1977). This informal explanation of software architectures is tightly coupled to "the structure" of software systems (Allen & Garlan 1997b), namely bringing apparent architectural skeletons (Kramer 1994) to fore in the sense of those promoting structural properties that can be perceived as higher level patterns or first class entities. Conversely, movable parts such as computational units are hanged on these structural elements (c.f. flesh in the human body). Consequently, they can be abstracted and thus can be replaced or altered through local adaptation or *reconfiguration*.

Architectural concepts do not only provide means of controlling the complexity of developing software, but also play a vital role in supporting the current need for systems that are evolvable and configurable at runtime (Andrade & Fiadeiro 2003). The demand for such architectural quality keeps growing as industrial application software systems strive to provide means for flexible and runtime re-configurations.

3.1.2 Engineering vs. Architecture

Software architecture as a discipline aims at supporting software developers primarily in creating structural and behavioural blueprints of systems. However, targeting runtime adaptivity would also maximise the role of customers and users in the development and the evolution of systems. Software architecture then would incorporate some features of other architecture disciplines: focusing on the customer's requirements and then designing the answers of these requirements in terms of effective software artefacts within given economic and technological constraints.

Correspondingly, and with regards to agility, software architecture approaches focus on supporting developers by maximising their role in the *evolution processes*— an inclination that is inherited from engineering— whereas they should have focused, as a tool for architects, on maximising the qualities that should be enjoyed by *capable* system users, participants and domain experts (e.g. usability, modifiability and resilience to runtime changes).

3.1.3 The Chapter's Structure

This chapter is structured as follows: Section 3.2 reviews the existing terminology, concepts and various architectural approaches. It also examines the capacity of existing architectural approaches with regards to providing suitable abstractions to handle and reason about the interactions of non-technical elements (i.e. components). The concepts and practicalities of architectural evolution are the focus of Section 3.3 to explicate the different approaches to software agility, in general, with particular interest on reconfiguration techniques. Section 3.4 explains the 3Cs architectural approach to agile and interaction-centric software architectural modelling which will be used, throughout this thesis, as a vehicle to convey new concepts, primitives and techniques for addressing biddable interactions.

3.2 Software Architectural Paradigms

The scope of Software Architectures overlaps several other research areas in Software Engineering and Programming Languages (Andrade et al 2003). For instance, the central concept of separation between "Computation units" or subsystems—the way basic functionalities of system are ensured— and "Communication"—the mechanisms through which these subsystems can be reconfigured in the sense of Configurable Distributed Systems (Moazami-Goudarzi 1999).

Software architecture paradigms can be differentiated according to the view of architectural elements: Perry and Wolf define processing elements as "transformation of data"(Perry & Wolf 1992), whereas (Shaw & Garlan 1996) promote *components* as "the locus of computation and state". This component based view was elaborated by (Shaw & Clements 1997): "A component is a unit of software that performs some function at runtime". Researchers such as (Fielding 2000) emphasise the differentiation and the separation between structural and behavioural abstractions of software systems as the former concentrate on the modularity of the static source code whereas the latter captures the runtime behaviour of system elements. Others e.g. (Bass, Clements *et al.* 1998) advocate that system architects can benefit from a combined view of both abstractions. The definition of software architecture in their well-known book— *Software Architecture in Practice*—is widely acceptable among the software architecture community: "*T*he software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them*".*

3.2.1 Architectural Elements

(Garlan & Shaw 1993) describe their view of system architectures as a collection of computational components associated with a description of the interactions between these components—the connectors. The architecture of a software system articulates that system in terms of components and of the behaviours that emerge from interactions among those components. In addition to specifying the structural and topological view of the system, the architecture intuitively interlinks system requirements and elements of the constructed system.

These architectural concepts provided the conceptual framework on which researchers later develop formal frameworks, models and languages to capture formally the informal diagramming icons (i.e. boxes and lines) that represent system modules and their interconnections. Hence, software architectures describe the overall properties of the system structure in the sense of what has been required for a while by the developers of complex systems, namely programming-in-the-large (DeRemer & Kron 1976). This structural view of complex software systems reveals behavioural properties of software systems. The captured behavioural properties are those that can be modelled through high level architectural patterns (i.e. connectors) which by their existence and their evolution are the determinant of system behaviour (e.g. contracts).

3.2.1.1 Components

Components are the most recognisable elements of software architecture. Research in software architecture devised different understandings of components; however, most of them can agree upon the aforementioned taxonomy of (Perry & Wolf 1992). (Garlan & Shaw 1993) describe components simply as the elements that perform computation. My research builds on Garlan and Shaw's general view of components as units of computation and adheres to a generic definition of components, as defined by (Szyperski 1998) which advocates that components can support multiple interfaces and can be internally composite.

3.2.1.2 Connectors

Connectors are abstractions by which communication and/or coordination between components can be achieved. (Perry & Wolf 1992) describe connecting elements as the glue that holds the various computational elements of the architecture together. A more precise definition is provided by (Shaw & Clements 1997): “A connector is an abstract mechanism that mediates communication, coordination, or cooperation among components”. From the functional point of view, several techniques were used to implement connectors as mean of communication between components: shared representations (e.g. Linda’s tuple space (Gelernter 1985)) remote procedure calls, message-passing protocols, and data streams.

An architectural connector consists of: (1) a set of roles that capture the types of component that can be interconnected and (2) a glue that enforces an interaction between components that instantiate the roles. The formalisms that are used for the roles and the glue differ from one approach to another, as it explained later.

Generally speaking, in interaction-centric approach to modularisation as promoted in the area of software architecture, connectors coordinate interactions as external entities. In the connector-based approach coordination mechanisms put in place through connectors are activated across *wires* that link components in the underlying communication network and thus can be superpose dynamically. Hence, evolution can be made to be compositional over the architectural structure of the system (Andrade, et al. 2003).

3.2.1.3 Configurations

The concept of configuration recalls different meanings to different IT people: designers, analysts, software engineers and architects. Generally speaking, “a configuration consists of entities or “items” that are present in a system and the inter-relationships between them” (Lock 2005). It can be comprehended as a skeleton or an exoskeletal structure (Kramer 1994) that does not focus on how processes are realised nor how the internal computations of its basic entities are realised, but rather exhibits the functional dependencies between various computational units (components) that emerge from their participation in a certain execution scene or protocol. Within an execution scene, physical connections are superposed on the participants to control their interactions. A patient station at an ICU unit or a theatre may be considered as an example of an execution scene or protocol

In other words, complex systems can be described in terms of a set of configurations to convey structural and behavioural properties of such systems at runtime. Various notations—either textual or graphical—may be used to represent configuration visually, facilitate the description of possible computations, interactions and reconfigurations (i.e. possible discrete configurations). When these configurations are presented formally they are capable of setting configuration constraints to control interactions, and reconfiguration, thereby facilitating animation and possibly verification of system properties, which in turn allows us in order to reason about the overall behaviour of the configuration at hand.

3.2.1.4 Properties

The set of architectural properties within software architecture includes all properties that derive from the gross decomposition of complex system into configurations of components and connectors, which ensure they will interact in ways that allow global system properties to emerge. Properties are either functional properties that are achieved by architectural elements or non-functional properties, such as component reusability, dynamic extensibility and robustness. The later type of properties is usually referred to as quality attributes (Clements, Bachmann *et al.* 2004).

Properties stem from the set of constraints within an architectural configuration, which is the sum of all participating elements along with emergent properties that originate from composing interconnections (e.g. applying a set of connectors on the same components might generate emergent behaviour).

As an example of these properties, the *pipe-and-filter* architectural style presented by (Garlan & Shaw 1993, Allen & Garlan 1997b) attains the qualities of reusability of components and configurability of the application by applying generality to its component interfaces—constraining the components to a single interface type. Hence, the architectural constraint is a "uniform component interface", motivated by the inclination to generalisation, in order to obtain two desirable qualities that will become the architectural properties of reusable and configurable components when that style is instantiated within the architecture (Fielding 2000).

3.2.1.5 Primitives

Identifying components that correspond to the domain concepts and element along with types of allowed interconnection are not enough to achieve concrete architectures. In order to construct architectures or architectural styles, *composable architectural primitives* are needed which comprise meta-level semantic modelling constructs that refer to domain elements and correspond to their implementation-level instances to be manipulated at runtime. These primitives exhibit well defined behavioural properties to provide implementation solutions that are compositional with respect to the semantic offered by the modelling primitives (Andrade & Fiadeiro 2003).

Compositionality allows not only a structure obtained at the modelling level to correspond to its concrete instance at the implementation level, but also minimises the effect of changes that are required at runtime (i.e. reconfigurations at the implementation level) when changes operated at the level of the business model. Supporting locality of changes that are performed at runtime, without taking down other services running parallel to the targeted service is a main objective of agile software systems (Andrade & Fiadeiro 2003).

Alfa framework of (Metha & Medvidovic 2003) is an example of how a framework can support composing architectural styles from architectural primitives. Their proposed technique is intuitive; however, the abstraction levels of these primitives are very low when handling business and domain-specific entities.

3.2.1.6 Styles

Styles in architectures were identified initially by (Perry & Wolf 1992) who emphasised constraining architectural elements and their relationships. (Garlan & Shaw 1993) defined styles in terms of pattern interaction among typed components continuing to treat software architectures as formal description of system. (Moriconi & Xiaoli 1994) coined another definition that highlights the gap between abstract or design elements that participate in a style and concrete ones. Their view of styles comprises the same architectural elements as *vocabulary* of design elements along with a set of well-formed constraints that must be satisfied by any architecture written in the style, together with a semantic interpretation of the connectors. The first departure from this static view towards handling architectures as running systems was given by (Abowd, Allen & Garlan 1995) who concretised and formalised styles as the syntax of components and connectors, behaviour as their semantics, and topological structure as the syntax of configurations; however, the did not separate interaction and data aspects. In this line, (Mètayer 1998) formally specified communications based on the geometry of architectural styles in terms of a graph grammar which model the box-and-line analogy. A more recent view of architectural styles takes commonality into account in order to abstract a collection of architectures by common resource types, configuration patterns and constraints (Fiadeiro, Lopes & Wermelinger 2003).

Supporting a particular architectural style or styles provides the ability to "specialise" generic interaction-governing patterns in the sense of (Mètayer 1998) and/or architectural adaptation at runtime. Analogously, and from a methodological point of view, one would recall Loerke's view of architectural styles (Loerke 1990), in the sense of programming styles, which postulated as critic's view where past architectures are accumulated and imposed in the form of constraints on the architecture at hand. In other words, a style is considered as a method of abstraction, rather than a stack of personalised design experiences. With regards to supporting architectural adaptation, there are few research attempts that utilise architectural styles to guide the process of adaptation (Garlan, Cheng & Schmerl 2003). More details in this perspective can be found in Section 3.2.3.1.

3.2.1.7 Views

A view is a projection of a whole system from the perspective of a related set of concerns and refers to a particular architecture of the system. Kruchten provided a comprehensive set of views namely 4+1 views: (logical, process, physical, development), and scenarios without any specific notation (Kruchten 1995). (Hofmeister, Nord & Soni 1999) offer a systematic, detailed architectural design method and a representation of software architecture. They use UML meta models to define conceptual, module, execution and code views. Compared to (Hofmeister, Nord *et al.* 1999) and (Kruchten 1995), Issarny and his colleagues demonstrate a functional and interaction view along with various quality attributes such as efficiency and dependability (Issarny, Saridakis & Zarras 1998).

Architectural views are defined by (Clements, Bachmann *et al.* 2004) as follows: "A view is a representation of a set of system elements and the relations associated with them". They devised three categories of views: Module, Component-and-Connector and Allocation. Another technical conceptualisation of views given by (Heckel, Engels *et al.* 1999): "A view is an incomplete specification of a system focusing on a particular aspect of a subsystem". The approach of Heckel et al. to view modelling partially specifies the structure of the system's state and analogously captures partially what the effect of an operation is (Heckel 1998, Heckel, Engels *et al.* 1999). This provides partial or loose semantics of transformations where views conform to a *reference model*. Chapter 6 will shed more light on how to exploit Graph Transformations (GT) to model and integrate architectural views.

3.2.2 Architectural Definition Languages (ADLs)

Architectural Definition Languages (ADLs) have been the focus of attention of architects and developers as means of specifying structural and behavioural properties of software systems. They facilitate constructions of high level models in which systems are specified as compositions of architectural elements (i.e., components and connectors) and support reasoning about structural and/or behavioural properties at early stages of the software engineering life-cycle. According to (Medvidovic & Taylor 1997), ADL is a language that provides features for the explicit specification and modelling of a software system's conceptual architecture, including at a minimum: components, component interfaces, connectors, and architectural configurations.

Corresponding to aforementioned understandings of the software architecture concept there are parallel approaches for specifying, implementing and reconfiguring ADLs. For example, Darwin (Kramer & Magee 1998) is a declarative architectural definition language which is intended to be a general purpose notation for specifying the structure of systems composed of diverse components using diverse interaction mechanisms (Magee, Dulay *et al.* 1995). Darwin's interesting qualities resulted from applying lessons learned from a previous framework (i.e., Conic, (Kramer 1990)). The dynamic composition of architectures is among its distinctive features, which allow for the specification of distributed architectures. Parallel to this approach, (Allen & Garlan 1997b) provide a formal basis for specifying the interactions between architectural components through specifying connector types by their interaction protocols using CSP (Hoare 1985), CHAM and pi-calculi for formalising the approach.

While earlier ADL proposal focused on presenting the static modular view of the system code, recent research surveys (Batista, Joolia & Coulson 2005, Gomes, Batista *et al.* 2007) have recognised the benefits of coupling ADLs with underlying runtime environments to support systematic and integrated system development.

Viewing ADLs from another angle, they can be perceived as methodological approaches and thus often introduce specific architectural assumptions that may limit their ability to express some architectural styles. More specifically, an ADL could be designed particularly for a certain architectural style, thus improving its capacity for capturing the properties of this style at the expense of generality. Conversely, ACME (Garlan, Monroe & Wile 1997) is an ADL that attempts to be as generic as possible, however this feature comes at cost of overlooking style-specific analysis.

With regards to software system agility, coupling ADLs with runtime environments drew the attention of software architects to invest on "Coordination Languages and Models", as presented by (Gelernter & Carriero 1992). The reason for that is it allows putting forward: (1) concepts such as separation of concerns (Hursch & Lopes 1995, Mens & Wermelinger 2001, Andrade, Fiadeiro *et al.* 2002), (2) architectural primitives (Andrade & Fiadeiro 2003), and (3) mechanisms through which architectural models can be reconfigured dynamically (Moazami-Goudarzi 1999).

3.2.3 Architectural Reconfiguration Models

Known classes of software systems that are able to benefit from dynamic software evolution include 24X7 systems, which should adapt to frequent changes in their execution environment. Emphasis should be placed on achieving agile systems through *architectural reconfiguration* or *adaptation* as a runtime concept which makes a departure from the well-know term of *software evolution* devised by (Lehmann 1980) which focuses, instead, on the whole development process life-cycle steps and requires the system to be off-line when performing changes.

Conic, as on of the earliest attempts at dynamic ADLs, introduced a general model for dynamic reconfiguration which only permits change to occur when the affected portions of the system are in a quiescent state. Structuring systems as interacting components was the key to address the scale, complexity and evolution involved. This research pioneered the separation the structural language—referred to as a *configuration language,* which describe component configuration of the system—from component composition support, however it was dependant on a specific programming language and it lacked separation of concerns.

This led to a new architectural framework, i.e. Darwin, equipped with a language for describing software structure in terms of components and their bindings. It is a pure declarative language, with sound semantics. Darwin has been designed to be sufficiently abstract to support multiple views in the sense of (Kruchten 1995). More specifically Darwin corresponds to a couple of Kruchten's views: the behavioural view and the service view for the purpose of behavioural analysis and construction, respectively. Each view is an elaboration of the basic structural view (i.e., the skeleton upon which the flesh of behavioural specification is hung).

3.2.3.1 Architectural Levels of Reconfiguration

In order to inject dynamic reconfiguration into architectural systems, they need to be causally connected at runtime to the corresponding high-level software architecture specification. In more detail, there are two causally-connected models: an architecture-level model and a runtime-level model. Dynamic reconfiguration can be applied either through an architectural specification at the architecture level, or through reconfiguration primitives at the runtime level.

The main goal of successful reconfiguration abstractions is to endow instance architectures with the capability to manipulate portions of its elements at runtime. These abstractions reside in either of the following architectural levels of abstraction, which can be utilised for the purpose of applying changes: ADL/style level and instance level:

- ADL or style level: generic patterns—an example could be a 'protocol stacking' style, which defines a basic set of elements and constraints for describing linear compositions of 'protocol' components.
- Instance level—domain-specific and easy to handle in terms of adding/deleting components and/or connectors.

An example of the latter (i.e., instance level) is Wermelinger's approach to reconfiguration (Wermelinger 1999). This approach provides simple reconfiguration scripts rather than a modelling language with complex construct. Reconfiguration scripts consist of primitive reconfiguration operation e.g. adding/deleting components and/or connectors. However, within the CommUnity framework, reconfigurations modelled using Graph Transformation (Wermelinger, Lopes *et al.* 2001). Another example of instance level reconfiguration is OpenCom (Coulson, Blair *et al.* 2004) which uses reflection as mean to query component states and perform ad hoc reconfigurations.

ACME is an ADL promoted by (Garlan, Monroe *et al.* 1997) that exemplifies architecture/style level configuration. Its reconfiguration operations depend on:

- Invariants— ensuring system-preserving constraints despite the dynamic insertion/removal of ACME elements.
- Extension operator— type extension (extend type at runtime)
- representation (local reconfiguration)—allowing re-instantiation of the component with different interfaces
- Properties—used to describe how components maybe changed at runtime.

The gap between runtime level and ADL level management of reconfigurations has been identified by (Joolia, Batista *et al.* 2005) who provided a *causal* connection between the ADL level and the runtime level to support both program and ad hoc reconfigurations. They came up with the Plastik framework, which is a meta-framework that integrates both the ADL level management and runtime level management. Such an integration attempts to address the limitations of depending on one of them in reasoning about architectural reconfiguration.

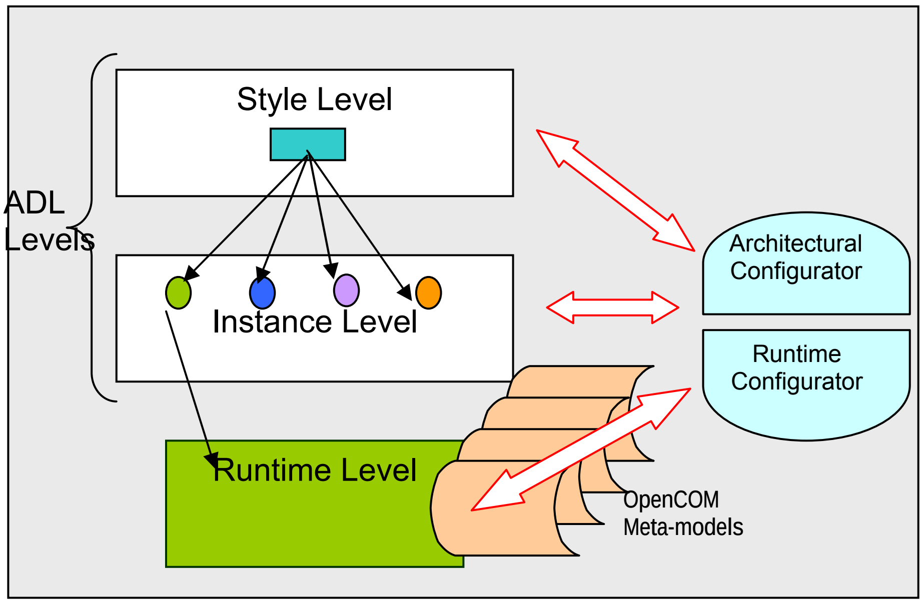


Figure 3.1 Palstik meta-framework (Batista, Joolia et al. 2005)

It formally specifies runtime configurations through integration and possesses an architecture configurator for ADL (ACME) and a runtime configurator for reflective component runtime (OpenCOM). Consequently, Plastik allows reconfiguration at multiple architectural levels, which enable considerable flexibility. Both foreseen (i.e. *programmed*) and unforeseen (i.e. *ad hoc*) reconfigurations are supported. Issues involved in handling these two types of reconfiguration at both levels and the mapping between them, are discussed in he following subsection. Herein, Figure 3.1 illustrates the architecture of Plastik.

3.2.3.2 Reconfiguration Operations

Reconfiguration operations are either *programmed* or *ad hoc*: programmed reconfiguration is a predication-action specification that is supported at ADL level (e.g., when a patient’s vital signs are below average the respiratory machine might be boosted beyond the system constraint), whereas ad hoc reconfiguration stipulates changes that are not and cannot be foreseen at system design. In order to handle such reconfigurations in a way that preserves system consistency, general invariants have to be incorporated into the specification of the system, and any changes have to be checked against these invariants, otherwise, they would be considered as violation. For example a doctor can exceed the system limits of a respiratory machine to handle a critical situation despite the fact it had not been considered at design time.

3.3 The 3Cs Approach

3.3.1 Background

Maibaum postulates that software engineering is a departure from other engineering disciplines where artefacts are conceptual rather than being physical (Maibaum 1993). Fundamentally, software engineering research is different from other engineering disciplines as research ideas may take decades to be filtered through sensible engineering practice. The main reason is that real-world acts as physical constraints on construction of physical artefacts in a way, which is more or less absent in science, and engineering of concepts.

An example of this is the extensive research work of Fiadeiro and his colleagues that put algebraic mathematical specifications at the service of systematic programs construction, behavioural modelling and evolution. The foundations of this approach falls in the tradition of general system theory and the underlying mathematical semantics of this approach is based on category theory, as thoroughly explained in (Fiadeiro 2004).

The purpose of extending these theories is to encompass software application develop a model of collaboration that extends formal notions of architectural connector as mentioned earlier. The semantics of such architectural techniques builds precisely on Goguen's categorical approach to general systems theory (Goguen 1973).

Fiadeiro's approach to formalising software architectures and particularly systematic program specifications take the above concepts into consideration; it matches the style of the counterpart formalising approaches that target ADLs in the sense of (Allen & Garlan 1997b) which in turn became a de facto in (Bass, Clements *et al.* 1998). The distinct feature in Fiadeiro's approach to architectural modelling is that it abstracts the choice of the underlying design language and behavioural models.

Compared to other language-specific ADLs e.g. (Berry & Boudol 1992, Garlan, Monroe *et al.* 1997) which focus on the action-view (and thus models the organisation of the behaviour of the system as compositions of components ruled by protocols of communication and synchronisations e.g. CSP and CHAM), the categorical approach of Fiadeiro et al., focuses on structures and interconnections.

More precisely, the categorical approach of Fiadeiro et al. has built upon on general system theory to specify systematic software construction (Goguen & Burstall 1992, Goguen 1996) and pushed it further towards a structure-based architectures. This categorical approach is reflected by a formalised language CommUnity and its diagramming tool (i.e., CommUnity Workbench) which formalises system structures as categorical diagrams where components are programs and connectors are star-shaped configurations of programs (Fiadeiro & Maibaum 1996, Fiadeiro & Maibaum 1997). The work of (Wermelinger, Lopes *et al.* 2001) provide a formalisation approach for specifying architectural reconfiguration scripts using double-pushout GT approach.

The mathematically rigorous techniques of Fiadeiro et al. informally crystallises the concepts, styles and primitives that were modelled in their 3Cs business architecture (Andrade, Gouveia *et al.* 2002, Andrade & Fiadeiro 2003), which in turn are borrowed from the CommUnity approach (Fiadeiro & Maibaum 1996, Fiadeiro & Maibaum 1997).

3.3.2 The 3Cs Business Architecture

The 3Cs architectural approach has been demonstrated in (Andrade & Fiadeiro 2003) and (Andrade, Fiadeiro *et al.* 2001) which includes a business micro-architecture to support engineers and system specifiers to model and implement evolving component-base systems. Architectural primitives such as coordination contracts (Andrade, Fiadeiro *et al.* 2001) model rules that determine how and when components need to interact in order to fulfil business requirements. The 3Cs can be classified as coordination based approach

(Gelernter & Carriero 1992) that borrows essential ideas from software architecture (Perry & Wolf 1992). It does this to externalise interactions from computations, and exploit superimposition from parallel program design (Fiadeiro & Maibaum 1996) to support compositional evolution.

Indeed, in software architecture, modelling techniques have been proposed for supporting interaction-centric approaches. More precisely, such techniques promote interconnections to first-class citizens (i.e., architectural connectors) by separating the code that, in traditional approaches, is included in the components for handling the way they interact with the rest of the system, from the code that is responsible for specifying computations that are ascribed to services offered by these components.

The 3Cs approach builds on event-condition-action (ECA) rules for coordinating the joint behaviour that a group of components need to execute in reaction to a trigger generated by another component or outside the system. A so-called coordination law defines how a number of partners interact. Partners are not named but rather abstracted as coordination interfaces that define types of system entities in terms of operations that instance entities need to make available and events that need to be observed. As an example, consider the coordination of the way a doctor interacts with a respiratory-control system:

```
coordination interface respiratory-control
import types pressure, ward;
services fixed_in(w:ward):Boolean
       verify():pressure
       decrease(p:pressure): post verify() = old verify()-a;
       increase(p:pressure): post verify() = old verify()+a
end interface

coordination interface doctor-in-charge
import types pressure, ward;
services    work_in(w:ward):Boolean;
          in_charge():Boolean
events      plus(p:pressure);
          minus(p:pressure)
end interface


coordination law restricted-respiratory
partners d:doctor-in-charge, r:respiratory-control
types a:pressure
```

```
    attributes   min,max:pressure
    rules      when d.minus(a)
               with r.verify - a ≥ min and r.in_charge()
                    do    r.decrease(a)

             when  d.plus(a)
             with r.verify + a • max and r.in_charge()
               do  r.increase(a)
     end law
```

Each rule of the coordination law identifies, under the "when" clause, a trigger to which the instances of the law will react—e.g. a request by a doctor for an increase or decrease in pressure. The trigger can be just an event observed directly over one of the partners or a more complex condition built from one or more events. Under the "with" clause, conditions (guards) are included and should be observed for the reaction to be performed (e.g. that the changes in the pressure keep it within the specified bounds and that the doctor has been authorised to be in charge of the device).

If any of the conditions fail, the reaction is not performed and the occurrence of the trigger fails, however, it can be subjected to further treatments.

3.3.2.1 Separation of Concerns

Andrade et al. have advocated the benefits of the separation of concerns in their approach: "separation of concerns helps software developers to get a conceptual grip on large software systems, to reuse parts of the system and to evolve it." The 3Cs business architecture is constructed as depicted in Figure 3.2 and comprises three-layer architecture: computation, coordination and configuration; and two architectural primitives: contracts (interactions) and coordination contexts (governing reconfigurations). Each layer is superposed in a transparent way on the layer below, which facilitates the modification of coordination and configuration policies to make the system evolve. The motivations behind the 3Cs approach to software architectures are taming the ever-growing complexity; modules, objects, components, design patterns; and planning changes whether programmed or ad hoc.

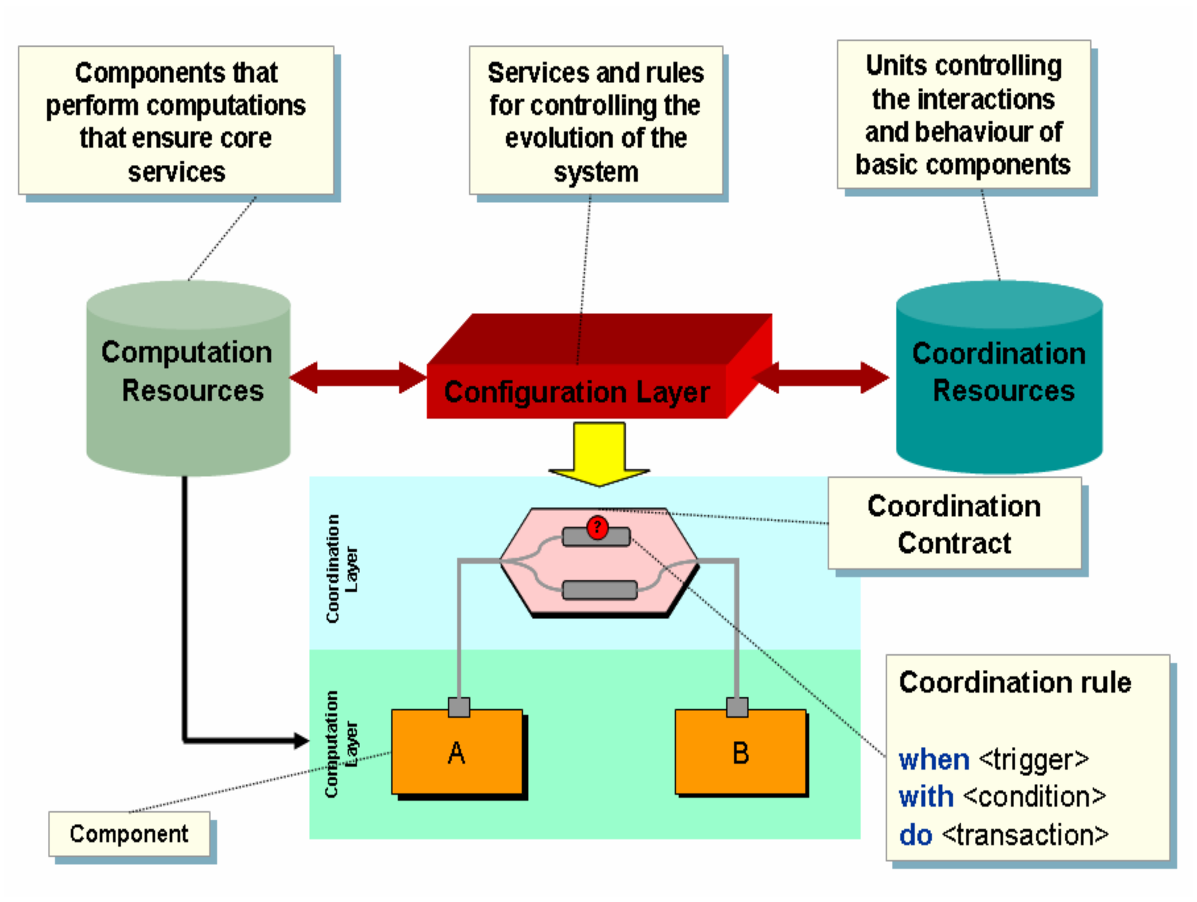

Components that perform computations that ensure core services
Services and rules for controlling the evolution of the system
Units controlling the interactions and behaviour of basic components
Computation Resources
Configuration Layer
Coordination Resources
Coordination Layer
Coordination Contract
Computation Layer
A
B
Coordination rule
when <trigger>
with <condition>
do <transaction>
Component

Figure 3.2 The 3Cs Business Architecture (Andrade & Fiadeiro 2003)

3.3.2.2 Coordination Contracts

The approach has been supported with a tool to provide runtime management of Java components behaviour called Coordination Development Environment (CDE). CDE, as explained in (Andrade, Gouveia *et al.* 2002), exploits the implementation of a collection of design patterns, which aim to terminate the drawbacks of object-oriented clientship. It was a step further toward implementing high level patterns for adding monitoring capabilities to programming languages in the sense of (Notkin, Garlan & Sullivan 1993).

The approach proved to be sufficient to model the key properties for managing interactions through contracts among diversified application domains. For example two case studies: managing software intensive systems (Koutsoukos, Gouveia *et al.* 2001) and tackling business-oriented systems (Koutsoukos, Kotridis *et al.* 2002) have provided a proof-of-concept and demonstrated its suitability to address industrial software development needs.

3.3.3 The 3Cs Approach to Reconfigurations

Normally, in order to enforce specific business policies of the organisation, reconfiguration steps are either programmed or take place in contexts that set constraints on the nature of the operations that can be performed on given configurations and states (Andrade, Fiadeiro *et al.* 2001). Such contexts capture business activities that can be described in terms of collections of connected components, coordination contracts that can be superposed on them, and the rules that define the ways in which this superposition can or must take place.

Regardless of the way existing components operate, and instead of performing changes in the components themselves, the architectural approach superposes, dynamically, new coordination and configuration mechanisms on the components that capture the basic business entities. If the interactions were coded in the components themselves, such changes, if at all possible depend on the availability of source code besides requiring the components to be halted and reprogrammed. Additionally, such changes would incur massive implications on the class hierarchy, and also may generate side effects on all the other objects that use their services.

On the other hand the need for explicit reconfiguration layer, with its own primitives and methodology, is justified by the need to control the evolution of the configuration of the system according to the business policies of the organisation, or more generally, to reflect constrains on the configuration that are admissible (configuration invariants). This layer is also responsible for the degree of self-adaptation that the system can exhibit.

Reconfiguration operations should be able to be programmed at this level, which enables the system to react to changes perceived in its environment by putting in place new components or new contracts. In this way, the system should be able to adapt itself to profit from new operating conditions, or reconfigure itself to take corrective actions and so on.

3.3.3.1 Coordination Contexts

As an example, in the medical domain that I decided to use as an example in the earlier section (i.e. Section 3.3.2), a coordination context normally exists for each doctor. The purpose of this context is to manage the relationships that *doctors* may hold along with various medical equipment that are controlled by the coordination system according to the hospital's codes. Coordination contexts are made available to doctors each time they login to the hospital system. The syntax of contexts can be illustrated as follows:

```
coordination context doctor(d:doctor)
workspace
    component types doctor, respiratory
    contract types restricted-respiratory, open-respiratory,
constants min-RESTRICTED, max-RESTRICTED, Normal-Average:
pressure
services
subscribe_RESTRICTED(d:doctor,r:respiratory):
    pre: exists r and d.work_in(w)
    post:  exists' restricted-respiratory(d,r) and
          restricted-respiratory(d,r)'.min= min-RESTRICTED and
          restricted-respiratory(d,r)'.max= max-RESTRICTED

subscribe_OPEN(d:doctor,r:respiratory):
    pre: exists r and d.work_in(w)and r.fixed_in(w)
    post:  not exists' restricted-respiratory(d,r)and
          exists' open-respiratory(d,r)
rules
OPEN-to-RES:
    whenexists open-respiratory(d,r) and
    avg-pressure = Normal
    post: not exists' open-respiratory(d,r) and exists'
    restricted-respiratory(d,r)
end context
```

Coordination context is "anchored" to a doctor instance, referred to as *d* in the definition of the context (type). Under "workspace" a system specifier identifies the component and contract types that are made available for evolving the way the anchor interacts with the rest of the system. *Configuration services* correspond to operations for ad hoc reconfiguration, i.e. they are performed on demand from users of the system.

Configuration services involve both components and contracts. The above example demonstrates contracts for enabling *restricted* and *open* manipulations of a respiratory machine. These services have pre-conditions through which business policies are enforced. For instance, both contracts are not available if the doctor is not a member of the ward in which the machine operates.

Configuration rules allow modelling *programmed reconfiguration*, i.e. to the ability of the system to reconfigure itself in reaction to external events or internal state changes. In the example above, *open-respiratory* contract is replaced by a *restricted–respiratory* one when the average balance of the pressure of a patient reaches the normal value and captured by the *normal-average* monitoring contract. Typically, the programmed configuration rules capture more dynamic properties that model the system reaction to changes in the configuration properties.

Adaptation logic in the 3Cs business architecture is obtained from both architectural constraints and configuration operations (Dowling & Cahill 2001). Therefore, the 3Cs as well as the advocated extension are categorised among self-adaptive systems as they provides *implicit adaptation* triggered by changes in the internal state of the system. Both concepts are similar in providing a clear separation between computations and reconfigurations.

3.4 Architecture and Non-causality

Software architects have been preoccupied with causality aspects, which prevail in purely technical systems (e.g. telecommunication systems) or strictly managed ones—where people can be replaced with machines. Even when they are used to capture business models (e.g. banking), they lose insight into the workplace environments where people (knowledge workers) are bid to adhere to codes of norm that embody social and organisational aspects, when they interact with technical systems.

Putting into consideration the example in the previous section, if there is a need to alter the respiratory machine pressure out of the allowed scope to save someone’s life and any of the invariant conditions fails (i.e. invariants that govern both programmed and ad hoc reconfiguration operations), the reaction is not performed and the trigger fails. Thus, explicit mechanisms should be defined for handling such failures in such contexts. There is neither a provision in the 3Cs approach, nor in any other architectural approach that I know, to model interactions that are only biddable, i.e. situations in which people (social components) are requested to perform given operations but the system cannot cause (force) them to perform these operations. For instance, biddable interaction would occur if the doctor would be requested to alter the current settings. In summary, one needs a richer model of interaction that can capture the fact that coordination in the presence of social components cannot be causal.

One of the early-bird attempts to address dynamic user processes within software architectures is Aura architectural framework (Sousa & Garlan 2002, Sousa, Poladian *et al.* 2005) which matches the specific needs of ubiquitous computing. It focuses on the support of its newly devised attribute of user mobility. User should take the full advantage of local capabilities and resources according to his captured task-based intents. However, the dynamicity in this framework was only pertained to the technical view to handle changes in the environment and the technical resources while the user roam within the system space.

3.4.1 Discussion

New levels of reconfigurability are needed that enable software intensive systems to respond, in a way that is both flexible and predictable, to changes in operating conditions, including those that result from variations in social and organisational contexts.

Coordination contexts fit well with: (1) scheduled tasks, (2) static actors—that are presented as business entities with fixed sets of permissions toward their operations—and (3) a data store that can be subjected to querying and manipulation. A coordination context whether executing a programmed or an ad hoc reconfiguration, causally validates the preconditions of a new process entry. Yet this process entry corresponds to a routine process that originates from, in the former case, capturing an expected change in the environment (i.e., programmed reconfiguration), or executing a reconfiguration script by a hidden system user with whom this coordination context is associated in the latter. For example, a branch clerk retains the authority of executing a configuration script that belongs to a coordination context of a customer account once the corresponding customer appears before the branch desk.

The challenge is to present an extension of this framework which could handle the above mentioned situation in a way that distinguish between the case of standing in front of a clerk or a branch manager for example. The latter can offer more services or can use his authorisation or capabilities to suppress routine processes or violate certain rules to handle unexpected situations.

In other words, achieving the balance between the necessity to initiate a certain task to alleviate an unwanted state or sub-ideal context, in the one hand, and the determining the capabilities of the human participant enacting a role, on the other hand, is the enabler of the success of a norm-based reconfiguration mechanism to support biddable interactions in emergencies. This mechanism should take in to consideration the availability of the minimum required technical elements to realise these capabilities when required. Needless to say, such a mechanism must have an external control sub-system to actuate reconfiguration by means of facilitations and sanctions that matches norms of code and behavioural control policies that are found in organisations.

3.4.2 Questions

The review of human interactions handling in software development paradigms presented in Chapter 2 and the study of coordination-based software architecture as examined in this chapter jointly concretise the targeted research problem as they both highlights the need to tackle social interactions at the architectural level and to address them within organisational processes. The result of both reviews emphasises the importance of adaptivity (i.e. reconfigurability) in attaining required systems properties in the face of changing requirements or environments. However, adaptivity models have to determine what new concepts have to be elected as first-class citizens in order to enrich the adaptation logic and mechanisms. Thus, the research problem is expanded to accommodate secondary questions that address such relevant issues. The subsequent Chapter will be dedicated to specify an extension to the 3Cs framework in order to provide flexibility and responsiveness to contextual changes and unexpected human behaviours. The answers for the following question will be discussed in the subsequent chapters:

- How can social/human interactions be handled as part of system configurations if they are implemented with the 3Cs architectural approach?
- How much "emergent" behaviour needs to be pre-planned at design time in order to monitor it and respond to it?
- How can modellers guarantee that the changes caused by human interaction will be supervised properly in such a model when interactions constitute an integral part of it?
- Can modellers map human capabilities to context and technical resource management as a mean for modelling the reconfiguration space for responding to biddable human interactions?
- Is it applicable to treat social interactions from an organisational perspective by encoding organisational abstractions (i.e., organisational charts, tasks and norms) into architectural configurations?

Chapter 4

Architecting Social Interactions: A Ternary of Roles, Norms and Reconfiguration

"Reasonable people adapt themselves to the world. Unreasonable people attempt to adapt the world to themselves. All progress, therefore, depends on unreasonable people."
George Bernard Shaw

4.1 Motivations & Objectives

The success of today's organisations depends mainly on the quality of the staff as well as the flexibility and interactivity of the organisational system they belong to. (Berens 2005) states that: "*Today, high demands are made on staff regarding expertise, communication ability, and commercial skills […] With automations, the purely routine aspects of the process can be supported more and more effectively or can be omitted altogether*". Hence, flexibility is an essential condition for the success of any socio-technical system that exists in any organisation. It should be exhibited in its product development, software systems, process control and automation, especially for the knowledgeable staff. This type of flexibility is needed to take on board the biddable interaction of social participants as discussed in Chapter 2. From the organisational point of view, managing such kind of interactions is particularly required to manage *fluidity*: an organisational phenomena that results from shifts in focus, priorities and roles of participants (Moran, Thomas & Anderson 1990, Edwards 1996).

In summary, the key objectives of this chapter is to develop an architectural framework that could be tailored to the application domain and reason about the actual behaviour of the application's active participants, particularly human/social ones. More specifically, the framework should give answers to the following questions:

1. What norms of the system should be respected by participants?
2. What properties of the system context do these norms rely on?
3. What should be done when these norms are violated and how does the context and roles of system participants affect the system response to these violations?
4. How to carry out system adaptations to take on board biddable human interactions in organisational settings and respond to them by means of high level reconfiguration operations?

From the organisational point of view, means of control should be provided that are not limited to the enablement of the causal management of systems' participants as seen from the point of view of automated and fixed use-case-like processes down to the level of monitoring each task they carry out. Therefore, this thesis explores an extension to the 3Cs architectural approach using new abstractions that support realising higher level processes (i.e. Human-driven processes).

The distinction of the above two levels advocated by (Harrison-Broninski 2005), has shaped my view of *separation of control* that constitutes, in addition to *separation of concerns*, the core of the proposed architectural framework. He differentiates human-driven processes from the mechanistic ones and provides concepts and notations to model them. The role of this chapter is to demonstrate how these concepts are carried out further and integrated with advanced techniques in software architecture bearing in mind the benefits gained from research disciplines that were discussed in Chapter 2.

Chapter 3 has shown the 3Cs business architecture, which lays the cornerstone for the intended framework as it provides the event-based mechanism and the contract-based technology needed for the *causal* management of systems' interactions and modelling primitives (i.e. coordination context) through which systems adaptivity is ensured in a *causal* way against expected environment changes and authorised *users* interventions. This chapter aims at modelling and managing human interactions within organisational settings in a more *flexible* way by deploying concrete architectural modelling primitives that are tightly coupled with interactively dynamic concepts borrowed from different paradigms: norm-based modelling, organisational structures and human-driven processes.

4.2 The Approach in a Nutshell

This research put forward a normative architectural approach, which takes biddability of social interactions into account as a means for gaining flexibility, allows systems to benefit from endowing knowledgeable staff the power to intervene when necessary to bring about new state of affairs. This approach, specifies the system response to biddable interactions of its participants that cannot be merely controlled by consulting access lists as observed in Role Based Access Control systems (RBAC) (Sandhu, Coyne *et al.* 1996a, Sandhu, Ferraiolo & Kuhn 2000). The system's response should be reflected by changing the current system configuration in order to limit the functional effects of these interactions or support them by means of negative or positive reconfigurations—*sanctions* or *facilitations*, respectively.

The approach promotes architectural primitives that allow system specifiers to guide the system's response towards biddable interactions through a new type of architectural connectors, namely *social laws*, which can be made effective across a new type of *architectural wires* that link social and technological components. These laws impose certain architectural enablement on required interactions of participants or architectural sanctions on forbidden ones, as a result of detecting a norm-violating trigger.

Norms, unlike rules, can be violated either by an unexpected human interaction or a deviation of the system context that requires launching a human-driven process. In both cases the adaptation is gained by providing the appropriate reconfigurations to the role assigned to system participants and/or the technical components that surround them. Role reconfigurations support humans and social entities in realising their required tasks and processes within their space of capabilities, whereas technical adaptations put in place components and connectors needed for coordinating and executing a task's interactions. With regard to unwanted interactions, the proposed reconfiguration primitives should provide sanctions as a tool to suppress or allow unwanted human interactions—with a report to higher management in the latter case—while being monitored by the configuration manager, which in turn consults social laws to respond accordingly.

A system specifier may specify a law that allows an unwanted behaviour by certain role players, even if they are not permitted to do so, provided that they are capable of doing the corresponding action. However, some sort of organisational control is required to keep the system stability (e.g. send a report to the management that contains the incident details).

The novel idea in this approach is that it provides architectural specifiers with means to specify architectural primitives that tame the freedom of human participant and keep their capabilities and rational as a reserve to be utilised in sub-ideal situations, particularly when the situation's signs are beyond those specified in the causal coordination and reconfiguration levels (i.e., coordination contracts and coordination contexts). I promote architectural constructs that are required to support adapting socio-technical systems using high level conceptual patterns. These patterns will to capture interactions within institutional contexts that aim for bringing about a new state of affair. An example for such patterns is the combination of speech act and deontic operators introduced by (Johannesson & Wohed 1999).

The focus is particularly placed on research efforts that have discussed design and modelling issues related to unifying concepts among organisational theory (i.e., *roles and tasks*), speech act and deontic logic (i.e., *obligations & permissions*). Therefore, I stipulate the necessity to enlighten the reader with a review of current investigations and lessons learned in these active areas of research before delving into the nitty-gritty of the proposed approach. The subsequent subsections will provide an interdisciplinary view of the notion of roles which are apparently still in need of a consensus when it comes to an integrative definition despite thoroughly research.

4.3 Roles in Organisation Theory

Organisation theory is a discipline that focuses primarily on organisations as units for identifying common themes for the purpose of solving problems, managing resources and maximising efficiency and productivity (Kast & Rosenzweig 1970). It covers a variety of areas that generally include organisation structures and psychology. Mintzberg defined organisational structures as: "the sum total of the ways in which its labour is divided into distinct tasks and then its coordination is achieved among these tasks."(Mintzberg 1992), p. 2. Handy pointed that the study of people in organisations is far beyond certainty and predictability due to the multiplicity of contextual information and the inherit ability of human participants to disobey norms of practice (Handy 1985).

4.3.1.1 Roles & Organisational Structures

Organisational structures are key concepts in the role theory and they are considered cornerstones for developing organisational frameworks. This research agrees with Hay's view as he stated it in his book (Hay 2003), p. 30: "Organisational charts are rarely adequate to describe the complexities of human interactions in an enterprise".

Any participant in a socio-technical setting occupies a role in relation to a system configuration. His behaviour as a role player, when enacting a certain role, depends on a trio of influences:

- The forces of the organisational system's norms of codes, which include definition of processes, permissions as well as obligations that are conferred to the role(s) he/she plays in the light of his/her institutionally recognised capabilities.
- The forces of the situation, whether ideal/planned or sub-ideal/ unexpected.
- The forces of the availability of technical resources and the possible of reconfigurations that can add or remove these resources.

To some extent, these sets of influences interact collectively to affect participants' interactions—discarding selfish or destructive attitudes. The transition of the participant's current role to another member of his/her role space is always influenced by the current context. Conversely, the management of a context, particularly a sub-ideal one, would require a role transition, which reflects a participant's capabilities, to take place in order to empower him to respond to this unwanted context. The role transitions influence system configurations, and once triggered, might add/remove some hardware/software resources and manipulate the participant's permissions to access them.

4.4 Roles for Modelling Software Systems

Several notions of the role concept can be found in many research areas of software systems modelling. Among these areas that inspired this research: object-oriented modelling, software architecture and Role-Based Access Control (RBAC). All these research areas utilise the role concept in analysing behaviours and assets access demands originated from presumable role players. In this perspective, roles provide a way of allocating and qualifying tasks. The behaviour of the role player covers different levels of abstractions starting from primitive programming language interfaces and ending up with compound notions like social roles. The rest of this subsection illustrates a review of role modelling in these paradigms.

4.4.1 Object-oriented Software Systems

Object-oriented modelling and design has prevailed among software developers as a successful methodology since early nineties of the previous century. A constituent part of this methodology (i.e. Object-Oriented Programming (OOP)) does not exhibit roles as first class entities but are rather introduced as qualifiers for identity-based associations between objects. Figure 4.1 depicts how roles relate to classes in object oriented models.

The OOram software engineering method developed by (Reenskaug, Wold & Lehene 1996) was the first approach to modelling objects and objects collaborations using roles and roles models. Riehle and Gross established a framework to support large scale object oriented systems through frameworks (Riehle & Gross 1998). They focus on the problem of describing the complexity of objects collaboration as it emerges in framework design and integration.

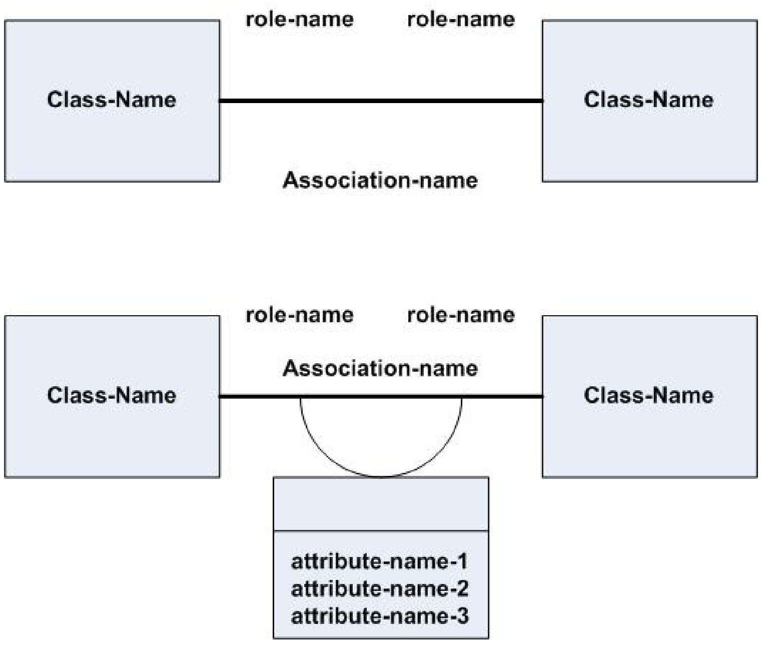


Figure 4.1 Roles in traditional object-oriented methods

Their research contribution was the first departure from the traditional view of roles in the object-oriented modelling paradigm as they express more sophisticated semantics on relationships between roles and system typed objects (i.e., classes as shown in Figure 4.2). Semantically, having an independent notion of roles can be useful to represent a wider range of contextual information, such as depicting a role in the absence of the role enactor, which models a state of sub-ideality, (i.e. (Lee & Bae 2002)).

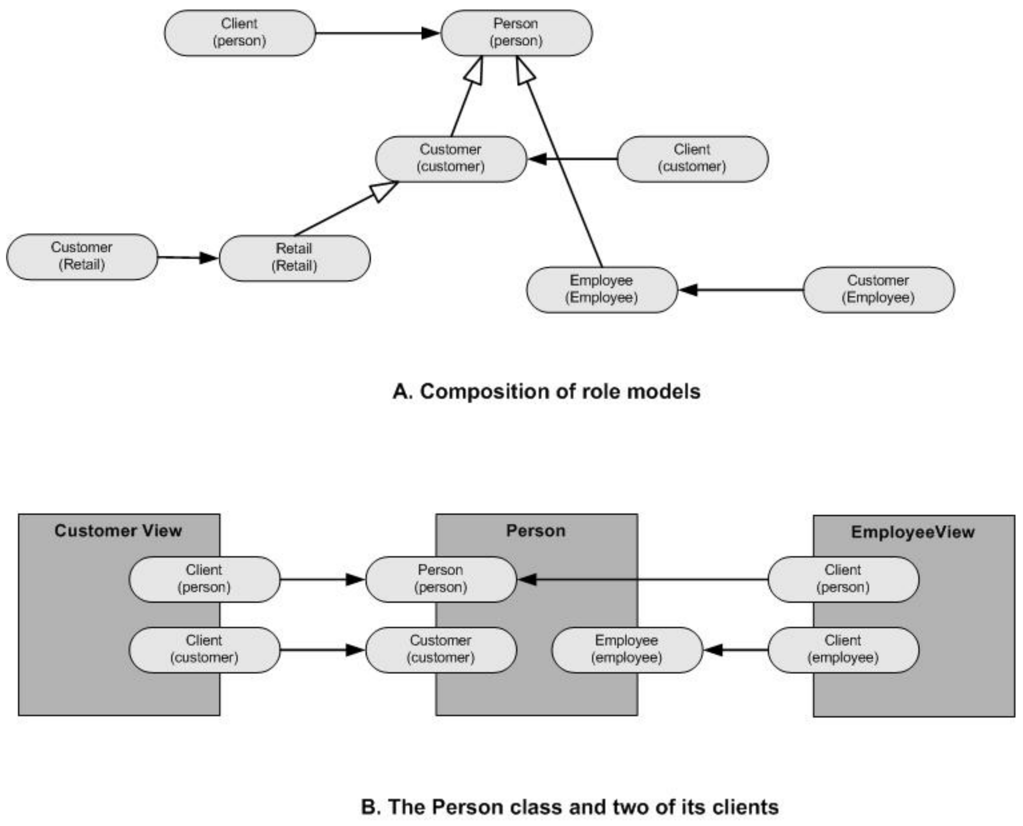


Figure 4.2 Role as first-class modelling entities (Riehle & Gross 1998)

The traditional view of roles in the object-oriented modelling/programming paradigm has been given by (Kristensen 1996): *"A role of an object is a set of properties which are important for an object to be able to behave in a certain way expected by a set of other objects"*. He also devised the characteristics of roles in object-oriented modelling, which includes:

- *Visibility*: the visibility of and access to, the object (i.e. player)
- *Dependency*: a role cannot exist without an object
- *Dynamicity*: a role may be added or removed during the lifetime of the object
- *Multiplicity*: several instances of a role have one identity

- *Abstractivity*: roles can be classified and organised into generalisations and aggregation hierarchies

Many researchers such as (Kristensen 1996, Kendall 1999) advocate the superiority of objects over roles and adopt these characterisations to use roles as proxies for filtering objects' behaviour. Thus, a role is a temporary object that demonstrates the behaviour of the original object in a context (Steimann 2000). Therefore, the role notion in object-oriented modelling is inadequate in terms of representing high abstract primitives that can be associated with organisational concepts.

Object-based structural anomalies have been identified by (Lee & Bae 2002) to provided means to express role-enacting anomalies and how to alleviate them. As a matter of fact, this research was a key starting point for the emergence of the approach at hand as it shows elements of independent role representation and role binding related violations. Moreover, role-promoting variations of object modelling, which aim at modelling key features of software agents e.g. proactivity and autonomy, cannot be overlooked. The work presented by (Depke, Heckel & Küster 2000) has led this trend and encouraged a novel representation of roles for fine-grained modelling of objects and agent by means of interaction protocols as well as evolution of agent structure and behaviour. However, their definition of role at runtime still exhibits life-time dependency on agents. Additional comments on this work can be found in chapters 5 and 6.

The research on constructing development techniques and methodologies for organisations and organisational structures will continue to be a major research topic for many years. Moffet has introduced a hierarchy that is based on organisational control principles (Moffet 1998). This approach has been adopted by the founders of Ponder in (Lupu & Sloman 1999) to deploy management principle (i.e. monitoring and control) through policies specifications over elements of distributed system networks. More details on this approach are available in the next section.

4.4.2 Roles in Agent-based Systems

One of the early steps toward agent-based modelling has addressed the requirement specification level, namely the *i** framework for enterprise modelling (Yu & Mylopoulos 1997), which was one of the earliest attempts to promote roles as first-class citizens. However, many modelling approaches have been developed to represent the organisational societies in Multi-Agent Systems (MAS) (Ferber & Gutknecht 1998, Dignum, Meyer *et al.*

2002, Esteva, Padget & Sieera 2002, Dignum, Vázquez-Salceda *et al.* 2005). These techniques aimed at yielding social, administrative and business control on autonomous and proactive agents. Chapter 2 states clearly the similarities and differences between autonomous and biddable elements in socio-technical settings. Agent based methodologies such as (Cuesta, Gómez & Rodríguez 2003) and the Gaia model (Zambonelli, Jennings *et al.* 2003), in addition to modelling techniques like OMNI (Dignum, Vázquez-Salceda *et al.* 2005) and (Esteva, Padget *et al.* 2002) have contributed to the advocated approach.

4.4.3 Roles in Access-based Policies

Access control modelling has become of a great importance as a key technique to enforce security policies and hence protecting technical elements, i.e. system resources, from unauthorised access by users or computational agents (subjects). It has been claimed by (Edwards 1996) that most of the policies that are found in an organisation can be captured by access control-based models. Control over system reactions to biddable interactions can be achieved not only by access control primitives but also by incorporating objects to which access is constrained. Edwards devised three requirements for any successful access policy-based access control system: (1) expressiveness to capture a wide spectrum of policy considerations, (2) flexibility to handle collaboration, (3) integration with information from and about “the real-world” context. Sandhu and his colleagues (Sandhu, Coyne *et al.* 1996a) have established the Role-based Access Control model as a means of enforcing security by assigning permissions to defined roles instead of ascribing them to users directly. They conceived roles as a means to define positions in organisation, bundling responsibilities or representing capabilities. Among other compelling features of RBAC, roles can be naturally organised in hierarchies, and policies can specify various constraints patterns e.g. Separation of Duties SoD) due to its policy-neutral feature. Figure 4.3 depicts the main elements of a standardised RBAC framework.

Roles may be assigned to several participants and participants may play several roles at the same time, however, no notion of state is present in this model. Very few research efforts have contributed to fill in this gap by explicitly adding the state and state transition notions e.g. (Steinmuller & Safarik 2001),. However, in this work state transitions referred to changes of user-permission associations for a single control policy rather than changes in the configuration of access control policies themselves which were assumed static.

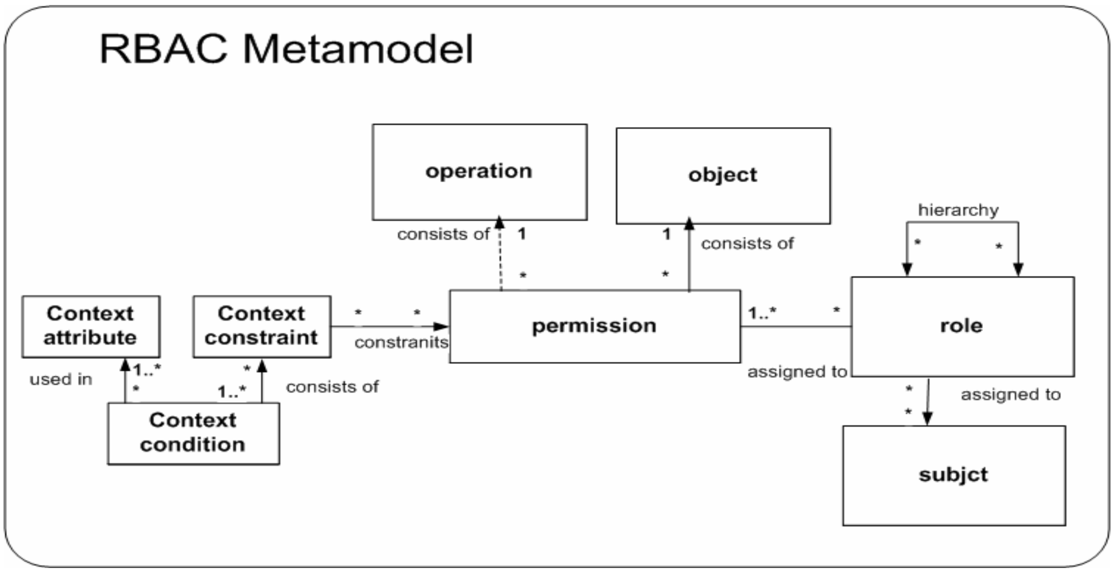


Figure 4.3 The RBAC meta model (Sandhu, Ferraiolo *et al.* 2000)

There are significant variations and interpretations of RBAC concepts. These variations resulted in several RBAC models with different levels of sophistication and internal modelling dialects (Crook, Ince & Nuseibeh 2003). A unification attempt was made by developing the NIST framework (Sandhu, Ferraiolo *et al.* 2000) to reach a common standard for RBAC family of models based on combining sequences of models by adding capabilities in a progressive way. The key feature of NIST is that it provides a means to inherit selectively (i.e. dynamically) permissions from junior to senior roles through consulting an associated activity hierarchy. For example it is not possible for a senior role to inherit permissions from a junior role unless they are both activated.

RBAC is only concerned with the *causal* filtering of information and access to resources against users' computer-mediated attempts; therefore, as a conceptual model, it lacks the comprehensive way of modelling interactions particularly those that cannot be physically filtered or prevented (i.e., accessing machines or equipment). The approach advocated by this thesis aims at bringing new state of affairs e.g. adding/deleting resources or putting in place new access control policies to handle the situation at hand.

In summary, despite of the RBAC's shortcomings, it still captures focus in this thesis. This is because it is a policy-neutral security mechanism that poses roles as a first-class design entity and provides the ability to restrict collaborative interactions. These features makee the research findings of the RBAC community suitable for cultivating the 3Cs approach in terms of their various hierarchical structures, role dependencies and constraints patterns.

4.4.4 Roles in Organisation-oriented System Modelling

This section presents a review of agent-agnostic organisational system modelling approaches that build on organisational structures and utilise organisational theories. A key research work of Bacon et al. (Bacon, Lloyd & Moody 2001) demonstrates how roles based on function and seniority can be combined. In order to be assigned certain roles, a user must have been assigned other prerequisite roles; for example, a doctor can only be assigned the role of a *senior gastroenterologist* if the roles *senior doctor* and *gastroenterologist* have already been assigned to him.

A further step towards role-based design, was taken by (Crook, Ince & Nuseibeh 2005), who borrowed a taxonomy of role types from (Mintzberg 1992) (i.e., market, functional and seniority) as prerequisite proxies for accessing an operation or set of operations in the sense RBAC but at a higher level of abstraction compared to its counterparts. In Crook et al., the access policy is modelled as a ternary relationship between role sets, set of operations and an asset category. Their framework includes three types of roles: *functional*, *security* and *contextual*. These policies are related to information assets only.

For instance, their example demonstrates a regional branch of a retail bank where bank tellers have access to accounts of the customers of that branch only. Under no circumstances can access be gained to the accounts of customers in other branches. Therefore, the bank outlet represents a context, which has to be assigned to both the account and the bank teller in order for access to be granted.

With regards to higher level methodological approaches, a few research attempts such as (Odell, Parunak *et al.* 2003) introduced the analogy of *roles* and *players* which has been further developed by (Colman & Han 2007) to provide an organisation-centric view of roles. Another view of modelling interactions of human processes within organisations has been promoted by (Harrison-Broninski 2005), namely Human Interaction Management (HIM), which supports *human-driven processes* with an analytical framework which models interactions and speech act by means of Role-Activity Diagrams (RAD). RAD's basic concepts have been introduced by (Holt, Ramsey & Grimes 1983) and later enhanced by Ould's variation. HIM plays a key role in the development of the proposed methodological approach as it supports the extension of the 3Cs architectural approach. This methodology will be explained in the next Chapter.

This thesis, will present an approach to managing interactions within organisational settings that exploits lessons learned from the aforementioned approaches. The specific adoption of a hybrid role construct, which combines the characteristics of functional and contextual roles, will be shown by examples. The next section explicates the details of the advocated architectural approach (El-Hassan & Fiadeiro 2006, El-Hassan, Fiadeiro & Heckel 2008).

4.5 The Role Model: The Ternary of Roles, Norms and Reconfigurations

Reasoning about human interactions within organisation settings requires high-level system modellers to be aware of the effect of biddable human interactions, particularly at sub-ideal situations, in order to come up with norm-based reconfigurations. Those will take the form of architectural connectors to be put in place, monitored and activated by the configuration manager. Norm and norm-based policies have been mainly explained in Chapter 2.

Making a clear and unambiguous model of these situations permits enacting tasks and possibly processes that are normally unauthorised in normal contexts where the coordination layer is causally controlling interactions through coordination contracts. This could be achieved through enacting capabilities that are not institutionally authorised or that override the contextual pre-conditions of the existing superposed contracts through new architectural primitives: i.e. social laws and roles, which provide flexible norm-based control of social interactions that support collaborative behaviour.

The advocated flexibility can be achieved by tailoring a role-modelling technique to establish a new perspective (i.e. the process view) which would include tasks within its definition. Tasks, as explained in this subsection, provide a process-centric view of certain interactions (i.e. speech act-like interaction). These interactions specify communicative acts that are signified by both human participants and the reconfiguration manager as a request for initiating a task, and demonstrate the intersection between:

- The role that is currently played or intended to play by the social component compared to the role requirements (i.e. qualifications and permissions).

- Norms that govern whether there is an opportunity to complete the task at hand successfully[3] or not.
- The ideality of the context in which task's initiation takes place.

Figure 4.4 sketches such a kind of intersection, which requires further elaboration conceptually and functionally.

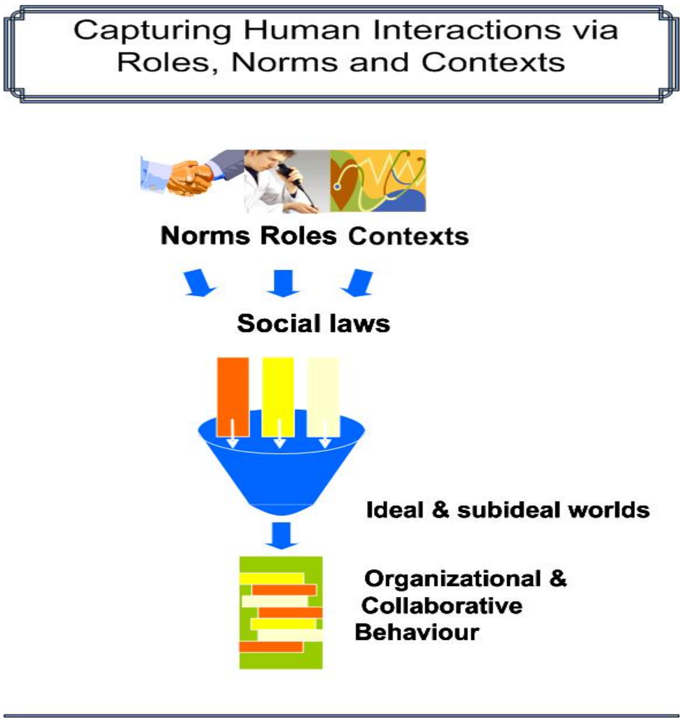

Capturing Human Interactions via Roles, Norms and Contexts
Norms Roles Contexts
Social laws
Ideal & subideal worlds
Organizational & Collaborative Behaviour

Figure 4.4 Role, Norms and Reconfiguration

Herein, the assumptions of the proposed architectural framework for managing and reasoning about biddable interactions are stated. The framework is inspired by several research efforts: (Mintzberg 1992, Moffet 1998) which present the perceptions toward organisational structures, (Crook, Ince *et al.* 2003, Crook, Ince *et al.* 2005) for proposing a hybrid approach to role modelling and the (HIM) approach (Harrison-Broninski 2005) for its process-aware view. Any individual in any situation occupies a role in relation to a system configuration.

- His/her performance in that role will depend on a trio set of influences:

3 A task is considered completed once its entry-operation succeeds not only in calling the intended service but also in bringing about ,through social laws, the configuration that is required for all its subsumed interactions (i.e. task's members).

- *The internal forces*: role attributes, skills ascribed to this role, and possible role transitions from that role.
- *The contextual forces*: whether ideal/planned or sub-ideal/ unexpected
- *The normative forces:* obligations, interdictions and permissions

Figure 4.5 represents a *static* and a *holistic* view of the core elements of an architectural framework that extends the 3Cs framework presented in the previous chapter. It shows *what* the framework's architectural concepts are and *what* connections map them to each other at design time and runtime. This initial view does not state *how* these concepts and their connections are maintained and reconfigured to keep overall good behaviour of the system at hand.

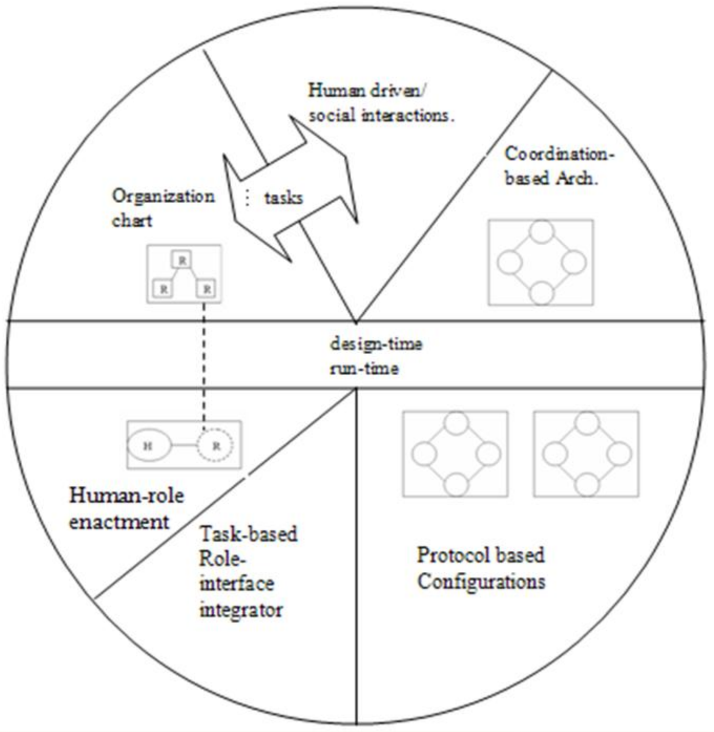


Figure 4.5 Holistic View of Extended Framework

The upper half of the Figure 4.5 shows the high-level concepts that can be used as meta data to reason about the runtime entities in the lower half of the view. These concepts relate to highly abstracted notions such as organisational structures and processes to equate components behaviour to patterns of organisational control. In other words, when a meta data (e.g. role element) is instantiated and connected at runtime to a component (e.g. social component), it can be used as a means for supporting the inspection and the modification of the properties and the structural interconnections of this component (e.g. reasoning about the capabilities of the social component can support its interconnections with equipment). The hypothesis is that this framework can be used by system specifiers to model and analyse a new type of architectural connectors that can handle system response to unexpected interactions or contextual change within organisational settings (i.e. social laws). The case study, which will be explained through out the rest of this chapter, demonstrates the proposed architectural concepts in the introduced framework.

4.5.1 A Motivating Case Study

This example was extracted from a survey undertaken at a Gastroenterology department[4] in the UAE. By reviewing their documentation and interviewing the department staff, the researcher elicited a group of norms that affect the behaviour of doctors:

(1) No operation can be undertaken without the patient's permission
(2) Surgical intervention should be carried by *surgeons*[5] only
(3) In the case of emergency, a doctor may commit simple surgical interventions if surgeons are not available and in a life-threatening situation

For example, consider the social law that applies to minor operations. Such procedures involve a social role—a *GP*—who is the anchor role in the sense that the social laws will apply to the actions performed by instances of this role, e.g. a gastroenterologist. In addition, three coordination interfaces are required to ensure that the *GP* interacts with the right components: the device that is monitoring the procedure—*monitor-procedure* and the software component that provides access to administrative data—*administrator*. In the configuration of the system, there will be coordination laws modelling the way these components interact. Chapter 7 will provide more details about the proposed reconfiguration language that is used within social laws constructs and will demonstrate qualitative evaluation of the language properties. However, some excerpts from the case study will be used throughout this chapter to explain the architectural primitives and the underlying reconfiguration language.

[4] A specialised medical unit at a government hospital, Dubai, U.A.E., which provides treatment for digestive diseases. The reader may refer to Chapter Seven for more details

[5] Readers should refer to Figure 4.6, Figure 4.7 and Table 4.1 for roles, tasks and permissions

4.5.2 Social Roles

The focus of the presented approach to socio-technical systems is on modelling the technical impact of incorporating people within systems. Such an approach incurs having mechanisms to combine human autonomy, capabilities and responsibilities through *role* models. It is clear that, in order to reach a viable role modelling technique that integrates well with the 3Cs architectural primitives and allows modelling higher levels of abstraction, research outcomes that have been discussed earlier in this chapter have to be considered. Therefore, this chapter also recalls and combines the literature review concerning these paradigms with earlier discussion in Chapter 2 particularly those handling human interactions in deontic frameworks, e.g. RNS[6] patterns (Nickles, Rovatsos *et al.* 2002).

Roles are abstract development constructs that specify the expected behaviour of social components by means of operations and ascribed normative aspects that refer to certain institutionalised positions or capabilities. More concretely, this approach distinguishes between having the ability to perform an operation and having the qualification or authorisation to do so: a social component may have the ability to perform an operation and still trigger a role violation if it is not an instance of a role that has the right qualification. Herein, the qualification term refers to the fact that an organisation has empowered the social component to perform given operations.

A role hierarchy is a tree of special types called role types. The root of this tree defines a role with the least capability and permission yet universally generalises the shared tasks' definitions among its descendants. The other nodes inherit tasks from their parents as they are or redefine them.

As discussed below, the execution of operations by a component when playing a role without the required qualification is governed by a social law. A social law specifies (social) rules that either impose sanctions or provide a configuration in which the operation can be safely executed, depending on the context in which the violation takes place. However, it is worthy to stress that the execution of operations, even by qualified components, can be governed by coordination laws and, as such, can be refused in certain circumstances for operational reasons, not deontic ones.

Hence, the following general structure of a social role is sufficient to describe a social law's details:

[6] RNS: a term that stands for Roles, Norms and Sanctions

```
social role rolename {specializes rolename}
types {{par}+:datatype}*
operations {
{'[+]'} opname {⊃ opname}
}*
```

A social role should have a unique name and has a Java-like way of modelling single inheritance to construct hierarchies. Types can also be declared to serve for the forthcoming operations' parameters. The social role primitive allows declaring a set of operations that individually label an entry or an exit of a task. Each operation, particularly entry operations, might be preceded with a modality sign that explicates capabilities and permissions of enacting this task. I denote operations for which the role is qualified with [+]. The following table shows a set of modality signs:

Modality Sign	Operation Syntax	Semantics
empty	Not mentioned/empty	Enacting the operation will never be allowed. Capturing such a biddable interaction would lead to severe sanctions (out of the scope of this thesis)
empty	opname	The operation is part of the player's capabilities, but the permission is not institutionally granted. The capability is inherited to children as is. Such an operation labels *declared tasks*.
+	[+] opname	The permission pertained to this action is institutionally granted and is inherited to children. Such an action labels *defined tasks*
–	[-] opname	The institutional permission is revoked from the current role and downwards (children).

Table 4.1 Modality Signs for Operations

The subsumption relation between operations can be defined as follows: by declaring $op_1 \supset op_2$ it means that op_1 can only be executed as part of op_2, in which case a component qualified to do op_2 is also qualified to do op_1. For instance, *GPs* are qualified to perform routine tasks of seeing patients and registering for shifts in wards. A *GP* can also perform minor operations but such an interaction will trigger a role violation unless (s)he is an instance of a role that is qualified to do so. A *registrar_sugeon* role inherits the permission and capabilities of a *GP* and his role should redefine the permissions regarding performing minor operations (see Figures 4.6 and 4.7).

```
social role GP
types p:patient,op:operation
operations
   [+]seePatient(p)
          collectData(p) ⊃
          checkBloodPressure(p) ⊃
      minorOp(op,p)
```

```
social role registrar_surgeon
specializes GP
types p:patient,op:operation
operations
     [+]minorOp(op,p)
            setupMonitor(op,p) ⊃
        majorOp(op,p)
```

Figure 4.6 An example of social roles

4.5.3 Social Tasks

In human-driven processes, processes cannot be simply described as predetermined sequence of tasks (Harrison-Broninski 2005), unlike automated and mechanistic processes. Within the execution of a process, the human participant should be capable of redefining the sequencing and sometimes even overriding the pre-conditions for tasks enactments.

Thus, the advocated approach promotes tasks as building blocks of social roles, as first class design citizens and as conceptual units of work that group a set of interactions. Additionally, the interactions that label the inauguration (i.e. the initiation) and the departure from these tasks, particularly those to which social and organisational meanings can be assigned, should be emphasised. In other words such interactions, namely *entry* and *exit* operations, could be perceived by monitoring systems or other participants.

Herein, the approach follows the communicative act approach, in the sense of (Searle 2002) and (Castelfranchi & Giardini 2003), that highlights acts, which are signified by other humans or monitoring systems as a means to convey social or organisational messages, e.g. "let us start this particular process". I stress, herein, that entry/exit operations are original constituent parts of the required task that trigger a service in the intended software/hardware component and not just mere behavioural signs or flags. Chapter Two gives details about communicative acts theories. There are many research efforts describing suggestions that fit tasks into organisational roles. Among which, (Garlan, Siewiorek *et al.* 2002, Sousa, Poladian *et al.* 2005) capitalise on user tasks to guide self-adaptation.

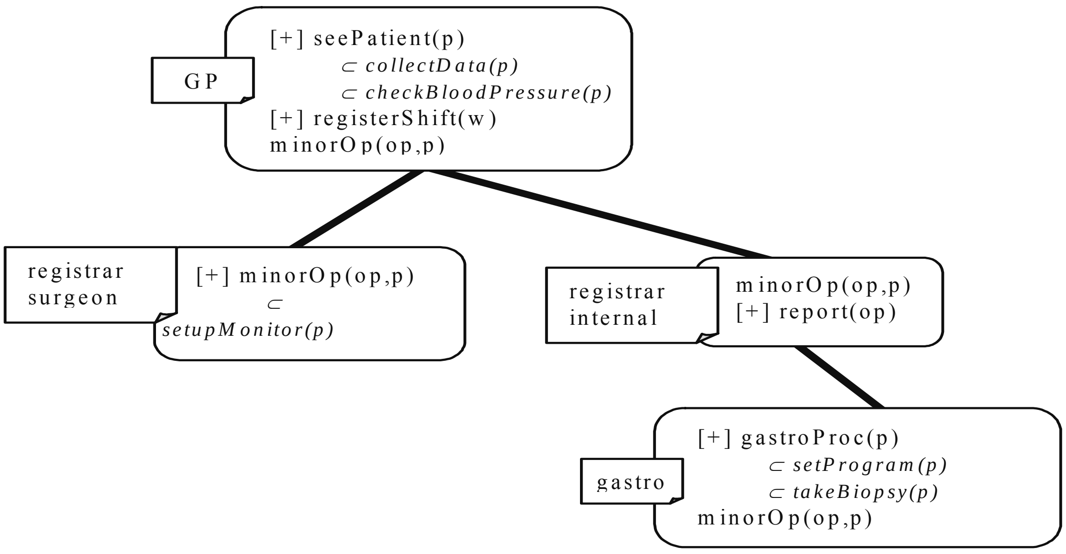


Figure 4.7 A role hierarchy example (with task inheritance)

4.5.4 Social Laws

Social laws are the primitives that the current approach proposes for controlling social interactions. Generally speaking, social laws represent new type architectural connectors that facilitate changes to architectural configurations. Social laws share many traits with policies in access control based frameworks, which restrict the accessibility of a system's *users* towards the system resources based on information about the environment context and the role enacted by the user. Social laws surpass the security view of RBAC models and capture normative aspects of collaborations between the system's participants and other technical components using deontic-like concepts (i.e., obligations and interdictions), which are applied onto actions once they are performed by social components (i.e. role player). Social interactions cannot be controlled by physical causation, as prescribed for coordination laws, but rather by a combination of monitoring, flexible role models and norm-based reconfigurations.

If the inclusion of the organisational dimension is considered, on the one hand, and the biddability of social components of the system, on the other, a gap needs to be filled in order to be able to reason about the behaviours that emerge from the collaborations within a system. The notion of collaboration that is adopted matches the work of (Barbuceanu, Gray *et al.* 1999): *"The coordination in organizations and societies cannot be accounted for without considering social laws of the organisation and the way they constrain the behaviour of individual agents".* It requires highly integrated and flexible laws between people as well as processes and technological components capable of governing the interactions that emerge according to the policies of the organisation. In order to make such concepts applicable, social laws need to incorporate three main components: roles, norms and sanctions.

The proposed approach also builds on normative positions as formalised by (Jones & Sergot 1996, Sergot 1999), which represent all logically possible (normative, control and influence) relations between roles in a certain configuration. I argue that human biddability can be modelled as a set of normative positions that apply to the set of roles involved. More details about the foundations of normative positions have been discussed in Chapter Two and the impact of adopting such concept on the overall methodological approach will be demonstrated in Chapter Five.

4.5.4.1 The Language Design

The trade off between retaining the expressive power and the clarity of the specification language of social laws has been managed carefully. Since social laws are interpreted by the configuration manager, their main clauses, particularly (re)configuration statements, should be clear and unambiguous allowing deterministic interpretation. However, some sort of ambiguity has to be pertained to the instantiation of the anchor role type to support the dynamic role binding mechanism—akin to object-oriented programming—that should be resolved at runtime when role instances are instantiated by their players. This leads to the corresponding social law capturing a trigger from them. Otherwise, the system modeller/architect should specify a social law for every candidate social role.

Additionally, the configuration manager can exploit the current role degree of the instantiated role type with respect to the role hierarchy to resolve conflicts between overlapping laws. Consequently, if a social component triggers two social laws, through the same interaction label and it can instantiate the anchor role of both laws then the law with the more specialised role would prevail. Such conflicts would be resolved regardless of the domain of the application. Hence, it can be argued that this approach maximises the likelihood of modality conflict resolutions (i.e., domain independent) compared to goal conflicts ones (i.e., domain-dependent). This taxonomy of conflicts has been promoted by (Lupu & Sloman 1999).

4.5.4.2 Sub-ideal Contexts

A social law defines what actions should be taken when an operation is initiated by a social component acting according to a given social role— the *anchor* role of the social law—which is not qualified to do so. That is to say, social laws provide a context for the system to react and adapt to a sub-ideal situation. The reaction can consist of either the imposition of sanctions or a reconfiguration of the system. The latter can be performed so as to put in place a context in which the social component can proceed with the operation in spite of the fact that it is not qualified, for instance, a doctor having to perform a minor operation in a life-critical situation. For this purpose, new equipment and/or social components may need to be added to the system configuration to assist the doctor, also software components that control the system may need to be reconfigured to enable the doctor to perform operations that, in normative states, should not be enabled. This sort of reaction captures what is sometimes called the role-binding anomaly as described in (Lee & Bae 2002). Social laws are put forward to handle role-binding anomalies but at a higher level of abstraction that can be easily mapped to instances of organisational roles that are ordered in hierarchies.

Another situation in which a social law allows detecting a violation is when an operation is initiated in a context in which it is not permitted according to some organisational norm. For instance, although a surgeon is qualified to perform a minor operation, the rules of the hospital are such that the consent of the patient is needed before initiating any operation. However, in a life-critical situation, it may be impossible to obtain consent. Yet, in spite of this, the surgeon should be allowed to proceed. In this case, a reconfiguration should again be triggered, implying a change in the structure of the system in terms of adding/replacing components and/or coordination contracts.

Social laws have the following general structure:

```
social law name
anchor role social role
partners
   {social role, coordination interface}*
types {{par}+:datatype}*
{violation rule
```

```
     when trigger
     if condition
     reconfiguration task
      sanction {operations}*
}*
```

Besides the anchor role, a social law identifies other partners through either social roles or coordination interfaces. The former are useful for reconfiguration operations and the latter is useful for both detecting triggers and reconfigurations as explained below.

There are three kinds of triggers for specifying violation rules: (1) operations of the anchor role, which are executed by social components that have no qualification; (2) operations for which the anchor role is qualified but are initiated in a context in which they are not permitted; (3) operations of the anchor role that are not executed in contexts in which they are required. The first takes the form:

```
unqualified operation
```

The second takes the form:

```
operation and not enabling state
```

The third are of the form:

```
active state and not operation
```

Notice that in order to detect a violation of the enabling state (permission), a coordination interface is required to provide an operation that returns a boolean value and, in order to detect the violation of the obligation, another coordination interface has to provide an event. The "negated operation" holds in the states in which the operation has not been scheduled for execution by the social component that instantiates the anchor role. Generally speaking, the definitions of permission and obligations and their relation have been informally adapted from (Boella & van der Torre 2003). Sanctions are used when the violation cannot be handled through a reconfiguration and it requires, instead, punitive actions to be taken, possibly with the assistance of system stakeholders identified as partners. In the above-mentioned example this could be the unit's head of staff or the hospital *QA* manager.

Following the Gastroenterology Unit example (Section 4.5.1), consider the social law that applies to minor operations. Such procedures involve a social role, a *GP*, who is the anchor role in the sense that the social laws will apply to the actions performed by instances of this role. In addition, three coordination interfaces are required to ensure that the GP interacts with the right components: the device that is monitoring the procedure –

monitor-procedure, and the software component that provides access to administrative data – *administrator*. In the configuration of the system, there will be coordination laws modelling the way these three components interact. Chapter 7 will provide a comparison between the provided social law example and its corresponding coordination laws as an evidence of the expressiveness of the social law description language.

```
social law minor-operation
anchor role d:GP
type p:patient, op:operation
partners
      a:administrator
      m:monitor-procedure
when d.minorOp(op,p) and not a.ensureConsent(op,d,p)
      if m.alarm(p)
      reconfiguration reconfCoord(d,op)
      sanction a.record(d,op,"no_consent")
when unqualified d.minorOp(op,p)
      if m.alarm(p)
      reconfiguration reconfUnqual(d,op)
      sanction a.record(d,op,"unqualified")
```

The social law has two rules triggered by the same event: the moment in which the doctor initiates the operation on the patient. The first rule handles the situation in which there is no record of consent given by the patient to perform that operation. If the monitor detects that there is an emergency situation, then a reconfiguration of the context is performed to put in place the components and coordination contracts that are required for the operation to proceed. This may involve, for instance, providing access to further information registered on the patient's file, say on allergies. However, if the monitor does not detect an emergency, sanctions apply by recording the violation in the doctor's file.

The second rule is activated if the actual doctor is not qualified to perform a minor operation, which is possible because the doctor's role matches one of the roles in the non-surgical branch of the doctors' role hierarchy: *GP*, registrar internal or gastroenterologist. In this case, the monitor has to distinguish again if there is an emergency or not. For simplicity, I used the same alarm condition provided by the monitor. If an emergency is indeed detected, a reconfiguration of the context is performed to allow the doctor to proceed, for instance unblocking actions that, in normative states, should be forbidden to the doctor. Otherwise, sanctions apply. Notice that the reconfiguration operation takes the doctor as a parameter: the hospital may have different rules about the context that should be present during an operation depending on the type of doctor.

Notice that both rules can apply—the doctor may not be qualified and the patient may not have given consent. In the case of an emergency, both reconfigurations may apply; otherwise, both sanctions are implemented.

The explanation of the operational part of the reconfiguration language which explicates generic reconfiguration tasks or operations e.g. *reconfCoord(d,op)* and *reconfUnqual(d,op)* will be presented in the next chapter (i.e. Chapter 6). These configuration tasks advance the primitives that have introduced in (Andrade, Fiadeiro *et al.* 2002) for the reconfiguration language used in 3Cs and follow the formalisation techniques of (Wermelinger, Lopes *et al.* 2001) in yielding a semantics of reconfiguration based on GT. The subsequent chapter provides a formalised approach to modelling and reasoning about social interactions and their corresponding technical and role-based reconfigurations, in a way that ensure the overall good behaviour of the system in sub-ideal contexts.

4.5.4.3 Summary

This section presents the core of the contributions of this chapter with respect to the introduction of new architectural modelling primitives that address the collaboration between biddable human interactions and technical components within organisation settings at higher-enough level of abstraction. New primitives: social laws, social roles and tasks are put forward and separated from the ones presented earlier in the 3Cs framework such as coordination contracts and interfaces. These primitives consist of structural elements that can be sufficiently explicated in terms of a language-agnostic specification language yet they embody the elements from which the operational semantics of reconfiguration from which the stability-preserving behaviour emerges.

4.6 Social Laws vs. Coordination Contexts

Social laws should be concerned with human-driven organisation processes, with which technical and human resources are associated (i.e., the norm-based level). These processes require specifying possible reconfiguration to handle the empowerment of social entities to realise unexpected activities. Conversely, the coordination level codifies causal evolutions related to routine tasks: business concepts, scheduled procedures and life cycles transitions of the system components from the business point of view.

As a matter of fact, social laws enable experts, high-level managers (performance and quality) oriented managers to supervise social interactions and human-driven processes in a flexible way, whereas technical system specifiers often focus on routine and mechanistic processes. Having this combined approach with separation of control would allow system participants to concentrate on their work, exercise their powers rightly in their own province (i.e., role space) and interact with their technical surroundings more effectively.

4.6.1 Separation of Control vs. Separation of Concerns

The separation of control, as a concept, has been borrowed from Harrison-Broninski's approach (Harrison-Broninski 2005) which distinguishes, with regards of organisational processes, roles that inherent the sponsorship of a process (i.e. executive control) from those that perform day-to-day supervision (i.e. management control). By modelling norms—as social laws—the target is not to integrate them in terms of contract compositions with the coordination layer. It would result in unnecessary complexity when analysing such a relationship by means of a translation or a refinement process because the composition of contract between different types requires continuous consistency checking, and management of laws priorities. Instead, this approach adopts an interaction-centric mechanism based on a multi-level event-based system and context information to decide the layer in which the interaction at hand should be managed.

In the case of triggering social laws, normative positions patterns organise priorities and identify how to go about the unexpected social behaviour upon which more radical interventions are required. In this case, the result would be either to empower the targeted social entity or to impose sanctions on them based on certain contextual information that devise the ideality of the situation. Therefore, I advocate a framework that comprises two levels of abstraction with respect to the process-aware adaptation of systems:

- Higher level: manages human or social component driven interactions that normally refer to human-driven processes
- Lower Level: supervises lower level changes related to business concepts and routine or scheduled processes

It is worthy of note that a process enactment by a participant can be managed at the lower level reconfiguration approach (i.e. coordination context) provided that it has been previously scheduled (i.e. coordinated) as part of the participant's expected and permitted transitions from one configuration to another (ad hoc reconfiguration) . Conversely, the same enactment can be treated at the higher reconfiguration level (the norm-based approach) if the enactment is unexpected. Additionally, the norm-based reconfiguration mechanism has the ability to impose *directed obligations* on participants, exhibit these obligations in terms of facilitating appropriate roles and technical requirement and then respond to the participant's reaction accordingly. Directed obligations as explained in (Tan & Then 1998) are directed obligations from one agent, called the bearer, the configuration manager, to another agent, i.e. the human participants who fills in the role slot.

Social laws give the modeller what (s)he requires to ensure that sub-ideal situations can be recovered despite the inevitable drastic adaptations made along the way by participants or by the configuration manager without losing control over the system. A key feature of the proposed norm-based reconfiguration is that it facilitates or enables interactions (as a means of conferring an obligation towards these interactions). Therefore, they can be used as a flexible, dynamic and powerful tool for communication between system participants in the sense of the Behavioural Implicit Communication approach (BIC) that has been introduced by (Castelfranchi & Giardini 2003) as explained in Chapter 2. Manipulating the existing configuration by making roles, machines—e.g. by flashing its built-in enablement light—and processes available at runtime would be taken more seriously than a notification appearing in a user interface. In fact, changes to the configuration are necessary to support the dynamic nature of human interactions to make the most of their rationale.

However, system specifiers who encode scheduled processes and routine practices through coordination contexts and contracts fail to consider, at design time, sub-ideal situations that may emerge due to unexpected human interactions. For instance, there is no provision to exert *social control* to motivate human participants to internalise a required interaction in response to an imposed obligation. Such control mechanism is also necessary to put into effect a prohibited interaction, which is originated from a capable yet unauthorised role player, who internalises the interaction to alleviate a captured sub-ideal context. For example, in a hospital setting, a *non-surgeon doctor* may wish to qualify a piece of evidence: (e.g. a patient showing vital signs), to proceed with a surgery that is usually beyond his permissions. What is needed here is to allow the reconfiguration mechanism to rewrite and maybe weaken the pre-conditions pertained to the enactment of the surgery task.

More concretely, the architectural framework should distinguish between three types of interactions—instantly coordinated, casually-reconfigured and norm-based reconfigured— and between the related architectural constructs of coordination contract, coordination contexts, and social laws and roles.

The separation of concerns is still maintained between coordination and configuration management, as defined by (Hursch & Lopes 1995) and as adopted in the original 3Cs framework (Andrade, Fiadeiro *et al.* 2002). However, when it comes to the management of the two reconfiguration mechanisms (i.e., coordination contexts and social laws), the approach adheres to the concept of separation of control as specified in (Harrison-Broninski 2005) and explained in Section 5.3.2. Figure 4.8 shows a generalised taxonomy of interactions with regards to architectural configuration from both the technical view (i.e. coordination context and partners) and the normative one (i.e. human components associated with their role space).

4.6.2 Modelling Social Laws

This subsection discusses the rationale behind several design decisions that were made in the course of developing social laws in this framework.

4.6.2.1 Pull vs. Push Modes of Social Laws

The above description of social laws is based on a blend of two modes: *push* mode and *pull* mode. In the Push mode, the configuration manager enables an interaction by providing the role and technical facilitation to the obliged party: the social component that is compelled to enact the required task by performing its entry operation. The configuration manager interprets the social law and makes an implicit directed obligation by preparing the technical and organisational grounds for the obliged task. In other words, the configuration manager pushes the message to the obliged party (hence the name push model).

However, there is one problem with this kind of models: mitigating sub-ideal situations, they need to keep the expected control of performance and preserve flexibility. Thus, they require architectural primitives that are capable not only of reasoning about role relationships and permissions with regards to interactions, but also about the capabilities of the participants who are enacting the system.

Suppose role A is ideally fitted by Bob and an urgent task of that role is required where Alice, who has the required qualifications to do the task but has not been entitled to that role yet, is the only available person at the given context. It would be very useful to have a model that distinguishes between authorisations and qualifications in such situations. If this urgent task is badly needed for the sake of the organisation and its context then it can be sufficiently captured by the configuration manager's monitoring mechanisms. Thus, it would be very beneficial not only to alert Alice but also to translate this alert in terms of facilitating the required technical elements for this task and be able to act upon ignoring such an enabled context by Alice (i.e. directed obligation). The main problems with the Push model are: (a) defining the "minimal set" of the context parameters, (b) guiding humans when there are several recovery alternatives that crosscut a single defined "sub-ideality" and (c) dealing with temporal properties of both social components and configuration manager responses.

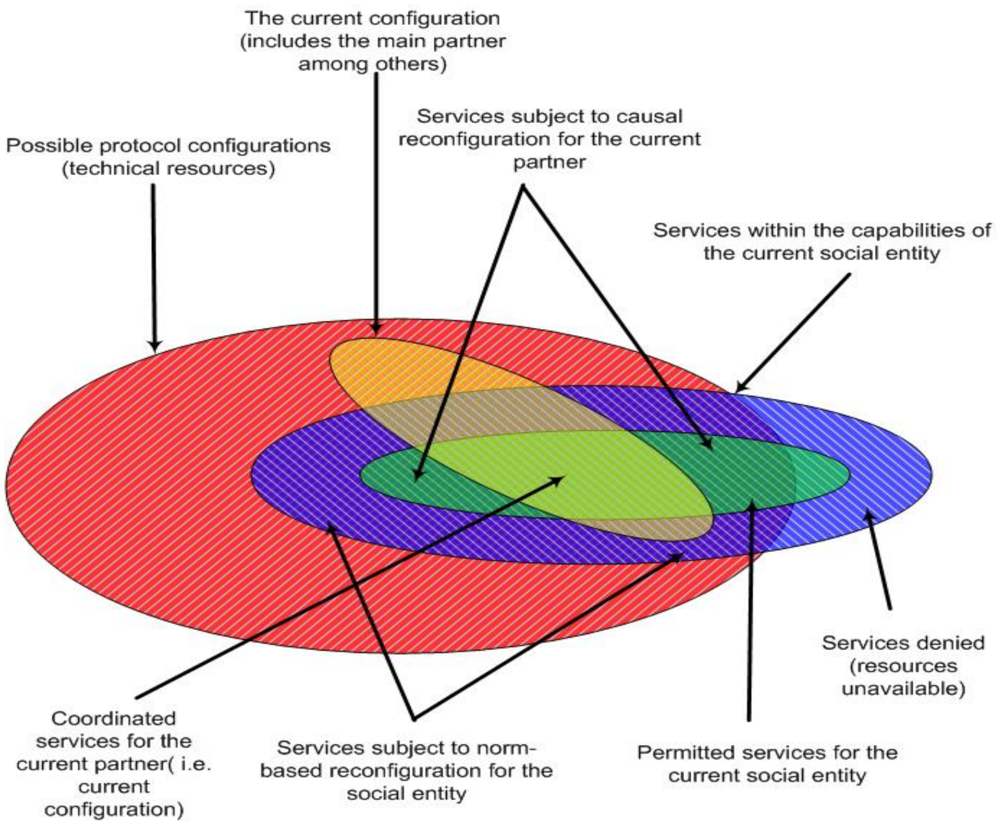

The current configuration (includes the main partner among others)
Possible protocol configurations (technical resources)
Services subject to causal reconfiguration for the current partner
Services within the capabilities of the current social entity
Services denied (resources unavailable)
Coordinated services for the current partner(i.e. current configuration)
Services subject to norm-based reconfiguration for the social entity
Permitted services for the current social entity

Figure 4.8 The space of interactions that emerges from extending the 3Cs business architecture

The *pull* mode favours social participants and gives them the upper hand due to their incomparable cognitive power over monitoring agents and mechanism that a social law relies on. For example compare the short-sighted and the pre-determined contextual set of information stated in the body of a social law in most of existing socio-technical settings to a knowledgeable human's rational and analytical skills (e.g. the minimal data set of equipment readings in an Emergency Room, i.e. (Summers, Jansen *et al.* 1997)).

Capitalising on the above mentioned case, Alice should interact with a flexible norm-based system to handle situations upon which she can trigger the required task and force the system to configure itself. This can be achieved either after a successful consultation of the predetermined context (in a condition-action mode) or by executing the reconfiguration combined with organisationally-oriented sanctions that aim at ensuring the overall good behaviour of the system. For example, in a norm-based system, Alice as a library clerk, would be allowed to let a library member borrow a reference due to some "urgent" need, despite the fact that only supervisors are entitled to do so. However, the norm-based mechanism would absorb such a violating interaction and send a message to her supervisor to ensure that the ideal state is preserved in a reasonable time interval.

This kind of flexible interaction-enabling mechanism can be perceived as a state-based delegation mechanism that has more rich semantics than the statically enforced, reactively triggered and causally managed counterparts found in (Sloman & Lupu 2000) and (Moffet 1998) approaches. The concepts presented in these approaches, many of which are in line with the advocated approach, have been proved by supporting tools and industry-oriented case studies. Yet, they are incapable of handling biddable interactions, managing violating states and bringing new state of affairs (i.e. system response through reconfigurations).

A major difference between the proposed approach and the one advocated by Sloman et al. is that the former provides a methodological approach towards developing role-based architectures that promote compositional design primitives, which superpose certain behaviours towards the biddable interactions of the populated social components (i.e. an adaptive role-based model of the system's enacting participants) at runtime. This, in turn provides a more practical organisational context where reconfiguration techniques support the evolution of system configuration as a means of responding to social interactions and contextual changes that lead to sub-ideal situations. Conversely, Ponder, as a policy specification language example of Sloman's approach, focuses on maintaining a distributed execution environment for utilising technical elements by enforcing post-deployment object-based access rules on subject-based obligations and interdictions. Policies in this approach refer to organisational high level goals and specify authorisations and obligations that realise long-term requirements and handle general cases. Subjects of policies can be either automated-agents or actor-like representation of the system's *logged-in* users whose behaviour is fully defined by the refrain and obligation policies that are imposed on them.

In other words these counterpart policy-based approaches are not capable of introducing organisational patterns to put legal enablement in effect within institutional settings as prescribed in law and deontic logic literature. The logic of such an enablement is based on reasoning about the enacting individual's capabilities and the norms of the institution. A detailed review of the research literature in deontic logic discipline has been discussed in Section 2.4.3 in the previous chapter.

4.7 Extending the 3Cs Framework

This section represents an architectural framework based on the ideas developed in previous sub-sections and extends the 3Cs framework that has been discussed thoroughly in the previous chapter. The hypothesisconcludes that the proposed framework's extension can be used to reason about and manage human interactions in organisational settings particularly in sub-ideal situation. This will mimic studies demonstrated in RBAC-base systems.

4.7.1 The Extended Conceptual Meta Model

The framework demonstrates relationships between the key conceptual components of the framework in which cardinalities are defined between the different elements. The extension of the 3Cs framework, introduces a framework that comprises two views: the role and the component views. The first includes the definition of organisational roles types, categories of processes' tasks—in terms of their corresponding configuration requirements—and finally context types that allow modelling normative contexts (for sub-ideal situations). The second view is typical to the 3Cs framework, which defines partners, coordination interfaces, contracts and coordination contexts. Figure 4.9 depicts a reduced abstract syntax that shows how the proposed role model maps to the 3Cs primitives.

Shaded rectangles label the extended concepts/primitives of the 3Cs framework and how they relate to the original 3Cs architectural primitives, which are demonstrated as unshaded rectangles. Herein, the interactions of human participants are modelled using collections of defined social roles that crosscut process-based capabilities. Thus, they are no longer introduced solely to the architectural configuration through pre-defined computer-mediated interfaces (i.e. actors/ partners) to which technological services and possible configurations are pertained. The newly introduced task primitive allows grouping a set of operations among which one is selected to label the task itself. These selected operations are then leveraged to a higher level of abstraction through the social role primitive that enables the inclusion of a set of tasks and their related labelling operations.

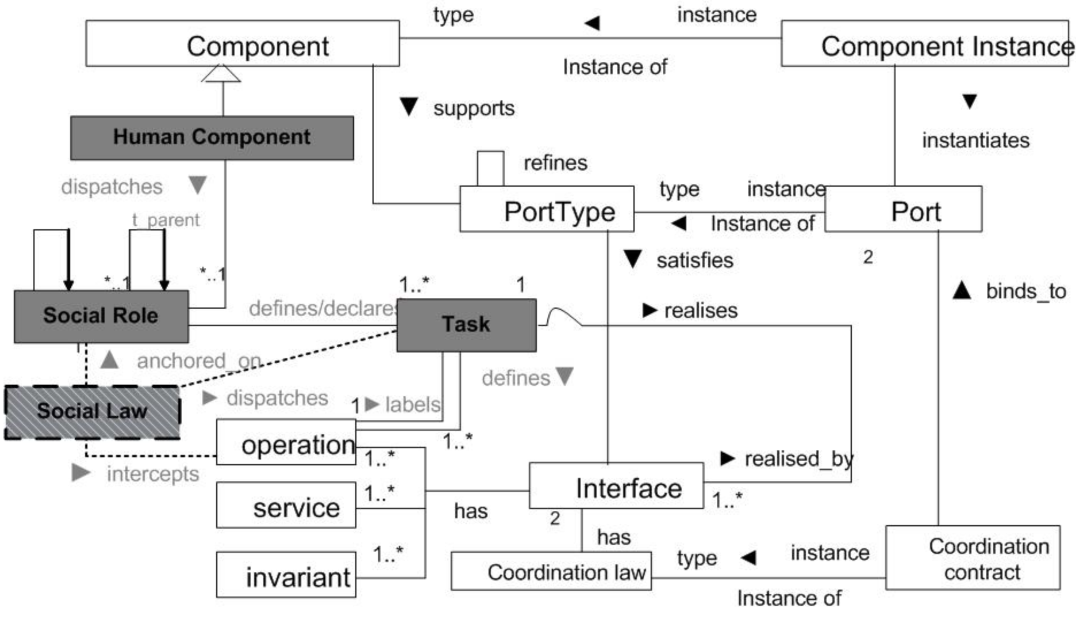


Figure 4.9 The abstract syntax of the extended 3Cs framework (conceptual elements)

Such operations are of a great importance, particularly if they refer to tasks that are part of the role capabilities (i.e. *declared tasks*) yet they are not part of the role's institutionally granted permissions (i.e. *defined tasks*). The taxonomy of these tasks was presented in Section 4.5.2 (see Table 4.1). The labelling operations of these tasks are filtered through the monitoring of the configured social roles and thus captured either upon the unexpected enactment of them or upon the unexpected ignorance of them. Especially, when they are required and purposefully configured for a human participant to alleviate some captured context deviation. The management of these operations might interfere with coordination-based management of social laws, which is limited to causal and reactive responses to user-defined triggers.

As shown in Figure 4.9., the abstract syntax demonstrates the notational language, which contains the concepts that populate valid configurations. Despite the fact that these newly introduced notational elements are part of the language syntax they have also structural semantics. These elements can be instantiated at runtime and thus can be reasoned about and evolved through transformations. For example social roles and tasks and their interconnections, among the rest of original 3Cs concepts, demonstrate the structural semantics of the proposed extension. Conversely, social laws cannot be instantiated, yet they embody the operational semantics, which is given precisely by their imposed reconfiguration operations. That, in turn, brings about a new state of affairs to respond to unexpected behaviours or sub-ideal situations (e.g. dispatching a required task). However, the social law concept and its interconnections are presented in the abstract syntax in a shallow way for the sake of clarity[7] (i.e. presented as rectangles with different shading pattern).

4.7.2 The Framework's Layers

In this framework, there are two levels: the normative (i.e. collaborative) level and the coordination level that refers to the aforementioned 3Cs primitives. The focus herein is on the normative level, which includes definitions of the proposed architectural primitives: social role types, social laws, participant's interactions and contextual signs. The second level is the coordination-based one, which includes definitions of architectural interfaces pertaining to technological components, the stiff and unresponsive representation of *users* who interact with them, and the coordination rules that causally coordinate their interactions.

By following this levelling approach, it is possible to provide better evolution support for those parts that have a higher changeability rate, or to provide different evolution techniques for different views on the software. This approach is inline with what (Mens & Wermelinger 2001) have stated: "separation of concerns allows us to separate parts of the software that exhibit different rates of change or different types of change". Another common way to achieve separation of concerns is by raising the level of abstraction to the level of software architectures, business rules and Meta-models. This makes software evolution more controllable. These motivations for evolution modelling provide the support for the argument to separate the social view from the technical one with regards to evolution as norm-based evolutions have different rate, nature and cause of change comparing to the coordination-based ones.

[7] Social laws, unlike other concepts in the meta model, cannot be instantiated at runtime, and thus they are not *genuine* parts of configuration states.

4.7.3 The Framework's Reconfiguration Mechanisms

A configuration manager is made so as to provide dynamic reconfiguration features in architectural models developed in the 3Cs framework. Dynamic reconfiguration involves the capture of the quiescent state of the target configuration, the addition, removal and binding of relevant components as well as connectors to transfer the state from the existing configuration to a new one.

An original view-based solution is proposed: (i) using a new role view for describing and reasoning about the behaviour of human participants as a set of roles, task, permissions and obligations; (ii) providing an atomic event-based execution model for the other technological elements (as components). In this thesis, a generic modelling approach has been achieved to cater for modelling and reasoning about interactions of human participants by means of behavioural description extensions to the 3Cs configuration language and concepts. Figure 4.8 shows the space for different sorts of monitored interactions with regards to a participant who plays a certain role among his roles space and instantiates, in the mean time, coordinated interaction (through HCI).

4.8 Related Work

Central to the view of this research is the dominant role of software architecture in planning, coordinating, monitoring, evaluating and implementing purposeful self-adaptation (Oreizy, Gorlick *et al.* 1999). Research on the self-adaptation paradigm has gained focus among other studies on self-* paradigms such as self-managed, self-organised and self-healing systems. Each of these paradigms has its own definition that characterises the functionalities that it focuses on. Self-adaptation is a class of adaptivity that characterises systems that are endowed with programs that *monitor* and *evaluate* conditional expressions to alter the behaviour of the system based on the outcome (Karsai & Sztipanovits 1999, Oreizy, Gorlick *et al.* 1999, Cheng, de Lemos *et al.* 2008b). Self-adaptive systems are able to evolve system behaviour after *design-time* and adapt themselves at *runtime* to unanticipated changes in their operating environment, which includes monitored social interactions and/or the system internal state, without explicit manual interventions (Dowling & Cahill 2001)

The notion of roles that the approach at hand adopts is inline with (Ould 1995) in the sense of grouping responsibilities that can be conferred to people or machines in the sense of (Tan & Then 1998) and (Cebulla 2004). The ROAD[8] framework (Colman & Han 2007) was presented within this new stream and demonstrated an approach, which exploits organisational theory and promote roles as first class entities to construct adaptive architectural frameworks. For the sake managing associations (i.e. coordination) between roles and binding between roles and players ROAD exhibits four-layer architecture: *computational-objects*, *functional-roles*, *management-contracts* and *organisation* layers. The first layer can be mapped to the 3Cs computational layer, which includes deployed components and their interfaces, while the second and the third layers (i.e., functional-roles and management contracts layers) play the same role as the coordination layer in the 3Cs framework. Functional-roles and management control both binding and superimposion mechanisms (i.e. indirection of instantiation and indirection of composition in ROAD's terms) through coordination interfaces and coordination contracts, respectively. The fourth layer, which does not have a counterpart neither in the 3Cs framework nor in any of its proponent architectural frameworks. Yet, it targets the organisational abstractions of adaptive systems and tackles the effects of the autonomy of participating human players, as presented in (Colman & Han 2005).

ROAD and the proposed framework have foundational differences and some shared traits. From a bird's eye view, both ROAD and 3Cs frameworks superpose contracts to alter the behaviour of the target system. However, ROAD aims at controlling performance variability through self-management whereas the 3Cs focuses on enforcing business rules and purposeful behaviours through self-adaptation. Additionally, each framework has its own communication backbone. ROAD components exchange explicit regulatory control messages through a coordination network while the 3Cs framework relies on intercepting interactions in a globally-shared event-based registry between systems components.

Despite the fact that both frameworks put forward roles as first-class runtime citizen with which organisational players engage, each of these frameworks takes an opposite approach in defining the separation between roles and players. ROAD adopt a full separation between roles and player for the sake of supporting the application developer in creating distributed environment of interacting entities in an organisational form in a way that is agnostic from the nature of role players (i.e. components, agents or human operators). Conversely, the proposed approach has targeted the component-based paradigm and thus drew a line from the beginning between the representation of technical and social

[8] An abbreviation that stands for Role Oriented Adaptive Design

components. Therefore, the proposed method addresses the role and player relationship from the point of view of social components, which are capable of violating obligations and permissions whereas the 3Cs traditional primitives tackle both regulation and the adaptation of purely technical configurations. The key aspect of the separation between roles and role players is the differentiation between permissions and capabilities as mentioned earlier in this chapter. Functional roles in ROAD and social roles in the proposed framework are highly abstract constructs that do not specify how the underlined system will be implemented. Moreover, they allow having many instances of the same role type at the same time (i.e. same configuration), and more importantly support representing organisational positions, which can be temporarily empty and can be played by several players at different times. The exploitation of social roles has two-fold purposes: regulating interactions of social components and provide the required knowledge for reasoning about purposeful self-adaptation in the face of sub-ideal circumstances and the biddable interactions of participants. Functional roles in ROAD are expressed in terms of purpose, function, performance requirement, interaction protocols and authority relationships.

The counterpart of the biddability concept and its management mechanism is found in ROAD in the form of the *Organiser Role*, which demonstrates high level autonomy in performing reconfiguration that is suitable to achieve the self-management of compositions (i.e. configurations). Finally one of distinctive features of ROAD, which is considered remarkable, is that it allows recursive composition and decompositions of composites at runtime.

Several research studies have contributed to multi-dimensional role modelling approach that is proposed in this chapter. From the methodological point of view, contributions of (Crook, Ince & Nuseibeh 2002, Crook, Ince *et al.* 2003, Crook, Ince *et al.* 2005) have focused on an analytical framework to support the requirements engineering view to cater for role-based security goals without providing any clue concerning the architecture/implementation level. A similar work that they refer to as a counterpart is the work of Fontaine (Fontaine 2001) who integrated Ponder language (Damianou, Dulay *et al.* 2001) with KAOS (van Lamsweerde, Darimont & Letier 1998) as an approach that leverages Ponder to a higher level of abstraction. Ponder language is a policy language designed to provide support for the deployment of security policies within distributed systems environments, and hence, is meant to be interpreted and executed by a deployment framework. Thus, Ponder lacks the required level of abstraction to capture the changing system's requirements and stakeholders' goals. On the other hand, KAOS provides the means to assign goals to agents that represent the system's stake holders.

The most fundamental difference between the illustrated approach and Fontaine's work is that he aims at modelling the assignment of system constituents (agents) to goals. Instead, this thesis advocates an architectural framework that contains a single agent (i.e., the configuration manager) which interprets biddable interactions by means of social laws, confers obligations and has the power to enable or suppress participant's interactions. OASIS model (Yao, Moody *et al.* 2001) is an important example that incorporates RBAC mechanisms and bundles them with Role-Based Access Control using parameters, active policy management, meta-policies and verification.

4.9 Discussion

The implications of the role theory, with regards to the presented view of modelling socio-technical system have contributed to the proposed role-based monitoring and reconfiguration mechanisms. As denoted earlier, the primary difference between this work and earlier research in this area is the decoupling of the management of human interactions from the traditional causal approach that normally addresses monitoring and reconfigurations mechanism at infrastructure or application level, i.e. hardware and software components. Alternatively, the approach promoted a set of new architectural primitives that capture the biddable nature of human interactions, a feature that might be vital in certain situations, to reason about such kind of interactions and allow appropriate reconfiguration responses that ensure the well-being of the socio-technical system under focus. A key issue here is how to map the allowed space of interactions that correspond to human participants to available— or subjected to scheduling—technical components, which have to be configured correctly to allow the required behaviour of such interactions.

In order to reason about the allowed space of social interactions, social laws are anchored on (social roles) to embody structural roles relationships and tasks to envisage key functional operations of social components.

The complimenting fit between the technical requirements of an enacted or compelled task upon handling a sub-ideal context, in the one hand, and the capabilities and the permissions of the enacting or compelled to enact social component, on the other hand, are the enablers of the success of the reconfiguration to alleviate such contexts.

Stripped from the context of a particular domain application, the proposed norm-based reconfiguration mechanism bounds organisational structures, permissions and capabilities to the participating social components in a flexible way to handle sub-ideal situations and support convergence to a more stable state. It is worthy to note, as a conclusion, that when biddable social participants internalise (or engage) in a social law it does not simply expand participants' set of permissions by exercising norm-based reconfigurations, but rather by allowing social participants to act differently, to handle sub-ideal situations, yet in a way that follows organisational norms and exploits the capability-based role model.

Chapter 5

The Methodological Approach

"How design decision shape the emergence of socio-technical system infra-structures and its accompanying work practice, is fundamental to the technology and ways in which human agency fits within its borders."
(Scott & Wanger 2003)

5.1 Overview

The increasing role of information technology in complex systems necessitates a substantial change in the engineering approach to these systems. This was particularly characterised by a shift from the conventional software architectures towards service oriented Architectures (SOA). This wave of service-orientation when applied to socio-technical systems modelling provides the basis for the composition of the norm-based architectural primitives that were presented in the previous chapter, to achieve a system method that offers a normative perspective to socio-technical models.

5.1.1 Objectives

The goal of this chapter is to present the engineering principles for constructing an architectural method that underpins collaboration dependencies including the interactions of the participating human components within the boundaries of software systems modelling. This method highlights concepts and abstractions that contribute to a "dual-view" model: this balances coordinated technical models with the biddable nature of social entities, in order to incorporate human interactions within the systems (i.e. socio-technical systems) in which these components participate.

The ultimate goal is not to propose a top-down method of engineering collaborative (sub) systems but to promote a method that utilises architectural primitives, techniques and styles to develop socio-technical applications. This goal is achieved by constructing a flexible process-aware view of the collaborations between social interactions and technological components within socio-technical systems, and making these collaborations adaptable and "fit for purpose”.

5.1.2 The Chapter’s Structure

This chapter is organised as follows: before the commencement of the next section, the basis of the proposed process-oriented view of social interactions is provided. Section 5.2 illustrates the relationships between domains, models and properties related to collaborations that include human interactions. Section 5.3 presents how Problem Frames would capture concepts and abstraction related to the adaptation of organisational models in response to changes incurred by biddable behaviours and context ideality. Section 5.4 discusses the perspective for a methodology that can be built to support the proposed method whereas the related research work is reviewed in Section 5.5. The outcomes of his chapter are summarised in Section 5.6.

5.1.3 Process-aware Interactions

Smith and Finger define a business process as: “the complete and dynamically coordinated set of collaborative and transactional activities that deliver value to customers” (Smith & Finger 2003). Ould stresses that in a socio-technical setting not only information is needed to perform processes but also technical software and hardware components (Ould 1995). He leads the shift from the information-centric to interaction-centric approach to process modelling: “[...] in our approach to process modelling a modeller concentrates unashamedly on what people do, rather than on what people do it with”. This view is inline with (Smith & Finger 2003) with respect to their observation of the combinatory nature of business processes which consist of both transactional and collaborative views. The previous chapter showed that the 3Cs approach fits well in modelling the transactional view and promoted social laws, roles and tasks to extend its capabilities towards the modelling of the collaborative view.

I have recognised the synergies between the proposed architectural approach and recent research finding on patterns of workflow-based systems, which have been segregated into three different streams: Workflow Control Patterns (Schnenberg, Mans *et al.* 2008), Workflow Data Patterns (Georgakopoulos, Hornick & Sheth 1995, Russel, ter Hofstede *et al.* 2005) and Workflow Resource Patterns (Russel, ter Hofstede *et al.* 2005). The first stream states that deviation of expected behaviour can be modelled and captured in terms of alternative paths of behaviour, whereas the last stream is more in line with my view, which focuses on responding to deviations according to the available resources.

Recent research on workflow has also shed light on the need for runtime changes – *momentary changes* in (van der Aalst & Jablonski 2000). This recent trend of keeping space for unexpected changes in the process at runtime tries to overlook the rigidity of design-time approaches to process change such as the Case-based handling of workflows (Berens 2005). A parallel work has also established a flexible framework for ad hoc changes in processes to support collaboration for virtual teams (Dustar 2004). Additionally, Lenz et al. (Lenz & Reichert 2007) stated that there is a price to be paid for isolating control flow from application logic. They put forward a workflow engine to accommodate ad hoc changes at different levels of abstractions. Although they distinguish between stable organisational processes and continuously changing clinical treatment processes, they are not able to model how knowledge about medical staff may affect such changes.

As the focus is on modelling biddable social interactions, it is necessary to come across certain types of processes (i.e. *human-driven processes*)—a term that was coined by (Harrison-Broninski 2005)—as an extension of Ould's view to refer to processes where the interest is to describe how people do things. This view is distinguished from mechanistic processes, which define and/or describe how tasks get done. Harrison-Broninski has emphasised that there is no common set of processes or technology can equally and efficiently address both types. Therefore, he introduced Human Interaction Management (HIM), which comprises as set of principles and patterns for structuring, supporting and controlling human work practices to deal not only with tasks, but also to provide a basis for innovation and creativity. In practice both processes interweave execution and implicitly communicate as shown in Figure 5.1.

Unlike the user-centric approach in which participants presented by actor models, their role in the system's well-being cannot be *replaced* by technical components (i.e. automation), therefore, the participants' goal-oriented tasks should be supported by means of empowerments, rewards or sanctions. This sort of support is realised through a set of well-defined reconfiguration operations that bring about a new state of affairs, which facilitates internalising the task or put an end to its progress.

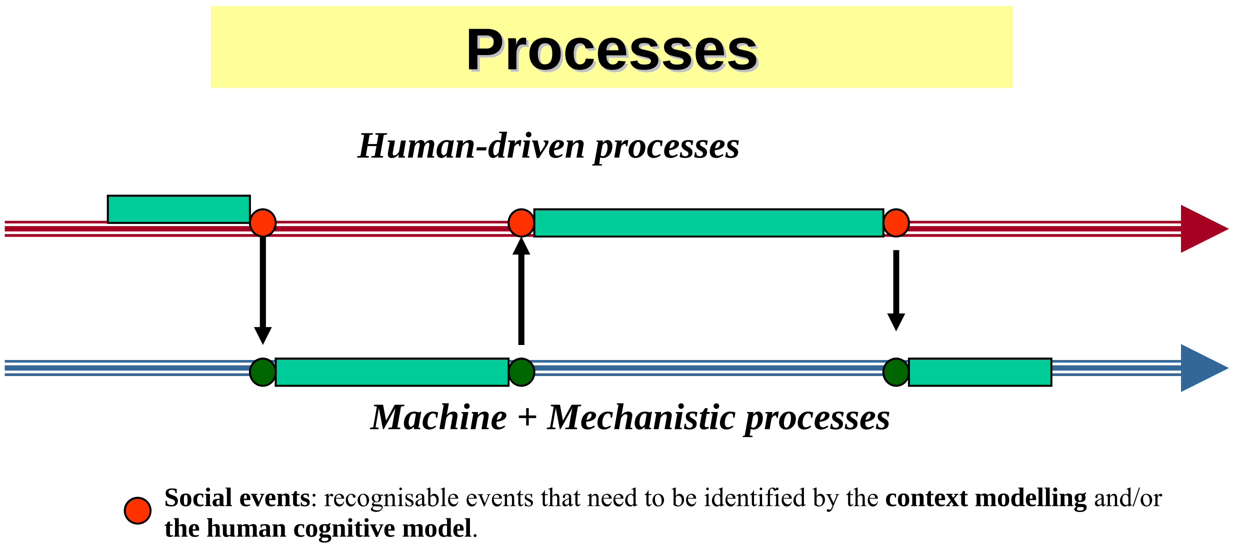

Processes
Human-driven processes
Machine + Mechanistic processes
Social events: recognisable events that need to be identified by the context modelling and/or the human cognitive model.

Figure 5.1 Processes Taxonomy of (Harrison-Broninski 2005)

Moreover, human participants deviate from plans (i.e. prescribed processes). These deviations may be unavoidable or even sometimes desirable from a social/cognitive perspective that corresponds to the system's higher level goals (Guindon, Kanser & Curtis 1987). However, addressing these deviations leads to a variety of difficulties at the process definition, the (sub)system configuration and the instantiations of the underlying components. The cognitive perspective includes elements such as the participant's perception of contextual indicators whether captured or not by the monitoring sub-system, reasoning based on these perceptions, memory and knowledge. The next section puts forward a step-wise guiding methodology that aims for the same goal yet departs from the cognitive perspective and brings to the fore organisational and technical configuration aspects.

5.2 A Method for Socio-technical Protocols

As mentioned in Chapter 4, the concepts of separation of concerns and separation of control have been utilised from software engineering and HIM respectively. Before delving into the details of the proposed methodological approach the following assumptions about the environment have to be emphasised:

1. Every human interaction with a complex machine that takes place through computer/software is considered a Human-Computer Interaction and thus human-machine interaction is modelled through a piece of software modelled in coordination interfaces (see Chapter 3 for details).
2. Every human interaction with a complex machine (i.e. tangible parts of software intensive systems) is monitored via the configuration manager.
3. Normative behaviours refer to entry operations that are enacted by humans or required by the system in sub-ideal contexts.

With regard to social interactions, the approach provides a design for a machine, namely an *architectural harmoniser,* to *combine* and *adapt* both interactions of participating humans and technological components towards the non-causal manner of social entities and changing environment context through norm-based reconfigurations to attain the required overall good behaviour of the system. As such this behaviour should support the convergence from a captured sub-ideal state to a more stable one. Figure 5.2 shows how an architectural harmoniser would interact with other elements within a socio-technical protocol.

The harmoniser can be perceived as a generic self-adaptivity manager, which extends the 3Cs configuration manager to support collaborative aspects between social components (i.e. role players) and technical components. An harmoniser is a software application that monitors the balance between unexpected interactions and current role entitlements. It interprets the operational semantics of triggered social laws (through the graph-based model) in order to effectuate the empowerment aspects of social interactions or the imposition of sanctions when appropriate

The harmoniser is a constituent part of contributions presented in this thesis as it is considered the heart of the proposed regulative approach, which consists of indicative and optative modes. From machine's perspective, the indicative mode specifies the behaviour of the controlled domains, regardless of the behaviour of the orchestrator machine (i.e. behaviours that stem from issuing commands to these machines that are subject to purely causal rules. Executing causal commands includes intercepting events, triggering superposed contracts and calling eligible corresponding services. For instance, an orchestrator may interpret and execute a rule to allow a lift to stop to serve a caller's request if it matches its direction and destination.

Conversely, the optative mode guides the behaviours that the orchestrator desires in order to maintain the stability of the system or to achieve a certain goal. This includes in addition to the above-mentioned steps putting into effect system norms that empower the lift user to direct the lift to a certain floor, despite the fact that it is not in its current trajectory, if an emergency is detected in this particular floor.

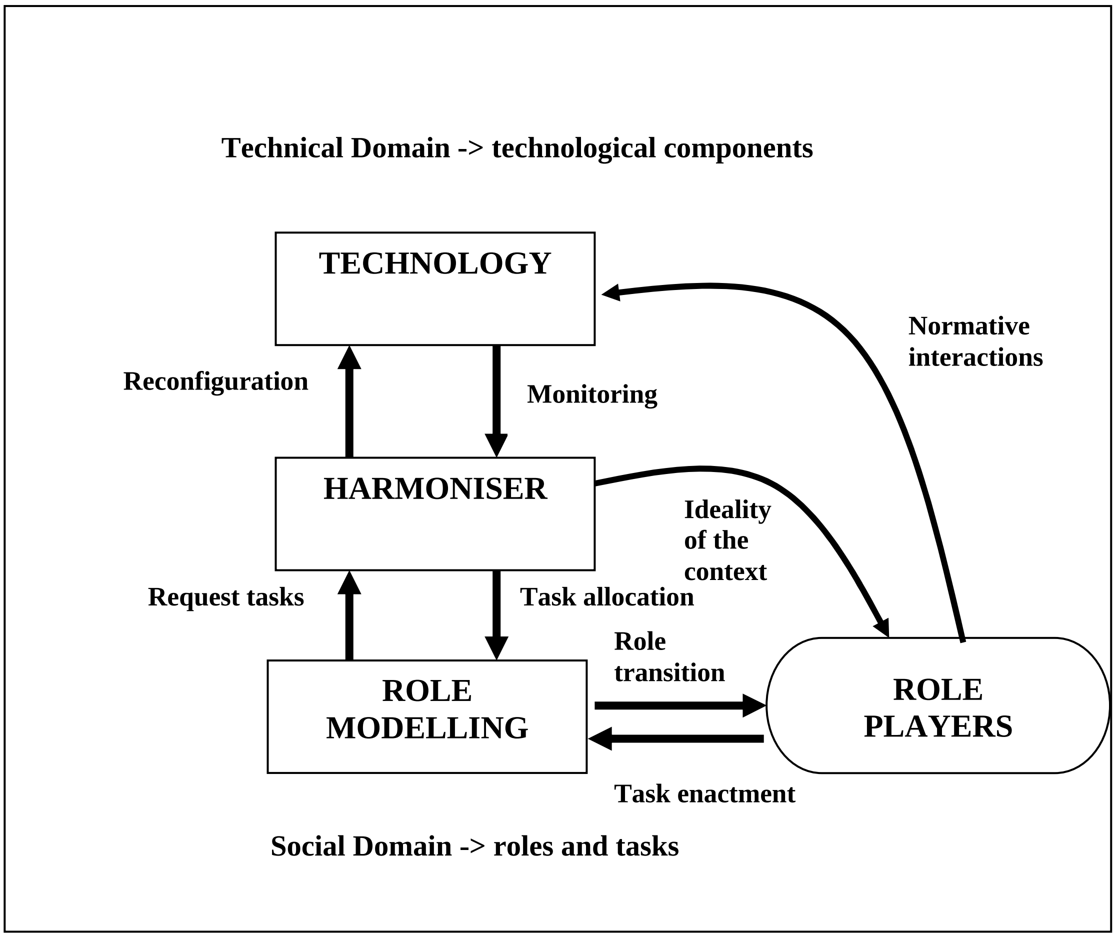


Figure 5.2 A harmoniser's interactions within a socio-technical protocol

The harmoniser machine takes a further step by providing dynamic requirements specification for collaboration and identifies a collaborative (sub)system that caters for the ideality degree of the context and the biddability of the enacting participants. Thus, the harmoniser presents a new view that considers the orchestrator's static requirements as indicative and supports regaining the system stability in response to the occurrence of a sub-ideal situation and/or an unexpected behaviour of the system's participant as an emerging optative behaviour. In other words the harmoniser executes a sort of reconfiguration that embodies human-in-the-loop. A harmoniser takes the following elements as inputs:

- A protocol, i.e. a set of social and technical components (which are identified through models of their behaviour) and the interactions that exist between them. These correspond to the roles of architectural connectors, coordination interfaces, as in the 3Cs approach (Andrade & Fiadeiro 2003), as well as the newly introduced social roles.
- A specification of the collaboration mechanism that applies to interactions, which corresponds to the identified sub-ideal context or violation and thus defines the semantic of required social laws.
- A purpose, which is an expression that represents the required behaviour that is expected to emerge from the interactions within the *deviating* protocol and exhibit "fitness of purpose"—what in a problem frame corresponds to the requirement specification.

As an extension of this work, it would be beneficial for architects to develop a hierarchy of sub-ideal situations in which the protocol may become involved together with a way of evaluating the distances that separate them from ideal states. Herein, the levels of sub-ideality that are captured within the trigger types of social laws as presented earlier in Section 4.5.4.1:

- o Capturing an interaction enactment that is not permitted
  - ▪ Managed positively through role-based reconfiguration
- o Capturing an interaction that is permitted but not enabled
  - ▪ Managed positively through coordination-based reconfiguration
- o Capturing the ignorance of an enabled (obliged) interaction
  - ▪ Managed through imposing sanctions

When the harmoniser acts in a proactive manner, the two triggers demonstrate the reactive response of the harmoniser to an unexpected social interaction by means of executing reconfiguration operations that bring about new state of affair in terms of removing coordination or role obstacles. The third trigger handles the case of detecting the ignorance violation of social participants to an obligation that has been put to effect through an architectural reconfiguration. The ignorance violation is assumed by the detection of the absence of the task enactment yet the proposed modelling language abstracts away the temporal aspects of the ignorance detection.

5.2.1 Extending the Methodological Principles of the 3Cs Approach

The coordination interfaces of the 3Cs approach constitute the cornerstone for achieving separation of concerns and business-oriented reuse. They were externalised from coordination laws for the sake of representing abstract business entities that preserve *compliance* relationship with the components that instantiate them (Andrade & Fiadeiro 2003). Coordination interfaces can be organised in hierarchies that exhibit the inheritance of the component-compliance relationships down the hierarchy.

The key methodological principle of the coordination interface that the approach capitalises on is the fact that the compliance relationships are driven by business rules and each of which exposes the business view of a certain usage of the related components rather than reflecting the essence of the functionalities pertained to these components in the business domain. Thus, it would be more efficacious to have as many *fit-to-purpose* interfaces as required for modelling business rules related to a respiratory machine, for instance, instead of having a *general-purpose* interface for this particular machine.

The argument behind this design choice is that it provides some room to manoeuvre when the system specifier needs to change the usage requirements, which are placed by the laws not by the entities, and thus, new business rules can be applied. Additionally, the binding mechanism that allows connectors (i.e. coordination laws) to be superposed over components instantiating their coordination interfaces, are independent of the target development environment. The degree of dynamic reconfigurability that can be achieved through coordination interfaces depends on the ability of the execution environment to recognise triggers and interactions between entities, and to represent *operations* and *events* (Andrade, Fiadeiro *et al.* 2001, Andrade, Fiadeiro *et al.* 2002, Andrade, Gouveia *et al.* 2002).

A coordination interface may realise one or more component interface, thereby adding more constraints through the declared set of services and operations as well as through pre- and post- operation conditions. Once a component is instantiated and bound to a coordination law instance through a coordination interface, only operations and services defined within this interface are recognisable. This state can only be changed after executing a programmed or ad hoc reconfiguration. Moreover, there are no means by which the operations of a component can be reasoned about in the case of non-technical component. Figure 5.3 is a view of layered constraints' in the 3Cs configurations.

Technical components, particularly machines when they are considered constituents of a configuration, could be approached by biddable entities in a way that is beyond the definitions given by the instantiated component and/or coordination interfaces that correspond to a business component instantiated by such biddable entities. For example a doctor switching on the emergency mode of a respiratory machine, tuning it regardless of the normally imposed rules (coordination contract). Such an event has to be captured despite of the fact that it is not shown in an ordinary coordination interface and thus is not managed by the corresponding law. This case among others has necessitated the matching of tasks and their labels (i.e. entry operations), as a means for relating the biddability of social entities towards technical components, with their actual physical interface.

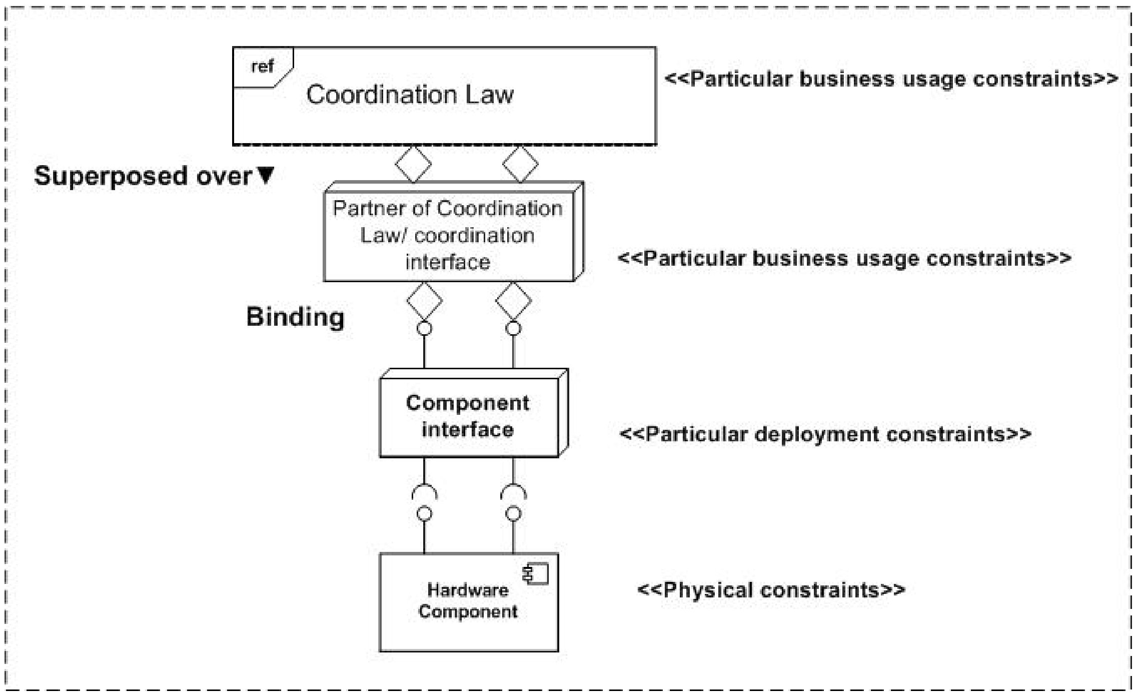

ref
Coordination Law
<<Particular business usage constraints>>
Superposed over▼
Partner of Coordination Law/ coordination interface
<<Particular business usage constraints>>
Binding
Component interface
<<Particular deployment constraints>>
Hardware Component
<<Physical constraints>>

Figure 5.3 Constraints spectrum in the 3Cs Approach

Having the entry operations within the task definitions does not alter the border between a technical component (i.e. software or hardware) and its environment, but at the same time it does realise a way for allowing human components to acquire permission for the particular task labelled by that entry operation at runtime. Once an entry operation is triggered at the component level, captured and recognised at the role level, and processed by means of sanctions or facilitation, reconfiguration can take place to instantiate proper component interfaces and/or laws. As described earlier, a role is a set of related tasks and thus a role instance model that corresponds to human component presents a mapping of endorsed tasks, capabilities and permissions ascribed to the participant to whom this component refers. The following section gives an illustrative example that explains the argument behind defining social roles, tasks and entry operations and how they are represented in a configuration.

5.2.1.1 An Example

As an extension of the respiratory machine example that has been proposed in Section 3.3.2 to introduce the 3Cs coordination primitives and expanded in Section 3.3.3.1 to illustrate 3Cs reconfiguration primitives, I presented how the newly introduced primitives (i.e. social role, social tasks and entry/exit operations) can support non-causal management of social interactions. In Section 3.3.4, I showed that if an interaction is required to adjust the respiratory machine to save a patient's life (i.e. tuning it out of the restricted scope) while the appropriate actor is not present then the tuning interaction(s) will not be performed. In such a case, normally any qualified doctor would be eligible to perform the operation even if he is not a permanent staff in this ward. Specifying such a fact in the coordination-context is not viable, as it will incur a scalability problem due to the need to cascade every constrained ad hoc reconfiguration operation (i.e. constrained by the truth of *d.work_in(w)*) with another one that consults the variable indicating the sub-ideal state). For example the pre-condition section of following reconfiguration operation should be supported with a reconfiguration operation that incorporates the vital signs context in every occurrence of d.work_in() to relax the condition of being a ward's staff.

```
subscribe_OPEN(d:doctor,r:respiratory):
     pre:  exists r and d.work_in(w)and r.fixed_in(w)
     post:   not exists' restricted-respiratory(d,r)and
           exists' open-respiratory(d,r)
```

The reader should consult Section 3.3.31 for the complete specification of the coordination context example. .Additionally, thing will get more complicated if the system specifier explicitly delegate the interaction to different categories of users (e.g. nursing staff) in such situations. Last but not the least, facilitating the appropriate configuration is not enough to yield the purposeful human-driven interaction anyway, as neither programs nor devices can ensure the required social interactions. Thus, in such cases organisational sanctions should be imposed to compel participants or at least to gain a better sub-ideal state (i.e. freezing their accounts and/or reporting the case to managers to whom they are accountable).

Suppose that the interaction could be conditionally delegated to nursing-staff, doctors and ward-doctors (i.e. doctors-in-charge) then each of these categories should contain a tuning-respiratory task definition in their corresponding role types. Only ward-doctors will have full permission to the task while the others should have partial access; however, it should be sufficient to handle sub-ideal situations (e.g. absence of the ward-doctor and the critical vital signs of the patient).

Figures 5.5 and 5.6 present how defined tasks and entry operations allow for the modelling capabilities of a role player and how they facilitate reconfigurations at both technical and role levels. In this example I show how a nurse role is modelled in a way that allow empowering her, if necessary, to internalise a respiratory machine *freely*. Figure 5.4 shows the traditional 3Cs user-centric approach posing a nurse's *actor* interconnected with a configured respiratory machine via the coordination contract restricted-respiratory yet it is eligible to other unknown configurations through coordination context. Herein, I assume that the respiratory-open contract is not among them.

If designers adopt the proposed approach it would be possible to reason about situations in which a nurse should be exceptionally enjoined to internalise the respiratory machine freely; however, such cases have to be addressed at a different level of abstraction. The designer has to correlate the nurse (the least capable role) and doctors that are not members of the ward (i.e. alternative roles) to the optimal role (ward-doctor) through a role hierarchy as shown in Figure 5.5.

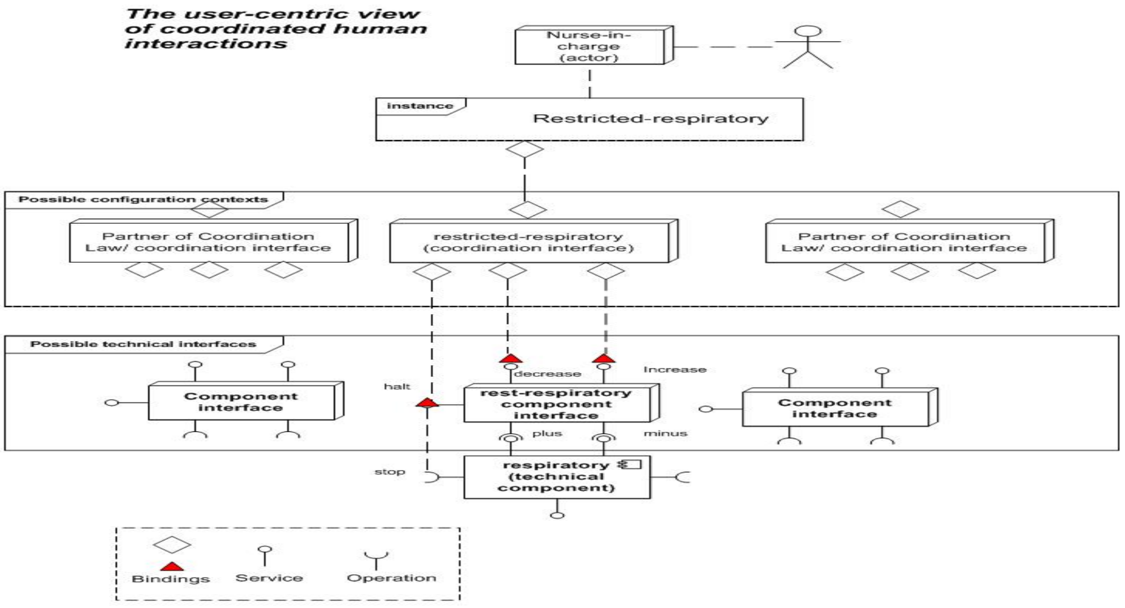
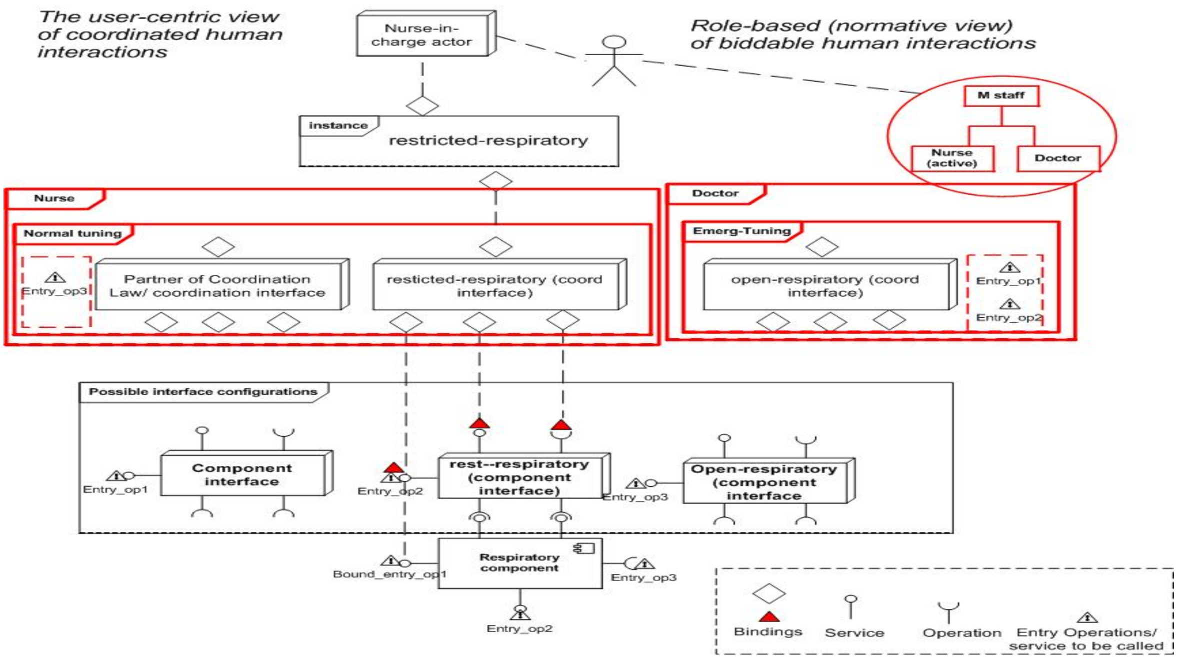

The user-centric view of coordinated human interactions
Nurse-in-charge (actor)
instance
Restricted-respiratory
Possible configuration contexts
Partner of Coordination Law/ coordination interface
restricted-respiratory (coordination interface)
Partner of Coordination Law/ coordination interface
Possible technical interfaces
Component interface
halt
decrease
Increase
rest-respiratory component interface
Component interface
plus
minus
stop
respiratory (technical component)
Bindings
Service
Operation

Figure 5.5 Modelling a task that is not part of the configured role

Therein, the nurse configuration includes a social interface that embodies a reference to respiratory-tuning task, namely to its entry/exit operations (i.e. the operation that changes the respiratory machine mode from normal to emergency so as to set its pressure out of the scope).

As shown in Figure 5.5 an entry operation of emergency respiratory tuning is embodied in the nurse social interface (i.e. which includes his possible tasks and related coordination interface. It is the responsibility of the social law as seen in the previous chapter to conclude whether the enacting player is eligible or obliged for an interaction with respect to the conceived social context. I did not name the entry/exit operations in the diagram yet it can be simply mapped to the operations found in the open and the restricted interfaces of the respiratory machine or to some other operations that can be mapped to the equipment interface (i.e. different buttons or mode-changing switch).


The user-centric view of coordinated human interactions
Nurse-in-charge actor
Role-based (normative view) of biddable human interactions
M staff
Nurse (active)
Doctor
instance
restricted-respiratory
Nurse
Normal tuning
Entry_op3
Partner of Coordination Law/ coordination interface
resticted-respiratory (coord interface)
Doctor
Emerg-Tuning
open-respiratory (coord interface)
Entry_op1
Entry_op2
Possible interface configurations
Component interface
Entry_op1
Entry_op2
rest--respiratory (component interface)
Entry_op3
Open-respiratory (component interface)
Respiratory component
Bound_entry_op1
Entry_op3
Entry_op2
Bindings
Service
Operation
Entry Operations/ service to be called

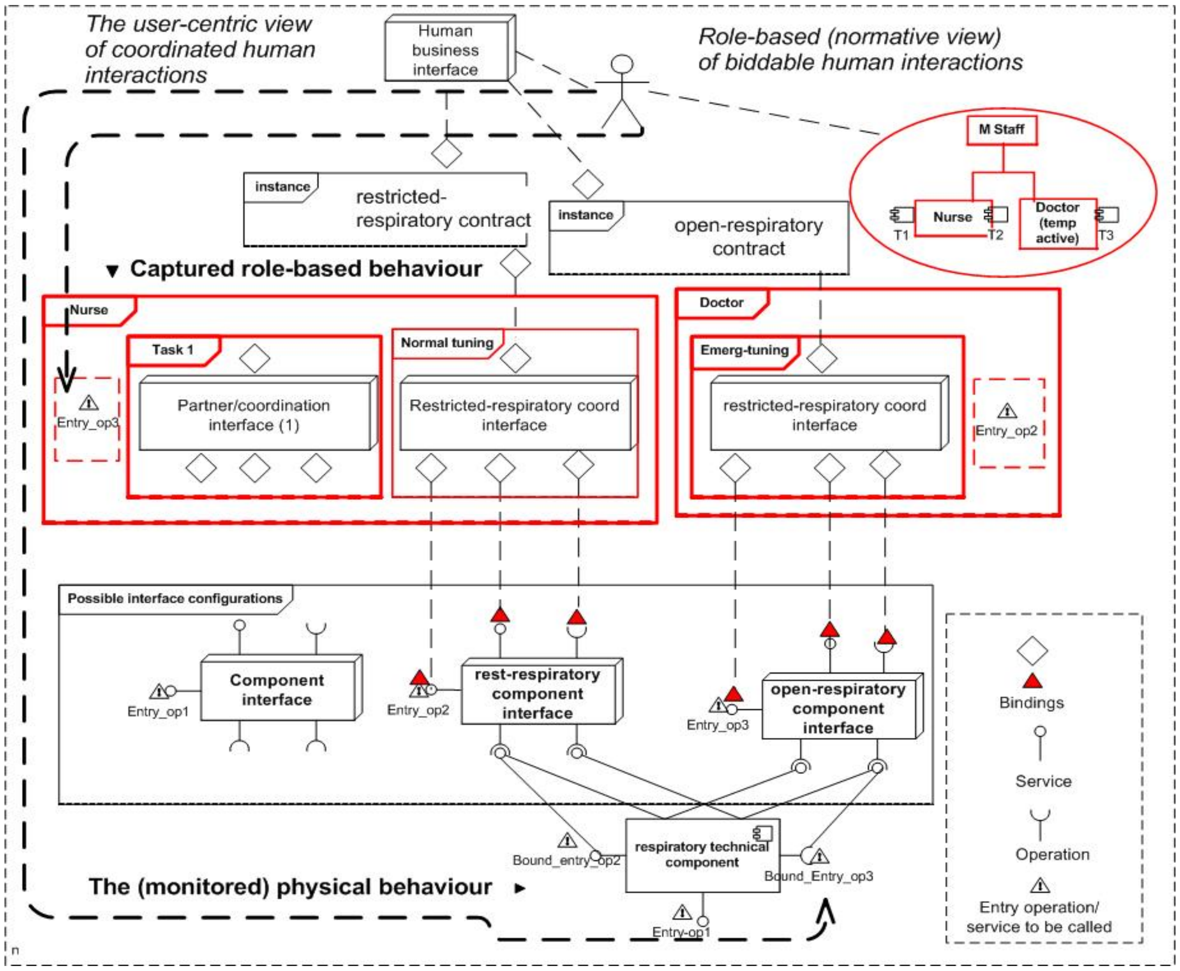

The user-centric view of coordinated human interactions
Human business interface
Role-based (normative view) of biddable human interactions
M Staff
Nurse
Doctor (temp active)
T1
T2
T3
instance
restricted-respiratory contract
instance
open-respiratory contract
Captured role-based behaviour
Nurse
Task 1
Partner/coordination interface (1)
Normal tuning
Restricted-respiratory coord interface
Entry_op3
Doctor
Emerg-tuning
restricted-respiratory coord interface
Entry_op2
Possible interface configurations
Component interface
Entry_op1
rest-respiratory component interface
Entry_op2
open-respiratory component interface
Entry_op3
respiratory technical component
Bound_entry_op2
Bound_Entry_op3
Entry-op1
The (monitored) physical behaviour
Bindings
Service
Operation
Entry operation/ service to be called

Figure 5.6 Enacting and permitting technical/role-based reconfigurations in two steps

5.3 Capturing Biddability: From Concepts to Models

The key reason for my interest in Problem Frames is not to support software requirements specification and design but rather to shape the methods and techniques that can be borrowed for supporting:

- The engineering of socio-technical systems: this may involve software applications as components, but it primarily involves complex interactions between social and technical components that are *indicative* in the sense of (Jackson 1995, Jackson 2001) as described in Chapter 2.
- The evolution of socio-technical systems: evolution should be supported with a highly abstract model and a reasoning approach to guide the response to changes in the system environment and the biddable interactions of its participants, particularly within organisational models, as specified in (Hall & Rapanotti 2005).

The author emphasises that analogies between Problem Frames and architectural connectors need to be taken with care. The synergies between software architecture and problem frames have been identified by (Hall, Jackson *et al.* 2002). Their research presents a slight extension to the Problem Frames notation aiming at expanding the definition of the Problem's machines to cater for architectural artefacts. This work was carried further by introducing the Coordinated Problem Frames approach, which correlates both Problem Frames and 3Cs approaches (Barroca, Fiadeiro *et al.* 2004), however, it can only be imposed on causal domains, not biddable ones.

This thesis provides a methodological approach that can model and manage the collaborations between social and technological components in a way that is adaptable and "fit for purpose." The inherent problem of social components, unlike technical ones, is that assumptions on their behaviour cannot be guaranteed by programs, as they do not control them. Thus, the aim is to compensate actual social components (i.e. biddable domains) with models to assist designers in inferring properties to validate possible configuration scenarios. The fitness-for-purpose view is borrowed from (Fiadeiro 2007) as depicted in Figure 5.7 where properties stem from the interconnections between the various system elements, which form purposive configurations and models, to support designers in inferring the evolution of underlying domains, which in turn represents the physical entities participating in the system (i.e. software components, devices and people).

The relation: fit (f_{ix} to X, where X represents the model and x a domain), shows how the three software-intensive system levels might relate. The line between the properties and models levels represents both logical inference and simulation, which are the two ways that designers can use to obtain system properties. Models (uppercase letters) and domains (lowercase letters) relate through fit (dashed arrows).

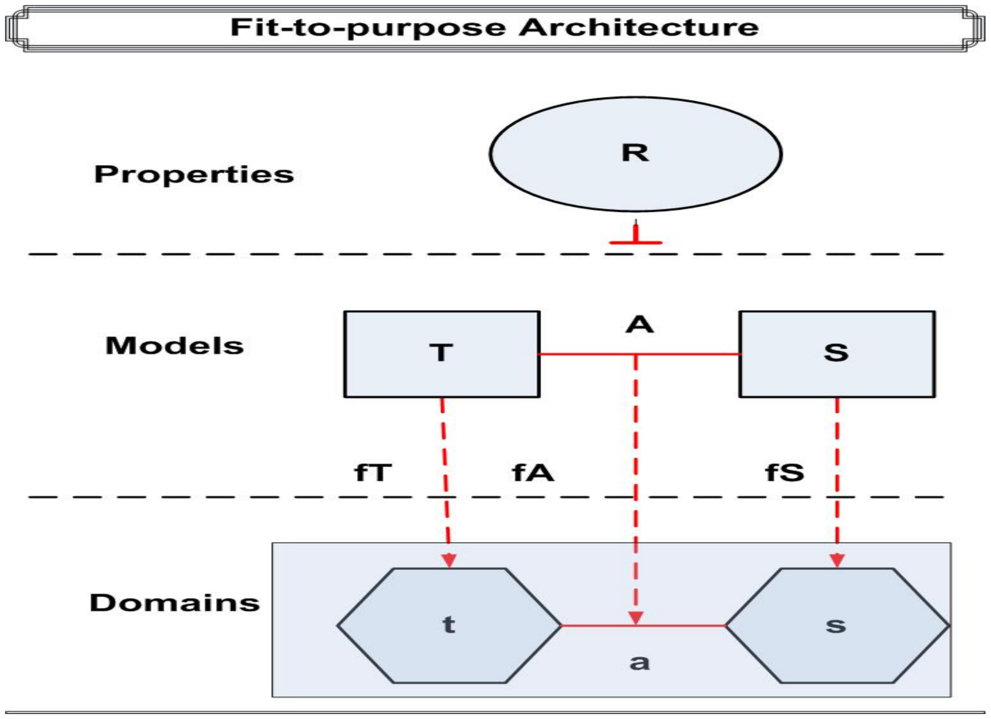


Figure 5.7 The fit-to-purpose Architecture (borrowed from (Fiadeiro 2007))

As proposed in (Fiadeiro 2007), there are three types of fits according to the corresponding domain type (Table 5.1).

Domain	Fit
Software domains	Expressed in the way programs implement the specifications (i.e. correctness)
Control/embedded systems domains	In control and embedded systems, the fit, which is based on an abstract representation in a mathematical domain, must operate an abstraction from a model of the target plan to the mathematical domain over which the models are expressed.
Social/human domains	The fit cannot be formalised, thus, the model expresses the norms that social components are expected to observe and defines the basis on which the component interactions elicit required properties.

Table 5.1 Domains and fit relationships

5.3.1 Modelling the Proposed Architectural Primitives

Social laws, along with their related role spaces that are enacted and interplayed by human participants, are suitable to specify the big *H* (i.e. human component representative in terms of role space and permissions). The big *K* (i.e. the knowledge domain) realises the interpretation of system instructions by means of technical (re)configurations together with human interactions as mentioned in Chapter 2 (see Figure 5.8). The *H-K* specification should differentiate between permissions and qualifications (i.e. competencies) in the light of possible technical configurations.

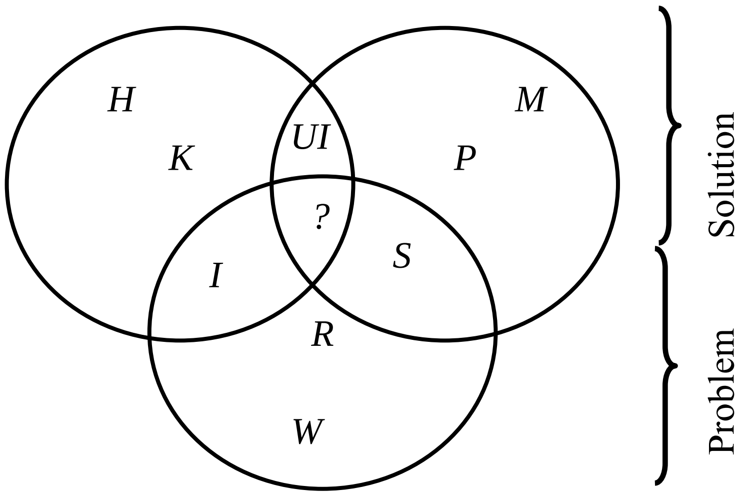


Figure 5.8 Problem Frames three-ellipse model of requirements

The knowledge domain has been introduced in the context of socio-technical requirements in (Brier, Rapanotti *et al.* 2004, Hall & Rapanotti 2005) to represent humans for which the instructions *I* have to be assigned. In a problem diagram, a knowledge domain is represented as a box with double bars on the right-hand side (unlike machine domains). Figure 5.9 illustrates a general form of socio-technical problem diagram; both machine and knowledge domains are subjects of design. I propose an extension to the work of (Brier, Rapanotti *et al.* 2006) by introducing the capability-based role concept and the technology-centric of view of tasks to enrich the knowledge domain.

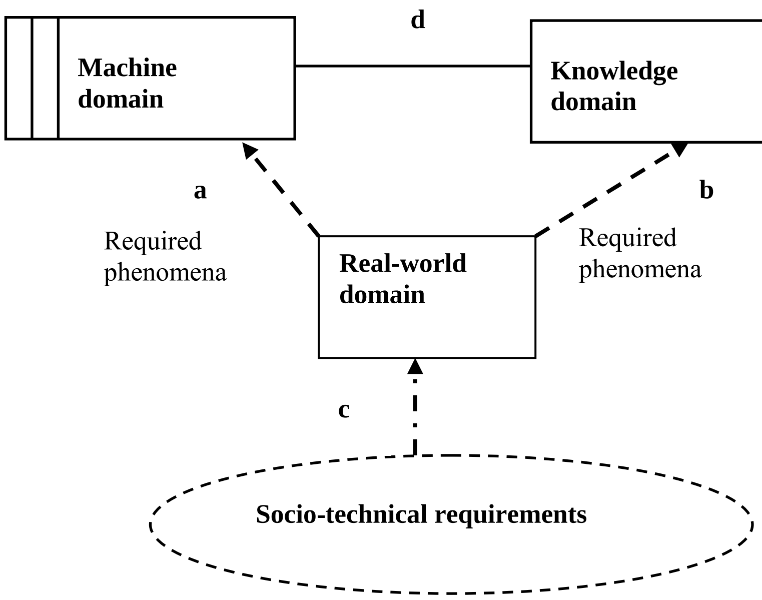


Figure 5.9: The general socio-technical problem diagram (Brier, Rapanotti *et al.* 2004)

Recalling the three-ellipse model, on the one hand, it is required to design a program *P*, which runs on the machine *M* to implement *S*, and on the other hand, to realise knowledge of how to execute the available services (i.e. guiding interactions of human *H*) to satisfy the set of expected instructions *I*:

- $P + M$ *satisfies* S (can be informally perceived e.g. $P_{SQL} + M_{DMBS}$ *satisfies* S (the targeted system specifications) and "+" denotes an informal runtime composition between the machine *M* and the program *P*
- $K_I + H_{role}$[9] *satisfies* I (a human with a sufficient role, knowledge and resources should satisfy the systems' instructions)

To present my view of architectural configurations, I promote two novel predicates namely *supported(X)* and *app(X, Y)* as basis of the intended extension of the notation of the organisational extension of Problem Frames presented in (Brier, Rapanotti *et al.* 2006). Both predicates are realised by the monitoring mechanism. The former returns true if the current (technical) configuration is realised with regard to a certain task X, whereas the latter examines the last interaction, which has been committed by the monitored human player X, against the required action Y.

However, to ensure the overall good behaviour of the system, a machine that can monitor and manipulate the K_I and H_{role} models particularly at sub-ideal situations, should be constructed. More precisely, $K_{I(task)}$ can be supported through adding or deleting

[9] H_{role} as a model of humans as role space (current/possible role configuration) instead of agents

technical components that suit that task, whereas roles can be enabled provided that they are part of the human player's capabilities:

1) supported($H_{I\text{-}task}$) + app(X, H_{role}) *satisfies I* (allow the interaction X)
2) supported($H_{I\text{-}task}$) + ¬app(X, H_{role}) *does not satisfy I* (impose sanctions on H_{role})
3) ¬supported($H_{I\text{-}task}$) + app(X, H_{role}) *satisfies I* (impose positive reconfigurations)

$H_{I\text{-}task}$ can be supported by providing the technical reconfiguration that is required to realise the intended task, i.e. providing software, hardware components and interconnection whose rules allow the required behaviours. A human role is considered appropriate if the role player is both capable and permitted to enact the task.

The proposed extension to Problem Frames targets finding a way to embed roles into the knowledge domain, and to use it as vehicle to reason about biddable interactions within organisational settings. It is akin to the extension presented in Figure 5.10 yet it is human-centric, taking into account the permissions and capabilities, on the one hand, and norm-based reconfigurations (i.e. facilitations and sanctions), on the other hand.

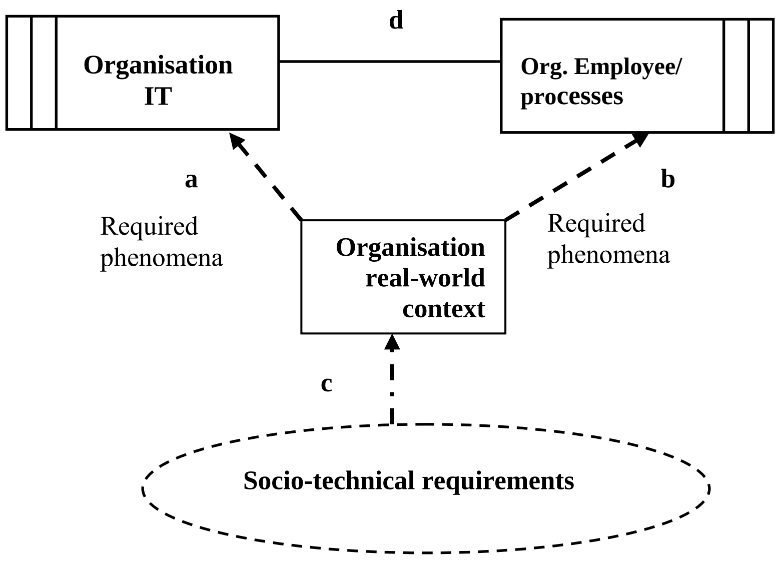


Figure 5.10 Extended Problem Frame notation or organisational problems (Brier, Rapanotti et al. 2004).

5.3.2 Separation of Concerns and Separation of Control

In Chapter 4 the concepts of separation of concerns and the separation of control were borrowed from software engineering and HIM paradigms respectively. With regard to social interactions, a design for a machine, namely an architectural harmoniser, was presented to *combine* and *adapt* both participating humans and technological components towards the non-causal manner of social entities and changing environment contexts through norm-based reconfigurations in order to attain the required overall good behaviour of the system. From this machine's perspective, the *indicative* mode specifies the behaviour that the controlled domains exhibit, according to causal coordination and reconfigurations (i.e., superposed contracts and coordination contexts), and regardless of the behaviour of the harmoniser machine. Conversely, the *optative* mode guides the behaviours that the harmoniser *desires* to bring about or maintain in order to keep the stability of the system. In order to generate this sort of desired behaviour, the proposed normative approach pays explicit attention to the norm-based self-adaptation rules enforcing or permitting interactions to human participants in order to achieve an overall desired behaviour of the system.

5.4 A Prospect of a Normative Methodology

This section discusses the prospect for a methodology that utilises the architectural primitives and the generic role-based technique that have been developed during the course of this research to support mapping and reasoning about biddable interactions within organisational settings. The proposed methodology is based on a new way of thinking, which injects the biddability of social interaction into early stages of development. It underlies an adequate formal conceptualization using the 3Cs business architecture (the technical view), and the newly introduced primitives. The original and the extended architectural approaches are constructed together in a stepwise way to capture and reason about social interactions within socio-technical processes (the normative view). In the proposed methodological approach, sub-ideal contexts in the view of role-player and optimal technical configuration inconsistencies can be taken as significant issues to identify perspective dependencies, in order to derive purposeful self-adaptation, and also, to allocate obligation and permission distributions over participants and/or social driven-processes.

The interplay between the instantiated primitives in the two views is realised based on the principle that the two architectural views are to be joined together to obtain the final architecture. After constructing this holistic view, biddable interactions, once initiated by social entities in the collaborative-mode, can change the role view and follow the consequences of the changes on the technical view, allowing modellers to take the right decision at design time. Moreover, designers can allocate requirements pertained to the normative view using a slightly-modified version of the Semiotics approach to requirements engineering which can be exploited to single out system norms (i.e., optative behaviours) from descriptive ones as mentioned in Chapter 2.

Semiotics is a relatively new paradigm that constitutes a candidate for the presented architectural approach to be promoted to a systematic methodology for the construction and the evolution of norm-based socio-technical systems. Stamper and Liu established the fundamental blocks of Semiotics: signs, information norms and systems (Stamper 1994, Liu 2000, Liu, Sun *et al.* 2001b). They advocate norms analysis as a system method that offers a normative perspective to system modelling and design, and utilises rich semantics to depict the ontological dependencies. Moreover, it facilitates the elicitation of system requirements (Stamper 1994). This method attempts to resolve several issues usually affecting systems with complex human interactions such as business exceptions, violations and normative positions. (Stamper 1994) provides an output norm template that can fit well with the constituents of social laws:

whenever <condition>
If <state>
then <role>
is <deontic operator>
to <action>

Organisational Semiotics presented by (Liu 2000, Gazendam, Jorma & Liu 2005) is a descendant of Semiotics that focuses on properties and behaviours of signs, that are exploited within organisational contexts and business-driven practice, as means for Human-Human and Human-Machine interactions. Despite the fact that Organisational Semiotics shares similar interest in modelling human interactions with the proposed framework (e.g. establishing commitments, permissions and obligation), it has a different scope as it focuses on explicit exchanged information, its structures and its meanings which constitute the basis of communication and negotiation between the system actors.

Moreover, Organisational Semiotics managed to model the collaborations between the system actors and facilitated organisational proxies for filtering social interactions yet it provides no means for generic and implementable applications that explicate the dependencies between social interactions, role structures and technical configurations in a predictable and assured manner to achieve desirable behaviours. Semiotics also lacks analysable models that show how the current state of affairs (i.e. in role-based or technical configuration terms) might change as a result of triggering norm-based adaptations (Kayser & Nouioua 2004) let alone executing the adaptation recipe for guiding the desired change despite of unexpected social interactions. However, this Semiotic-based methodology has proposed an elegant requirement elicitation method for deriving norms of systems buried in textual requirements documents.

A novel method, which extends the above-mentioned methodology, is put forward to support extracting organisational norms that correspond to normative positions. These normative positions influence biddable social interactions that are labelled as entry-operations of tasks required in sub-ideal situations. Norms can be extracted from system requirements as follows:

(1) Responsibility analysis: contextualised role/task relationships and human permission-agnostic capabilities.

(2) Partners identification (the coordination view)

(3) Triggers analysis

- *Pre*
  - A. Entry and exit operations of task
  - B. The conditions for activating and invoking norms
- *Post*
  - A. The resultant condition after successful norm execution
    - *Facilitating*
  - B. The resultant condition after unsuccessful norm execution
    - *Sanctions*
  - C. Actions required, suppressing unwanted behaviour

(4) Norm specification

5.4.1 General Steps of the Methodology

Defining the modelling primitives for specifying flexible self-adaptivity is a first step towards an architectural methodology for socio-technical systems that takes into account the biddable nature of social components through self-adaptation. Moreover, identifying potentials for sub-ideal situations is central to the proposed approach. Thus, I put forward methodological steps for identifying boundaries of contextual changes that contribute to sub-ideal situations, their recovery tasks and flexibility points. Flexibility points allow relaxing the requirements of these tasks, when required, to equate to existing operating conditions.

This sub-section draws the outlines of the adapted version of the HIM methodology that addresses the specific needs of the proposed architectural approach. HIM introduces process-based support for adaptive and collaborative social interactions, which may deviate from their prescribed plans, in a way that can be integrated with *routinised* processes, which are of a causal nature. In the proposed architectural approach, the biddability of social-interaction is tackled via self-adaptivity to achieve a system's high level goals, which make adopting the top-down design approach a natural choice. Thus, correlating process-aware tasks, human-driven processes and role-based interactions with the system adaptivity is a way to operationalise high level system goals such as preserving the stability of the system following the detection of a sub-ideal situation. Therefore, HIM is a justified starting point for the proposed methodology to support human participants to enact tasks that are within their capability-based roles space putting into consideration the changing organisational context.

The methodology in hand is a result of the combination of the HIM approach and the proposed extended 3Cs architectural method including its underlying primitives and patterns. However, further work is needed to evaluate its suitability for large–scale industrial projects. In this methodology, a couple of concepts are treated as first class citizens and governed by social laws: roles and tasks. In addition to these newly introduced concepts or primitives, 3Cs based primitives are still considered but as second class citizens (e.g. coordination interfaces and coordination laws).

Before describing the methodological steps, assumptions on targeted systems and the input information (e.g. requirements documents) should be clarified. These methodological steps are meant to target socio-technical systems that operate within monitored organisational contexts where roles of social participants are understood in terms of their capabilities to perform well-defined tasks. The proposed steps are based on the following assumptions:

- These steps target socio-technical systems that operate within monitored organisational contexts where roles of social participants are understood in terms of their capabilities to perform well-defined tasks.
- Tasks are normally parts of unstructured processes.

- Context-awareness is supported by system monitoring services that capture the behaviour of social and technical components.

Before delving into the details of the methodological step, the following documents should be prepared, as they constitute the inputs for commencing with these steps:

- Specifications of the system's processes that include tasks and operations. It should include also how people should go about them
- Organisational charts that embodies functional roles that corresponds to above-mentioned task and roles
- Code of norms that specify organisational obligations, permission and interdictions that prescribe the behaviours of organisational role players who should conform to these norms

The proposed steps aim at externalising system tasks that are human-driven and can be obliged or permitted by the system norms to alleviate certain situations (i.e. sub-ideal states). Figure 5.11 presents a simplified version of these steps.

These steps build an incremental model of purposeful tasks from the specifications of business process. A task from the social participant view is a collection of his interfaces to technical components within a purposeful configuration. A task is endowed with the "optimal" technical configuration and then related with certain goal. Then, modellers should query system norms to find when these goals become priorities (i.e. obliged to alleviate some sub-ideal state). In this context, the modeller should identify contextual information to be monitored, understand the required knowledge and skills of human participants in order to find flexible points to relax the task's operation conditions and role entitlements (i.e. the permissible space of role players who can lead the task execution), when required. This is in contrast to building and investigating models of the context of the environment to discover the physical boundaries of sub-ideal situations as this approach overlooks the impact of context-awareness technologies and human capabilities in identifying and managing these situations.

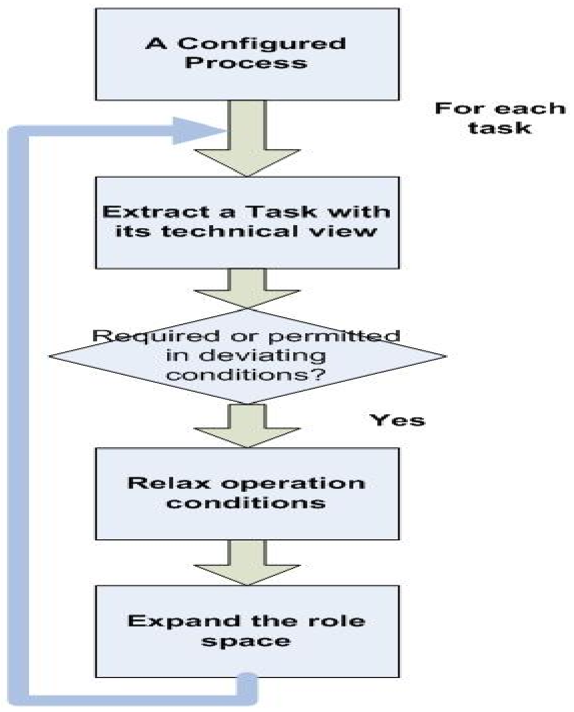

A Configured Process
For each task
Extract a Task with its technical view
Required or permitted in deviating conditions?
Yes
Relax operation conditions
Expand the role space

Figure 5.11 A simplified view of the proposed methodological steps

In more details, I propose the following steps in a systematic and discursive way as shown in Figure 5.12:

(1) Consult the system's process architecture to unite business goals with business process. This is *sin qua non* unless the methodology implementer starts from this point, the architecture will be shaky. As the methodology adopts a self-adaptive approach, goals should be identified first in a top-down manner.

(2) Assess the business processes of the system at hand taking into consideration the differences between the *transactional* and *collaborative* natures of these processes. Steps 3 to 6 should be iterated for every process.

(3) Based on this understanding, select a *routinised* process that operatioanalises a high level system goal/ requirement e.g. organisational goal.

(4) System analyst should concretise the relation between the selected process and its goal(s) through refining the process's constituents from the transactional point of view. Such a refinement incurs having all the required configurations for the entire process e.g. use case-like in the light of *required technical configurations* as if the process's technical resources and the actors are properly configured before hand to enact the entire process alternatives (c.f. use cases).

As such the configuration is pertained to a key actor who should guide the process progress (i.e. namely the key player) provided that the process is human-driven, (i.e. not purely mechanistic), otherwise the modeller should stop here and select another process (i.e. step 3).

(5) The system analyst should divide the process's use case-like static configuration into purposeful sub-configurations (e.g. a configuration should satisfy a specific functional requirement or a user requirement, which may contain several technological components serving a number of actors. This step is divided into sub-operations according to the following sequence:

   i. Constructing coordination interfaces (i.e. domain level): this step can be iterated as long as new pieces of software, equipment and their business-oriented exploitations are added to the socio-technical system

   ii. Accumulation of every actor interfaces to realise purposeful business needs from the architectural configuration's point of view and assign the permission for enacting the defined operations in the interface

   iii. Determining coordination contracts that superpose the required functional behaviour of small configuration steps on top of the coordinated entities and organise them on actor basis (i.e. coordination contexts) to realise the use case-like space of processes (i.e. configuration steps)

(6) Elicit the organisational structure depicting functional roles of the organisation. This should include the hierarchical relationships between organisational roles where lines demonstrate real inheritance of capabilities and knowledge (e.g. consultant-doctor and specialist-doctor relationship) rather than supervise or report-to relationship (e.g. manager and engineer one).

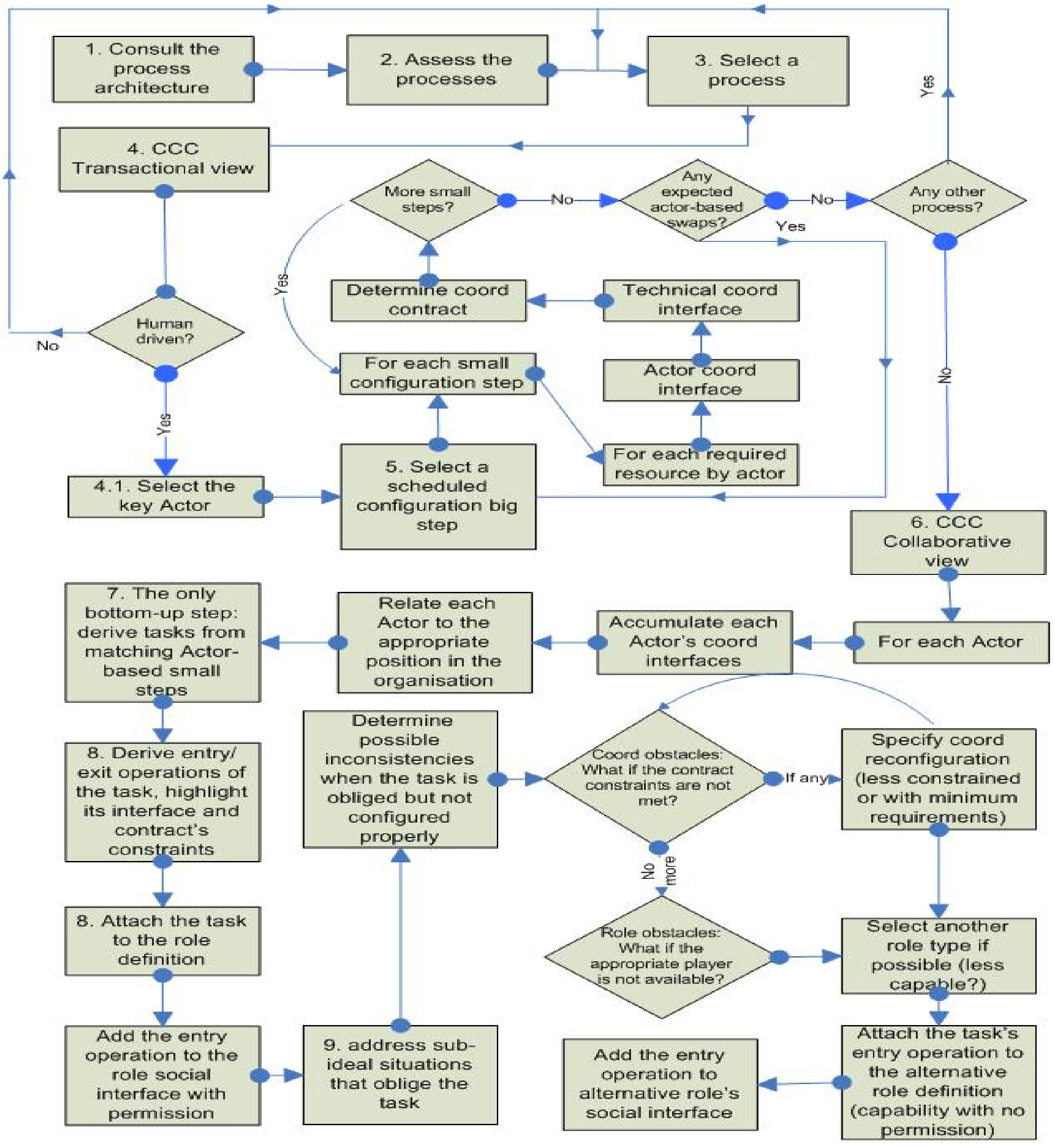
(7) Attach tasks to roles as the system designers answer the following questions after constructing each task: who is the natural role player of the task? In what natural context this task can be permitted? And what is the optimal system technical configuration required to achieve the task. Answering these questions yields the transactional view of the task executions within the socio-technical systems.

(8) Identify and analyse the inconsistencies, either the static inconsistencies in role structures and their relationships, or the dynamic ones, which may require tools for animating different executions. The main aim of this step is to find and remove obstacles that hinder the emergence of the required overall system behaviour. These obstacles can be removed by permitting or obliging the task in hand. This should be followed by imposing sanctions or rewarding the corresponding social behaviour afterwards. Obstacles are identified as sub-ideal situations that considered problematic as they lead to a hazard (i.e. usually related to valuable resource or a system objective) and can be managed by the re-definition of a relevant purposeful task. The redefinition of the task in hand can be achieved through removing one or all of the following obstacles subcategories:

i. *Role obstacles*: identifying roles that are capable yet not permitted to commence the required task allows reasoning about them to facilitate role-based configurations (i.e. role transitions) at runtime. Thus, system designers should allocate the task-capability space of each task to the role hierarchy by identifying:
    1. The least capable role (c.f. the abstract class in dynamic binding)
    2. The optimal role (c.f. the first concrete role)
    3. Redefinition roles (roles that are allowed to redefine the task excluding the entry/exit operations

ii. *Coordination obstacles*: In the 3Cs approach a task's requirements are modelled through coordination rule invariants and/or the specification of required technical entities (i.e. other partners). Both can hinder the execution of the task when it is urgently required. Thus, this step supports allocating task *redefinition* to the above mentioned roles within the role hierarchy. Such re-definitions include reducing technical requirements (e.g. engaging less technical resources) or weakening contextual constraints and they are specified in the corresponding social laws.

(9) Based on the previous step, the system analyst plans what sort of self-adaptivity should take place when the interaction under focus is triggered and/or obliged beyond the coordination scope, in order to handle sub-ideal situations and provide runtime evolution. The result would be a set of social laws that, once triggered, create normative positions normative positions to empower role players so as to compel them to execute the required/permitted interaction by means of facilitations and sanction.



1. Consult the process architecture
2. Assess the processes
3. Select a process
4. CCC Transactional view
More small steps?
No
Any expected actor-based swaps?
No
Any other process?
Yes
Yes
Yes
Determine coord contract
Technical coord interface
Human driven?
No
For each small configuration step
Actor coord interface
Yes
No
For each required resource by actor
4.1. Select the key Actor
5. Select a scheduled configuration big step
6. CCC Collaborative view
7. The only bottom-up step: derive tasks from matching Actor-based small steps
Relate each Actor to the appropriate position in the organisation
Accumulate each Actor's coord interfaces
For each Actor
Determine possible inconsistencies when the task is obliged but not configured properly
8. Derive entry/ exit operations of the task, highlight its interface and contract's constraints
Coord obstacles: What if the contract constraints are not met?
If any
Specify coord reconfiguration (less constrained or with minimum requirements)
No more
8. Attach the task to the role definition
Role obstacles: What if the appropriate player is not available?
Select another role type if possible (less capable?)
Add the entry operation to the role social interface with permission
9. address sub-ideal situations that oblige the task
Add the entry operation to alternative role's social interface
Attach the task's entry operation to the alternative role definition (capability with no permission)

Figure 5.12 The proposed methodological steps

Steps (2) to (5) allows constructing the transactional view of the architectural approach while the rest of the steps allow reasoning about certain unexpected or blurred participants' interactions that have never been taken into account in traditional software architectural approaches e.g. the 3Cs.

This methodology facilitates the management of emergent collaborative processes at runtime and keeps the overall stability of the system by providing the appropriate response to these unexpected interactions that cannot be causally controlled.

5.4.2 Remarks on the Proposed Methodology

The main aim of this methodology is to bring to the fore the detection and the management of sub-ideal situations and biddable social interactions in order to guide the system reconfiguration to self-adapt to changes of context. Reconfigurations include removing role obstacles and coordination obstacles to realise these interactions, when needed.

These steps support the extension of the 3Cs approach. The 3Cs approach is based on eliciting and modelling aspects such as business rules, functional requirements and completely anticipated design time reconfigurations (i.e. programmed and ad hoc reconfiguration). The processes of eliciting these aspects from specifications, encoding them through primitives at both coordination and configuration level, and executing them at runtime are relatively easier than electing, and managing uncertainties (e.g. unexpected context changes and social interactions).

The above-mentioned methodology steps address a new class of communication/interactions within systems that is not purely Human-Human (i.e. negotiation-based) or simply HCI but rather a class of human-driven and task-oriented interactions (e.g. human using equipment and software pieces) that have purpose and affect social and technical contexts through transitional and emergent collaborative processes. While the transactional view of the architectural model is sufficiently addressed by the 3Cs primitives such as coordination interface, coordination laws and coordination context, the non-normative collaborative view has just been equipped with new primitives for capturing possible enactment of unexpected interactions through availing entries for emergent tasks that aim at attaining well-identified short-term goals.

One of the key issues in this methodology is justifying the order of its steps, which relies heavily on dependencies between social or process-oriented tasks and coordination interfaces. The perception of this thesis is that tasks are collections of coordination interfaces, and thus, the latter should be defined first as they can be freely extended without changing the corresponding task template. Moreover, assigning tasks and/or their entry/exit operations to organisational roles in terms of social roles gives these roles a multi-dimensional semantics and allows understanding the interactions of the role player (i.e. social participant) in process terms making it possible to address and manage “hot task swapping”.

Beside social roles, social laws deal with sub-ideal situations and/or unexpected interactions as they surface. The critical point is which knowledge of the system participant is required when they are invited to join it or when they enact it. Moreover, system specifiers must allocate interactions to which implicit social meanings can be ascribed that convey the need for *hot process swapping*. This exactly specifies what emergent behaviours be pre-planned before hand and then tackled at a higher abstraction level. Therefore, it is better to concentrate on the entry operations of organisational tasks/processes and relate them to the participant’s capability representation in the model.

Another argument to support the order of these steps is the fact that positions in organisations are relatively stable, particularly if they are capability-based, whereas tasks and their technical details keeps changing as new technologies are introduced to the system. Thus, if smarter equipment is introduced to the system resources, then another round of configuration-based analysis has to be performed to re-evaluate the tasks in which this equipment participate. Such re-evaluation procedure may entail the expansion of the scope of roles that can enact this particular task as lower levels of knowledge and skills would be expected from the human participant’s side. This change can be easily tackled with minimal efforts, as the task will be fit to a new appropriate role that is closer to the root of the role hierarchy (i.e. parent roles). Chapter 7 provides a case study that demonstrates the applicability of the proposed methodological steps and evaluates their outcomes.

5.5 Related Work

Many insightful and interdisciplinary research efforts have targeted social interaction modelling issues within information systems, but from different perspectives: (Checkland 1984) Software Practice (Floyd 2002), Ethnography (Martin & Somerville 2006), norm-based requirements analysis (Stamper 1994), Computer supported

Cooperative work (CSCW) (Moran, Thomas *et al.* 1990, Grudin 1994, Zhang, Xu & Gu 2005) and Groupware systems (Ellis, Gibbs & Rein 1991, ter Beek, Ellis *et al.* 2003).

The view of (Taveter & Wagner 2001) to business processes as social interaction processes for the purpose of doing business matches the presented approach. In the advocated methodological approach, biddable interactions have been put forward as first-class citizens and a sub-system for norm-based has been joined to the knowledge domain to manipulate its settings, i.e. role settings. Their knowledge domain consists of a role model and a configuration model that is shared with the general-purpose machine *M*.

The Problem Frames approach to requirement specifications and decomposition recognises the distinctive characteristics of biddable domains and captures their phenomenal relations with both problem and solutions domains (Jackson 1995, Jackson 2001). Early studies in the approach propose conceptual structures to model reactive systems that interplay with social systems, utilising symbolic interactions and norms to bring the required effects to the social system (Wieringa 2000). An extension of Problem Frames towards the realm of socio-technical systems and organisational modelling has been introduced in (Brier, Rapanotti *et al.* 2004), and carried further towards bringing together high-level business requirement and low-level Problem Frames through AFrames patterns in (Hall, Rapanotti *et al.* 2004). Additionally, a *change frame* has been put forward to facilitate the analysis and synthesis of organisational-driven change in socio-technical systems (Brier, Rapanotti *et al.* 2006).

The presented methodology moves the 3Cs approach a step forward in providing flexible and to evolvable architectural based systems. Current software development methodologies treat system participants as stable elements who always react to the system in a predictable and “rational” manner. In short, all these methodologies take a technology-centric approach to system analysis to seek the best design of the system (Checkland & Scholes 2001).

The first departure from this assumption in the architectural modeling paradigm was presented in the Aura project, which showed the effectiveness of using architectural layered models in addressing self-healing mechanism (Garlan, Siewiorek *et al.* 2002). Aura’s approach presents the user intent and makes available to the rest of the system a powerful basis on which *user* needs can be anticipated and then answered through system adaptations. The self-adaptation approach in this thesis is top-down and takes into consideration the differences between the transactional and collaborative natures of human and mechanistic processes as proposed in (Harrison-Broninski 2005). His work is based on maximising the reliance on role models and their dependencies within organisational settings as shown in (Ould 1995) to reason about interactions in different contexts.

A candidate approach to achieve a stateful organisational model with separation of concerns has been demonstrated by (Zhang, Xu *et al.* 2005), which put forward the Organisational State Machine (OSM) and Role-based State Machine (RSM) over which the system norms are applied on CSCW interactions. A normative analysis approach has been proposed by (Liu 2000, Liu, Sun *et al.* 2001b, Kayser & Nouioua 2004) but at the requirement level; however, an adapted version of this approach was proposed in this chapter that provides methodological steps for capturing requirements for social laws that can be specified at the architectural level. The aim of these steps is to address, analyse and support the software development method for modelling and implementing social, collaborative and organisational systems in organisational settings.

5.6 Discussion

The methodological approach at hand supports software-intensive systems, which operate in organisational settings, and demonstrate a process-aware view of their interactions. The process-aware view is achieved through the execution of steps 5 to 8 in the proposed methodology, which links functional goals to processes; correlates tasks to purposeful participants' intentions and/or stable-state preserving requirement; and poses the signified task's entry/exit operations as communicative behaviours that explained by the voluntarily enactment or the ignorance of the imposed obligation towards these operations. These communicative behaviours are considered biddable interactions, which are put forward as first-class citizens upon which a sub-system for norm-based governance has been joined to enrich knowledge domain and to support self-adaptivity and manipulate the existing configuration accordingly, i.e. role settings. The proposed extension provides a configuration model that consists of two views: a role view and a configuration view. Both of them are managed by the general-purpose machine *M (*i.e. the harmonizer as mentioned in Section 5.2).

In the proposed methodological approach, conflicts or deviations from normal situations can be used as an efficacious way of identifying perspectives for dependencies, deriving reconfiguration and also for locating permissions and obligations. Permissions and obligations have to be adjusted to respond to biddable interactions and changes in the environment in which social entities operate. Practically, the need for adaptation may result from monitoring services such as (Baresi, Ghezzi & Guinea 2004) and imply reconfiguration of roles and technical components.

Chapter 6

Graph-based Formalisation & Meta-Modelling of Socio-technical Protocols

"*Poor notation can cloud important concepts but notation alone cannot rescue inadequate concepts.*"
Cliff B. Jones

6.1 Overview

Graphs are among the elegant and most universal models for a variety of systems that include not only computer science, but extend to engineering and biological sciences. Agile software architectures require—as first-class concern—ways to model how to predict, support, or react to situations in which systems should evolve in a way that keeps their overall good state. The Graph Transformation approach (GT) combines the idea of graph, as a universal modelling paradigm with a rule based approach to specify the evolution of the system.

This chapter introduces two graph-based approaches namely the view-based approach and the Dynamic Meta-modelling approach, for specifying an evolutionary architectural modelling method that aims for constructing normative models for evolvable and adaptable socio-technical protocols. The key target of this chapter is to demonstrate a formal definition of the generic reconfiguration operations that have been defined in

Section 4.5.4.2 and their impact on monitored and evolvable software architectures particularly at deviating contexts. Herein, graphs precisely define roles as structural semantics, while the laws' reconfiguration operations are specified through the operational semantics given by graph transformations.

6.1.1 Objectives

The main objective of this chapter is introducing a graph-based modelling approach to address the following issues:

(1) modelling the *intertwining* between the *well-separated* technical and social aspects of socio-technical protocols

(2) proposing the use of GT rules to formalise the operational semantics of the generic reconfiguration operations in which graph rules provide precise semantic specifications needed to be interpreted by the configuration manager so as to reflect the way the systems at hand should evolve in response to biddable interactions and/or sub-ideal situations

(3) allowing tool support to validate and animate the structural and operational properties of the instantiated architectural configuration based on the GT rules.

Two graph transformation based approaches are examined to achieve the above-mentioned objectives. It can be argued that the mapping between the textual language and its graph based semantics is very obvious and there will be no contribution in defining mathematically this mapping. The challenge is to provide a both structural and operational semantics that reflects the relationship between the social aspects related to biddable interactions of human participants, on the one hand, and the and the system response to theses interactions, on the other hand.

With regards to operational semantics, it is of great importance that the reified architectural configurations, on which reconfiguration operations will be applied, should be extended to incorporate the *state* notion, which constitutes a departure from traditional meta-modelling approaches and establishes the foundations for defining the operational semantics of the targeted model.

6.1.2 Incentives for Using Graph Transformation Approaches

Mainly, there are four motivations for the usage of GT techniques: specification, description of systems, model transformations and formalising concrete systems (Ehrig, Engels *et al.* 1999). They allow filling in the gap between the *state-transition* based view given by program code and concealed *state-based* view needed for comprehending and reasoning about behavioural properties of certain system configurations. In other words, programs do not illustrate sequences of states but rather sequences of transitions that emerge from a set of instructions of how to query and then manipulate the current state to move to another. Hoover, made a simple comparison between programs and musical scores that resemble state-based models (Hoover 2006): "In a musical score the instructions tell the performer what state the music is in at any instant of time. The opening of the score for Beethoven's 5th says to play G three times followed by E. You can open a score at any point and know immediately what the music sounds like at that point. A musical score is a sequence of states that the music is in. The meaning of the score is out in the open for all to see. [...] It's as if the opening of Beethoven's 5th was described as follows: "Start at G. Play a note. Play a note. Play a note. Go down a minor third (3 semi-tones). Play a note." You would not be able to simply look at the score and see or hear what the music sounds like mid-piece. In a program, the meaning is all between the lines!"

Comparing with programs, a graph is an abstraction of a system state. It projects the part of the state, which remains constant between two events states (i.e. interface graph), as well as all possible event-instances that might be consumed by the state at hand, in terms of pre-conditions and rewrite rules. The rewrite rules have to cover all the effects of the events in the abstract model. Thus, graphs and graph transformations enable checking consistency and completeness with regards to structural properties and provide a way to query states and animate possible changes starting from a given state.

Additionally, graph models, in contrast to textual ones, are intentionally more intuitive and suggestive; however, their meaning must be clear to avoid misunderstandings and mistakes. Like in the case of text-based modelling languages, there are the two possibilities of *operational* and *denotational* semantics to equip these models with the required sort of formalisation to ensure properties such as consistency and maintainability.

Among the above-mentioned incentives to exploit GT techniques, this thesis aims at describing valid socio-technical protocols through a synthesised type graph that can be populated with concrete components and specifying architectural reconfiguration that refer to control of self-adaptation of architectures based on given organisational/technical rules. Technically speaking, this chapter exploits GT to check the consistency of possible role-based reconfiguration by means of a formal architectural based model that takes role as structures and specify the effect of social laws' generic reconfiguration operations in terms of precise operational semantics given by transformations over a subset of the synthesised type graph.

These rules are interpreted by the adaptation manager (i.e. the harmoniser) as they specify how the system should respond to identified sub-ideal situations and role violations. Controlled by this adaptation manager, social laws can be applied proactively (i.e. with regards to social interactions) to contextual changes or reactively to unexpected social interaction. The approach emphasises the analysability of the generic role-based reconfigurations space pertained to social components. Additionally, I promote abstract transformation rules that support generic concepts e.g. social roles and tasks to secure the generality of the approach. Finally, this chapter highlights some preliminary elements for extending the graphical approach towards correlating both the technical and the social view of the synthesised type graph (i.e. task-interface relationships) through model-based transformations.

6.1.3 The Chapter Structure

This chapter introduces a survey on the basis of GT in Section 6.2. Section 6.3 discusses the challenges to face with regards to modelling socio-technical protocols. Section 6.4 presents the view-based graph transformation concepts and abstractions that deem to be valid for capturing the particularities of both technical and social views. Section 6.5 examines the capacity of existing meta-modelling approaches with regards to providing a semantic model for socio-technical protocols. An extension of the meta-modelling approach towards formalising reconfiguration operations is presented in Section 6.6.

6.2 Graph Transformation in a Nutshell

GT emerged from extensively researched mathematical theories and supported by various tools for validating and analysing graph-based modes. This computer science paradigm was put forward as an answer to the drawbacks of classical approaches to rewritings, like Chomsky's grammar (Chomsky 1956) and term rewriting (Klop 1992), in dealing with non-linear structures. Altogether, the notion of GT has been used to realise and combine the concepts of graph grammar and graph rewriting.

The first milestone along the path to establishing algebraic basis of the graph transformation approach was presented by (Ehrig, Pfender & Schneider 1973) which gave the inspiration for collective research work in this area including mathematical foundations (Rozenberg 1997), applications-oriented research (Ehrig, Engels *et al.* 1999) and concurrency, distribution and parallelism issues (Ehrig, Kerowski *et al.* 1999). Among others, the Double-Pushout approach to graph transformation (DPO) (Corradini, Montanari *et al.* 1997) has proofed to be suitable for modelling reconfiguration as transformations. More recently, graphs and graph transformations have been successfully used for modelling the following:

(1) architectural modelling:
    a. specifying architectures and their computations using various underlying formalisms such as process calculi (Allen, Deuence & Garlan 1998, Mètayer 1998, Canal, Pimentel & Troya 1999) and rewriting of labels (Hirsch, Inveradi & Montanari 1998).
    b. architectural reconfiguration approaches to which close attention will be paid such as (Wermelinger 1999, Hirsch, Inveradi & Montanari 2000, Wermelinger, Lopes *et al.* 2001).

(2) representations and model transformation (Taentzer, Ehrig *et al.* 2005, Biermann, Ehrig *et al.* 2006).

6.2.1 A Formal Basis of Graph Transformation

The basis of the presented approach to modelling architecture and architectural reconfiguration are formal GT systems. For that reason, a short introduction to the formal definition of graphs, graph morphisms, graph transformation is given in the following. For further explanation, (Rozenberg 1997, Baresi & Heckel 2002) presents the solid mathematical foundations of graph transformation.

Definition 6.1 (Graph)

Let a graph $G = <N_G, E_G, s_G, t_G>$ consists of two finite sets $N_{G,}$ and E_G of nodes and edges, two source and target functions: $s_{G,}\ t_G\colon E_G \rightarrow N_G$. Graphs are related by graph morphisms, which map the nodes and the edges of a graph to those of another one. Graphs, in addition to graph morphisms form the category Graph.

Definition 6.2 (Graph Morphism)

Given two graphs $G_i = <N_i, E_i, s_{Gi}, t_{Gi})$ i∈[1,2], a graph morphism f: $G_1 \rightarrow G_2$,

G_2, $f = (f_{Ni}, f_{Ei})$ consists of two functions, f_N: $N_1 \rightarrow N_2$ and f_E: $E_1 \rightarrow E_2$ that preserve the source and target functions, i.e. $f_N \circ s_{G1} = s_{G2} \circ f_E$ and $f_N \circ t_{G1} = t_{G2} \circ f_E$.

6.2.2 Semantic Choices for Graph-based Modelling

This subsection introduces the semantic choices that have to be taken when modelling with graph, such as: which notion of graph to adopt? What conditions should a resulted graph fulfil and the different ways to specify the transformation rules?

6.2.2.1 Type Graphs

To allow graphs to describe models of abstract things, especially complex systems, they should be backed with comprehensive and consistent modelling techniques.

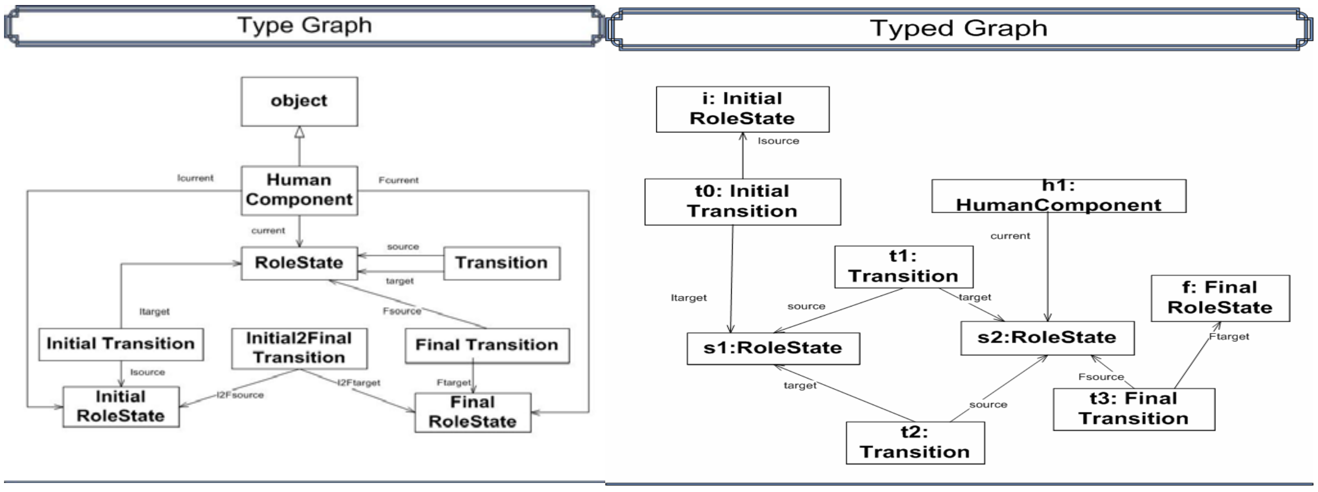


Figure 6.1 Type and typed graphs

A type graph *TG* in the sense of (Corradini, Montanari & Rossi 1996) defines collection of types and interconnection constraints to which the instance graph *G* conforms. A graph *G* belongs to *TG* class if u can find for each node and edge in *G* the corresponding node and edge type in type graph *TG*. A type graph is a “filter” that restricts the allowed types of the instantiated nodes as well as types and cardinality of edges that connect them to populate an instance graph (i.e. *typed graph* as shown in Figure 6.1).

6.2.2.2 Labelled Graph Grammars vs. Attributed Graph Grammars

Typed graphs as above correlate the valuation of their nodes and edges (i.e. *labels*) to types defined in the type graph structure and its label set. *Labelled graphs* are less constrained graphs as their nodes and edges conform to a label set but without a graph structure (Baresi & Heckel 2002). More concretely, if nodes and edges are labelled over a collection of independent label alphabets L_N, L_E, the relational variant is given by (N, E, l_v) with $E \subseteq N \times L_E \times N$ and $l_v: N \rightarrow L_N$. Another variation of labelled graphs is the *attributed graph* form in which labels refer to pre-defined abstract data types such as strings or natural numbers (Löwe, Kroff & Wagner 1993). Naturally, when attributed graph instances respect the structural constraints of a type graph they are called typed attributed graphs in which nodes may represent classes in, the object oriented sense, containing abstract data types and their operations. Among other implementations, attributed graph grammars are popular in describing visual languages (Bardohl 2002). Figure 6.2 shows taxonomy of graph's types.

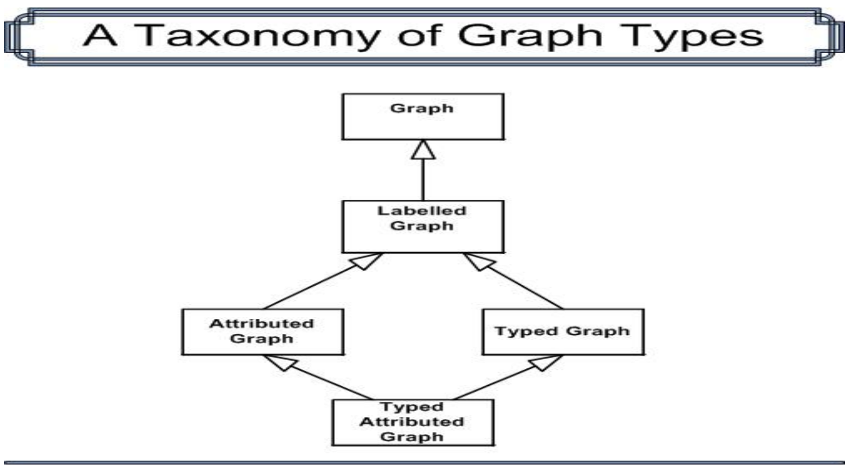


Figure 6.2 A taxonomy of graph types

6.2.2.3 Clan Morphism and Modelling Inheritance

From a philosophic point of view, *instance-of* relationship between the instance graph nodes their counterparts in the constraining graphs (e.g. type or type attributed graphs) could be model at the abstract level. A modeller should decide what properties are inheritable and what are not. In what follows, a presentation of a formal definition of a *hierarchy graph I* is demonstrated, as advocated by (Bardohl, Ehrig *et al.* 2003, Ehrig, Küster *et al.* 2006), for typed graphs and the extended work towards attributed typed graphs (de Lara, Bardohl *et al.* 2007). I borrowed these concepts from those research contributions to establish a graph-based inheritance model that supports hierarchical structures of roles and tasks.

6.2.2.4 Definition 6.3 (Inheritance Graph)

Let I be Inheritance graph $I = <N, E, s, t>$, where N is a finite set of nodes. The inheritance graph I shares the same set of nodes of N and a set of $A \subseteq N$, called *abstract nodes*. For each node n in I the inheritance clan is defined as follows, $clan_I(n)=\{ n' \in N \mid \exists \text{ path } n' \to * n \text{ in } I\}$ where path of length 0 is included, i.e. $n \in clan_I(n)$ is included. The sub-graph spanned by the hierarchy edge must be acyclic. Figure 6.3 extends the main constituents of the type graph *TG* depicted in Figure 6.1 by merging them with the inheritance graph I into a combined one. There is a single abstract node (*NamedElement*), which is shown in italics and is connected with rest of I by means of hollow arrows (i.e. is-a arrows).

6.2.2.5 Definition 6.3 (Type graph with inheritance)

A type graph with inheritance $G_{TI} = <G_T, I, A>$ has the following components:

(1) a type graph $G_T = <N, E, l_V>$

(2) the set of inheritance edges $I \subseteq V \times V$ which must not contain circles

(3) the set of abstract Nodes $A \subseteq V$

The key advantage of such type graphs is that they allow specifying abstract nodes that contribute to the conciseness of the type graph (i.e. comparing Figure 6.3 with Figure 6.1), and devising abstract graph transformations, which efficiently group similar transformation rule. However, if the hierarchical relationships are explicitly specified at the model level (i.e. type graph level), this would constraints the applicability of rules to application-specific concepts, and thus make meta modelling out of necessity.

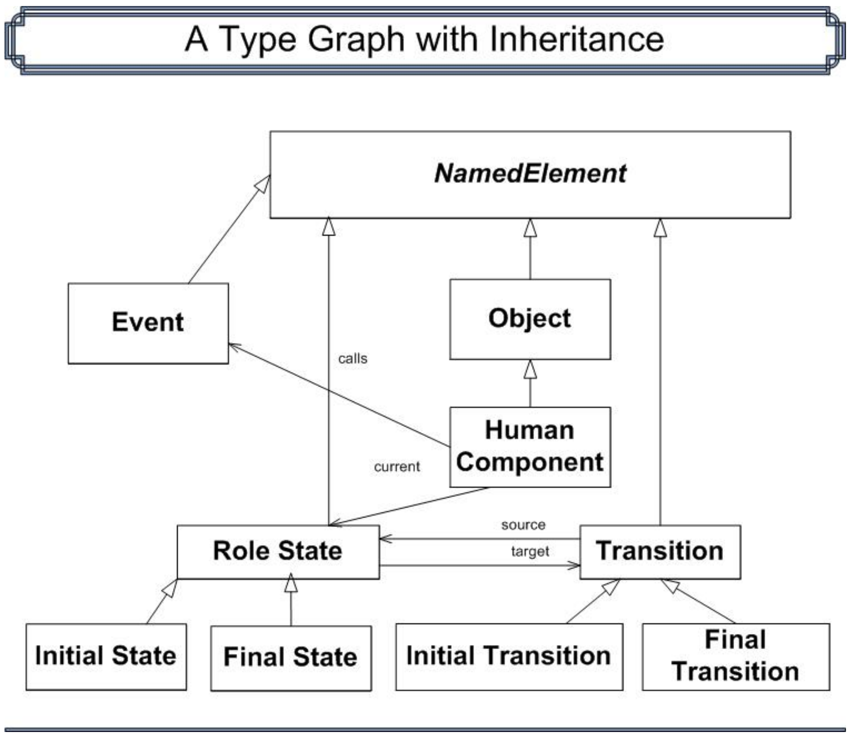


Figure 6.3 A type graph with inheritance

6.2.3 The Graph Transformation Approach

The main idea of GT is the rule-based modification of graphs shown in Figure 6.4.

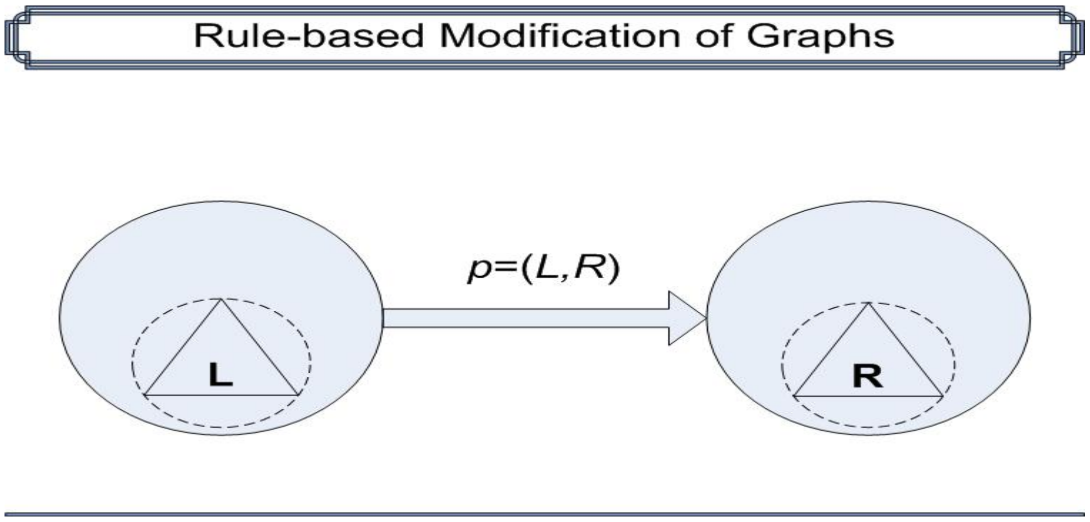


Figure 6.4 Rule-base Modification of Graphs

Formally, a GT rule or production p: $L \rightarrow R$ consists of a pair of TG-typed instance graphs $L \cap R$ such that the intersection $L \setminus R$ is well-defined (this means that, e.g., edges which appear in both L and R are connected to the same vertices in both graphs, or that vertices with the same name have to have the same type, etc.). The left-hand side L represents the pre-conditions of the rule while the right-hand side R describes the post-conditions. The left-hand side can also state negative pre-conditions (Negative Application Conditions), i.e. (NAG). Additional definitions with regards the double pushout approach to graph transformation can be found in literature.

6.2.4 Graph-based Modelling for Architectural Reconfigurations

Using graph transformations to model dynamic architectural reconfiguration in an abstract and visually compelling way seems to be a natural choice. Applying state-full transformations, particularly on labelled-graphs as suggested by (Hirsch, Inveradi *et al.* 1998, Hirsch, Inveradi *et al.* 2000), allows to perceiving dynamic reconfiguration as a rewriting process over graphs labelled with program instances (i.e., component instances) instead of just programs. This ensures that the state of components and connectors that are not affected by a rule do not change, because labels are preserved, and thus keeping reconfiguration and computation separate. This approach to modelling architectures has been advocated by Wermelinger and his colleagues (Wermelinger 1999, Wermelinger, Lopes *et al.* 2001, Wermelinger & Fiadeiro 2002), as well as Hirsch and his fellows (Hirsch, Inveradi *et al.* 2000, Hirsch 2003).

The research of Wermelinger et al. (Wermelinger, Lopes *et al.* 2001) presents an algebraic software architectural approach where architectures are modelled through labelled graphs that visually explicate instantiated components and their interconnections. Based on a categorical framework, they provide semantics that result from a mathematical computation (i.e. "Colimit") that convert the architectural diagram to an equivalent component representing the whole system on which computations and transformations can be performed (Fiadeiro, Lopes *et al.* 2003). This approach is anchored on the fact that performing computations on such categorical diagrams relates the architecture and the computational levels. With regards to reconfigurations, they are modelled as GT derivations, as defined in DPO, that are associated with additional constraints to preserve the consistency of the resulted graph (Wermelinger & Fiadeiro 2002). These constraints yield a reconfiguration step, which is a derivation from a given architecture (configuration)

G to architecture (configuration) *H*. The key contribution of this approach is extending the reconfiguration capabilities of an ADL-like language by specifying clearly the boarders between computations and configurations.

Additionally, a simple reconfiguration language (i.e. configuration scripts) has been incorporated to their approach in an attempt to utilise the formality of the ADL (i.e. CommUnity), however, these scripts lack formal basis, purely imperative (causal), and address only low-level reconfiguration operation, making it impossible to take into account higher-level evolution patterns or adaptation perspectives such as the social perspective. These observations also deemed correct with regard to the reconfiguration primitive (i.e. coordination context) promoted by the 3Cs approach. The following sections of this chapter, will study two graph-based approaches to give semantics to the interconnections between the social and the technical aspects as well as the operational semantics of reconfigurations: i.e. the view-based approach and graph-based interpreters for formalising operational semantics of reconfiguration operations.

6.3 Challenging Issues

6.3.1 Biddable Interactions Modelling

In graph-based modelling of user as agents, GT rules determine the overall effect of the interactions among (agents and objects) and describe local autonomous operations that the represented *human*s may react to regardless of changes in the environment or the *obligations* imposed on them by the organisation in which they operate. This view of agent *proactivity* and goal driven behaviour is a good candidate for enriching the concepts of my approach. One of the distinctive features of agent behavioural modelling i.e. autonomous operations—operations that are not triggered by a method call but by the detection of new objects/agents in the *LHS* of a GT rule— was cleverly modelled by (Depke, Heckel *et al.* 2000).

Agents act autonomously driven by their goals and plans sensing and reacting to environment, whereas in coordinated activities of business process management, agents (people) are invited to adhere to the prescribed behaviour; however, they are capable of violating these prescriptions.

6.3.2 Task-Interface Relationships

The proposed conceptual framework advocates self-adaptive mechanisms, which bridge the differences between bound coordination interfaces and the instantiated tasks. Difference and similarities between the two architectural concepts have been discussed in the previous chapter. Interconnecting tasks and coordination interfaces brings about additional information and emergent architectural properties that can be utilised to reason about possible reconfigurations even those that are not accepted in normal situations. This could be somehow analogous to the notion of software adaptors, which need to overlook definitions of communication protocols and type systems to capture anomalies to bridge applications that have compatible functionality but incompatible interface (e.g., (Yellin & Storm 1994)).

Architectural views are used to capture the semantics underlying the relationship between tasks and coordination interfaces. They are useful in representing coordination of actions from both the role perspective and the configuration context perspective. However, the way I chose to deploy architectural views makes a departure from classical architectural views. This decision is justified because the purpose is not projecting two views on the same model but rather to *interconnect* two models that have some intersection in their concepts in a way that allows determining their dependencies. Figure 6.5 depicts the intersecting parts between the two models.

Generally speaking, a satisfying solution would establish a *reference model* to match different yet corresponding concepts in both views. A holistic view of socio-technical protocol should comprise: (1) a meta class diagram (a structural diagram for the protocol elements and their interconnections), (2) an extension of the meta class diagram that caters for a meta state-machine to provide state information e.g. control state for technical components and role state for human components.

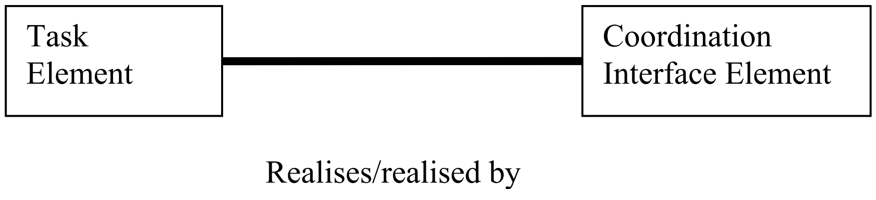


Figure 6.5 The reference model

6.3.3 Generic Graph Transformation Rules

It would be beneficial to have reusable transformation rules, which can be exchanged across different platforms or application domains. Issue such as abstract nodes hierarchies of nodes at different levels of abstraction, and instance-of relationship should be tackled with care. In GT, abstraction techniques in which state graphs are reduced by grouping nodes that are sufficiently similar—with regard to their behavioural properties. In this perspective, roles and tasks are exploited for social entities, and coordination interfaces for technical ones resulting in smaller states and a reduced evolution space. Moreover, the application of GT rules will be addressed in a higher level of abstraction, particularly at the normative view of the model. This approach is similar to the one proposed by (Rensik & Distefano 2006); however, they were targeting a feasible technique for model checking.

Table 6.1 describes different levels of abstractions in graph-based modelling which correspond also to textual language modelling. Building transformation rules that tackle higher level concepts such as role *R*, human components *H* instance or task *T* then these rules presents generic and domain-independent aspects of a system interpreting the semantics of the language using meta models. Conversely, these rules query and manipulates specific problem domain such as *GP*, patient, etc. then the model targets low level aspects of the language (i.e. concrete states).

6.4 A Graph-based Approach: Semantics, Views and Interconnections

The target of the modelling method at hand is to extend the 3Cs architectural language to address social interactions; therefore, this research proposes a domain-specific language for the *generic* domain of software architecture without relying on the concepts of a certain application domain or a platform. Additionally, it provides a graph-based integration between the architectural primitives that reflect the technical domain and those capturing social components and their biddable interactions.

6.4.1 Modelling Socio-technical Protocols

Socio-technical protocols are extended 3Cs sub(systems) that are queried and manipulated be configuration manager. A configuration graph is a labelled/attributed graph—as attributes are dealt with as labels. More precisely, a configuration graph is a graph where nodes are components labelled with instantiated interfaces and edges are connectors labelled with law type. It is reified by the configuration manager to control interactions of the instantiated architectures.

I developed a specific graph typing structure to distinguish between configuration entities (nodes) whose corresponding permissions are fixed (e.g. technical components and configured actors) and some other entities that hold permissions amenable to change at runtime, e.g. social entities enacting well-defined capability-based social roles. The reader may refer to Section 3.3.2 as an example for the former type of configurations and Section 4.5.4 for the latter type of configurations. An explanation of changeable permissions is provided in Table 4.1, with which social roles can be combined and then transiently manipulated by social laws. This typing structure yields a twofold representation of the configuration graph that comprises a *components configuration graph*, similar to (Wermelinger, Lopes *et al.* 2001, Wermelinger & Fiadeiro 2002), and the extension presented in this thesis i.e. *role configuration graph* that captures instances of roles, tasks and entry actions of social entities.

More concretely, the *components graph* is sufficient to reflect casual properties of software and mechanical components together with their interconnections but it falls short in providing a suitable representation of human components that are biddable and subject to organisational norms that can be violated. Conversely, the *role graph* includes a biddable dimension and an organisational dimension; the former addresses the biddable nature of human components, which requires non-causal modelling primitives; the latter are constructed for modelling human capabilities and permissions within an organisation. The bridge between the two-configuration graphs, which will be elaborated further in the next subsection, consists of the common human nodes and the edge between the targeted task and its associated coordination interface copy that defines the signature of operations and services included in this task.

The organisational dimension is clearly specified in terms of the explicit relationship between role elements, which reflect that organisational chart of the organisation to which participants belong as well as the formulating of the code of behaviour ascribed to key role players particularly in sub-deal situations.

The biddable dimension of human components within the role model and its enclosed labels of human-driven processes whose initiations generate speech-act-like communicative actions are explicated within the role internal structure to enable reasoning about norm enforcing/ violating interactions and contexts based upon them. Such reasoning should take into account the organisation's operational goals e.g. maintaining precious resources. The result of the reasoning process is achieved through self-adaptation, which include role transitions that traverse the role space graph pertained to the key role player. These transitions demonstrate the operational effect of sanctions or positive reconfigurations that are triggered due to the capture of norm violating behaviour or context.

6.4.2 The view-based Approach

From the point of view of architectural reconfiguration primitives, it is obvious that the model is too complicated to be captured by a single concern. Therefore, I aim for a semantic framework to support reasoning about the causal and normative reconfiguration separately. In this view, any reconfiguration operation manipulates the role and the component view of a socio-technical protocol in a different way. The reconfiguration operations defined in Chapter 4 are given loose operational semantics through an interpreter that execute alterations on both role and components view. Views generally and architectural views in particular have been defined in Chapter 3 (Section 3.2.1.7).

Before delving into the details, I emphasise on the vision of software runtime modelling techniques that draw a strict and a clear line between the *actual world* (the environment) and the corresponding representation of this world, namely the *model* (Dijkman, Quartel *et al.* 2003).

What this thesis strives to achieve through this multi-view graph-based model, is to relate the human view (i.e. *role graph*) with the technological one (i.e. *components graph*). More concretely, the world model needs to take into account changes caused by environment that are beyond the type/structural constraints of the coordinated view of the model under focus. This is required, particularly to allow certain unspecified changes (add/delete) of a graph's elements during the execution of a GT rule. This kind of transformation rules exhibits the loose semantics of open systems that have been introduced by (Heckel 1998, Heckel, Engels *et al.* 1999).

6.4.2.1 Communication between Views

With regards to executing graph transformations there are two distinctive approaches to model interactions via GT: (1) synchronous: through deploying Amalgamated Graph Transformation (Taentzer & Beyer 1994), and (2) asynchronous communications between views: (i.e. shared-memory) like between views where modellers use the reference model to present extra construct to mimic the shared memory together with the intersecting concrete constructs. Technically speaking, a (partial) specification is called a (view) on another specification, if the *renamed* version of the first can be embedded into the second. The formal basis of the view-based graph transformation system has been described by (Heckel, Engels *et al.* 1999).

6.4.2.2 Integrating Views

Views are integrated in two steps:

(1) Managing new dependencies (not covered in the reference model) by a model manager through: renaming, extensions

(2) Doing actual integration automatically

The first step is trivial as the reference model allows sharing domain-specific notions and operations. The new dependencies that require the intervention of the system modeller are problem-specific. The methodological approach that was presented in the previous chapter devises ways of introducing new tasks and/or technical components to the system specification. It must be emphasised that this view-based approach is meant to cater for language specification targeting software development. Therefore, it is not suitable for identifying inheritance and highly abstract behavioural patterns because a modeller cannot specify domain-independent concepts within the reference model. This approach excels only in supporting domain-specific frameworks as a starting point for software development projects. As a result, the reference model approach will be utilised only in constructing the interconnections between tasks and interfaces and cannot be extended to abstract nodes posing hierarchical structures i.e. roles.

In the light of theses fundamental discrepancies between technical and Role view, it is clear that the bridging between these views is not trivial. (Harrison-Broninski 2005) suggests that the integration or refinement between the two views is not possible simply by putting them all in one diagram. Inconsistencies such as:

(1) same names (views denotes semantically different concepts)

(2) different names yet representing similar concepts, Ontology and the efforts in databases research to solve schema integration problem.

In the light of theses fundamental discrepancies between technical and Role view, it is clear that the bridging between these views is not trivial. I suggest that the integration or refinement between the two views in sot possible simply by putting them all in one diagram.

6.4.2.3 The Reference Model

Types of architectural elements that are shared between the two views are modelled in an abstract way in a *reference model.* Such an abstraction would allow these constructs to be subjected to clan morphisms, renaming and extensions. The reference model, particularly, include the elements that should be represented in the underlying views by means of *open types systems.* Runtime configurations represent instances of architectural constructs: *coordination-based* and *role-based*. The component-based elements have inherited causal reconfiguration primitives, namely coordination contexts, which provide simple programmed reconfiguration operations.

The proposed reconfiguration language introduces a new model-based with an explicit single node inheritance mechanism through clan morphisms. This is in line with the approach introduced in (de Lara, Bardohl *et al.* 2007) that allows enriching the type graph of an attributed and typed graph model with abstract nodes and inheritance relationships. Such enriched graph types allow the specification of Abstract Graph Transformation rules c.f. domain independent transformation rules that give the operational semantics of reconfigurations.

The approach is anchored on a role model with three abstract node types: *role*, *task* and *human*. All of the three nodes might inherit definitions from their ancestors, however only the role hierarchy can be represented at the instance level. The argument behind these design decisions is to capture behaviour inheritance as a means of reusing specification in a way that is already captured by organisational chart.

Moreover, pushing the domain specific issue to the model level rather than the meta model, allows flexible entities sub-typing. In this perspective, The (causal) coordination view is typical to the graph-based approach to reconfiguration in CommUnity provided by (Wermelinger, Lopes *et al.* 2001), which can be associated to the 3Cs reconfiguration primitives (Andrade, Fiadeiro *et al.* 2001, Andrade & Fiadeiro 2003).

In what follows, an Open Graph Transformation system presents a modelling of the role's view of the system. Figure 6.6 puts together the *task* node, the corresponding *coordination interface* node and the runtime linking edge between them. The open types for deletion and addition are indicated by "-"and/or "+" markers in square brackets following type and attribute names. Attributes may be created and deleted along with their carrier objects. The results of a successful *reconfUnqal(role, entryOp)* adding new dispatch/enact edges and reconfiguring the connection with the appropriate coordination interface accordingly.

At any configuration, active human participants are enacting a role and this in turn should be bound to a task element. Tasks here represent an open type as it can changed in a way that is out of the role view control and this also applies to the coordination interface. The formal description of the integration of views is adapted from ((Heckel 1998, Heckel, Engels *et al.* 1999)) .

The essence of this integration approach is the Open Graph Transformation System, where open types can be specified for deletion and addition even independently from their carrying objects.

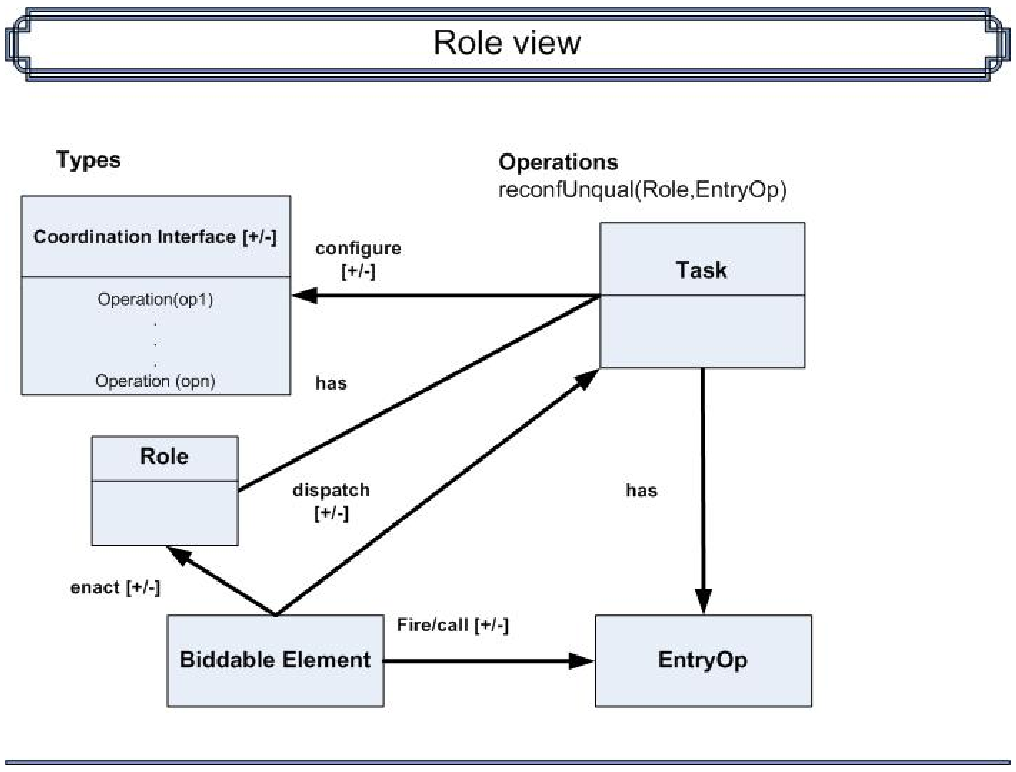


Figure 6.6. The role view of the Open Graph Transformation system

6.5 Meta Modelling for Reconfiguration

The introduced meta modelling approach lends itself more naturally to the goal of synthesizing architectural style elements (i.e. the 3Cs primitives and the newly introduced ones) in the sense of (Metha & Medvidovic 2003) where structures, interaction, data, behaviour and topology are the concerns of the architectural style. As has been discussed in Chapter 4, I am only interested in structural and topological properties' impact on possible reconfigurations driven by the behaviour of social elements participating in a configuration (i.e. socio-technical protocol).

From the GT point of view, styles define the structure and operations available to applications through UML-like meta model such as Meta Object Factory (MOF) in order to apply transformations over its instances so as to specify the dynamics of a style of that target a specific domain e.g. socio-technical systems (Cebulla 2004), mobile systems (Heckel & Guo 2005) and Service-Oriented Architectures (SOA) (Baresi, Heckel *et al.* 2006). MOF and GT can be integrated by identifying symbol classes that are associated with node types and associations with edge types (i.e. abstract syntax).

Level of abstraction	Languages abstraction	Visual modelling abstractions
3	EBNF	Meta Meta Model (MOF, EMOF)
2	A Language Grammar CFG (EBNF-based)	Meta Model (UML Stereotypes) with CFG representation)
1	Programs with control states	Models with control states (e.g. statechart diagrams)
0	Runtime Instances of states and configurations	Instance graphs with runtime states (attributes and labels)

Table 6.1 Different level of modelling abstractions for textual/visual language

6.5.1 Abstract Syntax

Abstract syntax graphs are forms of graphs to define language grammars. The purpose of forming an abstract syntax is to mark the starting point to define a language grammar through which simulation and transformations will be applied on the models that are represented by this language (Bardohl, Ehrig *et al.* 2004). For instance operational semantics can be derived by ascribing a sort of “state notion” to the abstract syntax to allow an interpreter to execute steps by applying transformations. Figure 6.7 illustrates a MOF-agnostic abstract syntax of socio-technical protocols.

6.5.2 Mapping the abstract syntax to the Textual Syntax of the Reconfiguration Language

The mapping from the subset of the extended 3Cs configuration language i.e. social laws and roles (see Sections 4.4.2 & 4.4.4) to a graphical abstract syntax elements is a pre-condition for providing the operational semantics of the reconfiguration operations that manipulate models of socio-technical protocols.

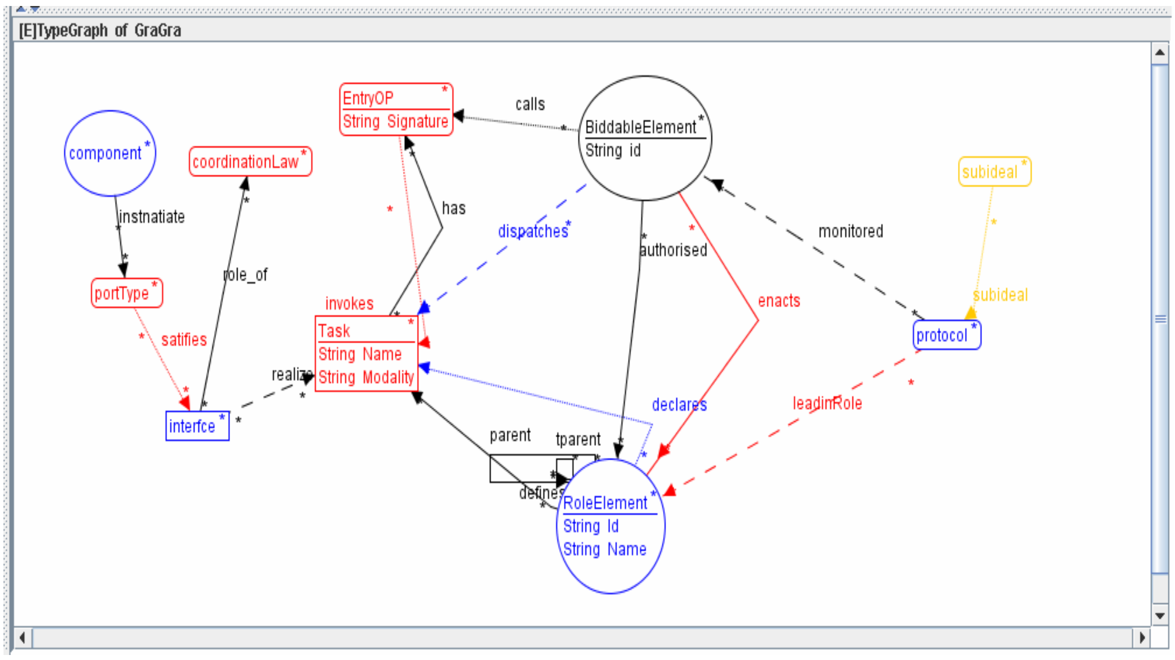


Figure 6.7 A MOF-agnostic type graph (initial integration)

I argue that the mapping process is intuitive and does not need mathematical proofs, as the proposed simple individual text-icon mappings are capable of composing more complex semantics. Pairs that resulted from the mapping of the basic elements of the proposed reconfiguration language are defined in a tabular form (Table 6.2), where every graphic model element is uniquely mapped to a corresponding textual keyword. For example, red rectangles correspond to the task keyword in the reconfiguration language.

Table 6.3 presents excerpts of more complex expressions in the textual modelling language and how they are composed from the aforementioned basic graphical notation.

Textual Element	**Graphical Element**	**Meaning**
social law	NA	
social role	○	Role as a name along with its collection of tasks and hierarchical relationships
Task	▭	Tasks are lists of attributes
biddable element	○	Name anchored to set of role types (high level roles)
Operation	▢	entry_operation that labels a task
operation {call}	⋯▶	A biddable element *calls* an entry_op
entryOp.task() {has}	──▶	1-1 relationship (every task labelled with one entry_op
anchored role	⋯▶	{enacts} a biddable element *enacts* a role i.e. role is enabled either directly or via inheritance
Specialise	──▶	Role-role hierarchical relationship
enabling state	- - -▶	{*dispatches*} link biddable element and task which refers also to coordination aspects
[ ] action	⋯▶	{declares} the operation is part of the player's capabilities, but the permission is not institutionally granted. The capability is inherited to children as is.
[+] action	⋯▶	{defines} the permission of this action is institutionally granted and is inherited to children

Table 6.2 Excerpts of basic text-graph elements mapping

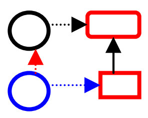
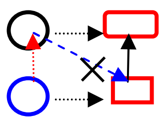
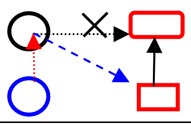
Textual Element	**Graphical Element(s)**	**Meaning**
Unqualified operation		operations of the anchor role, which are executed by social components that have no qualification
operation and not enabling state		operations for which the anchor role is qualified but are initiated in a context in which they are not permitted
active state and not operation		operations of the anchor role that are not executed in contexts in which they are required

Table 6.3 Graph-based mapping of complex expressions

6.5.3 Specifying Operational Semantics over Abstract Syntax

To specify the operational semantics, at higher level of abstraction, the language notation is based on meta modelling. The proposal to operational semantics is in line with the Dynamic meta modelling approach (DMM) (Hausmann 2005). This approach exploits both GT rules and meta models for specification of operational semantics of a visual modelling language. It extends the work presented in (Plotkin 1981) and (Corradini, Heckel & Montanari 2000) in which Structured Operational Semantics (SOS) are augmented with abstract syntax graph in addition to a state notion (i.e. statechart diagrams).

This approach was carried further by the research work of (Hausmann 2005) which comprises: (1) SOS: abstract tree augmented with a statechart diagram and (2) graph transformation rules to specify semantics.

6.5.3.1 Abstract Syntax Meta Model

The abstract syntax meta model constitutes of a type graph and an augmented MOF-defined statechart machine to keep the protocol's configuration state with regards to roles/tasks. Figure 6.8 illustrates the Abstract syntax together with some productions starting from an initial graph.

The formalisation of the presented reconfiguration language relies on two main pillars: the static view, which is represented by the abstract syntax language on the basis of the UML/MOF extension, and the GT rules that provide its operational semantics, in the sense of Graphical Operation Semantics approach (GOS) (Corradini, Heckel *et al.* 2000) to formalise the derivation of the behaviour of the model specified by reconfiguration primitives.

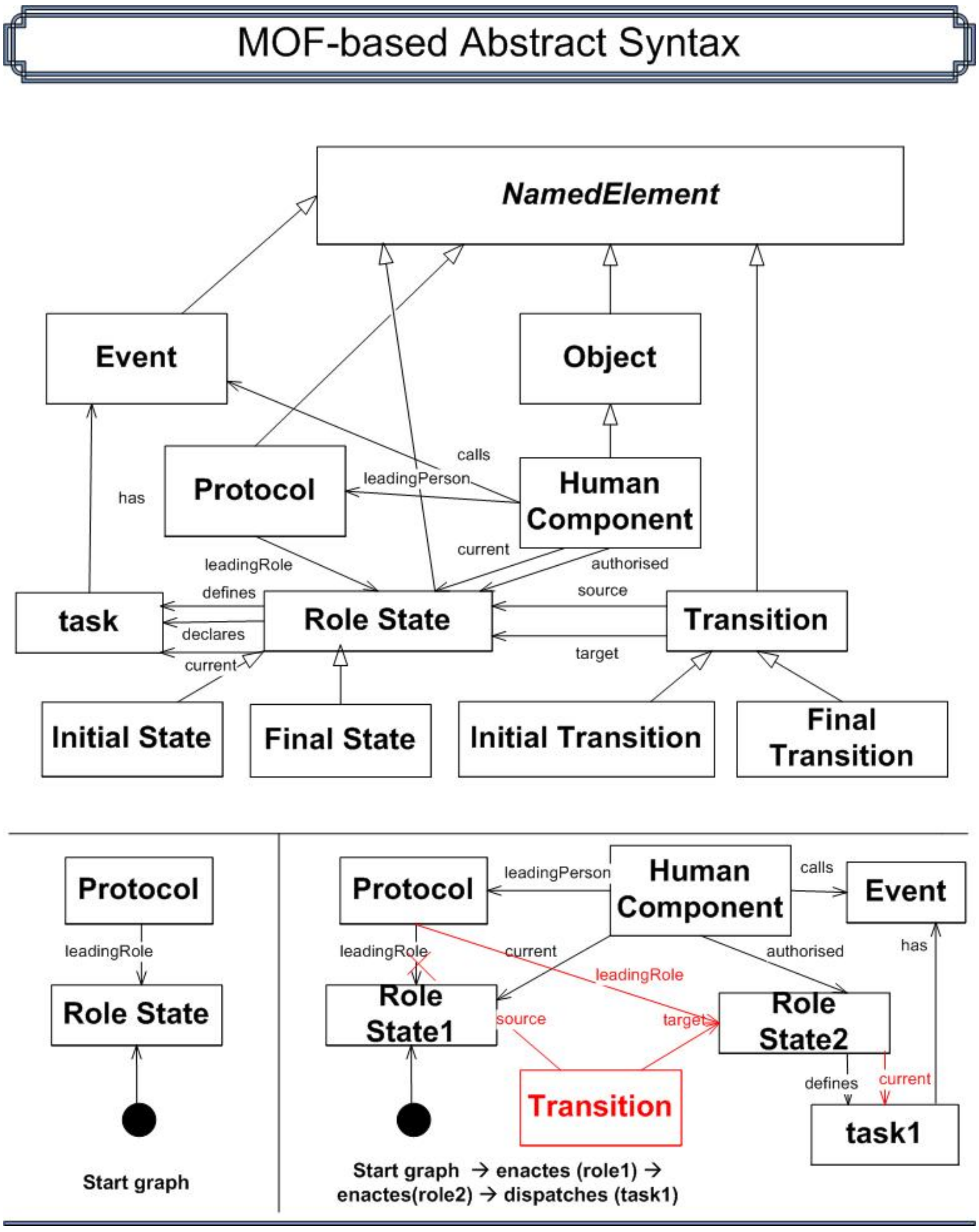


Figure 6.8 The MOF-based abstract syntax

6.6 Discussion

At formalised view-based graph transformation semantics that was presented in (Heckel, Engels *et al.* 1999) is utilised for the 3Cs extension presented in Chapter 4 (El-Hassan & Fiadeiro 2006, El-Hassan, Fiadeiro *et al.* 2008). Despite the fact that I followed the steps of the aforementioned view-based technique of (Heckel, Engels *et al.* 1999), I was not targeting a systematic software development methodology, as they did, but rather modelling runtime software evolution (particularly the allowed reconfiguration space). Additionally, the target is not modelling a full system configuration, as presented in (Mètayer 1998, Wermelinger & Fiadeiro 2002) to query and preserve a global state, instead, it addresses a protocol-based configuration (i.e. subsystem). Last, but not the least, their reference model is highly abstracted and subjected to recursive multi-level integration, accordingly to their methodology, whereas the current approach maintains only two fixed views and their interconnections.

Technically speaking, modelling socio-technical protocols rely on adopting graph-based views to represent the intertwining between coordinated actions of the causal superposed contracts, on the one hand, and social laws that reflect "slack" control mechanisms such as norm-conferring, sanctions and rewards, on the other hand. It has been shown in (El-Hassan, Fiadeiro *et al.* 2008) how a combination between GT rules and meta models is capable of representing the semantics of nom-based reconfiguration operations within socio-technical protocols. Herein, the semantic differences between reconfigurations that manipulate technical components and those that associate with social ones are illustrated. I utilised the promoted role-based meta model to establish a reusable architectural style to support domain-independent and socially-driven reconfigurations that correspond to human-driven processes particularly when people deviate from their prescribed behaviour or required to react to sub-ideal contexts.

Despite the fact that most of the applications of GT tend to correlate graphical syntax and semantics, my approach takes a different path by providing the semantics of textual reconfigurations (i.e. social laws) without providing a corresponding visual language. This approach, instead, supports the textual reconfiguration language, by providing the required operational semantics of the embodied reconfiguration operations (i.e. *reconfigUnqualified()*) through the use of typed GT rules. Thus, developing a visual language over the proposed graph-based abstract syntax is trivial and does not contribute to the thesis objectives (i.e. formalising the structural and behavioural properties of socio-technical protocols). Many architectural styles have been defined following the steps of graph grammar based work of (Mètayer 1998) and (Hirsch, Inveradi *et al.* 1998). Security is another research direction that presents complex relationships that can be modelled with GT rules for reasoning about the consistency of these models (Koch, Mancini & Parisi-Persicce 2002) and specifications of policies (Koch & Parisi-Persicce 2002).

As the target is to model reconfiguration operations that manipulate a protocol state, an abstraction technique that has been presented and explained in Section 6.3.3. It is worth paying attention to the relationship between the normative layer and the coordination layer, as it cannot be perceived as a translation relationship, which can be modelled as a set of transformations between two models in the same level (i.e. horizontal transformation) as shown in (Akehurst 2000, Biermann, Ehrig *et al.* 2006). Also it cannot be model as a refinement relationship in the sense of (Baresi, Heckel *et al.* 2006), because the concepts of the source model (i.e. the social aspects) are even richer in terms of concepts than the target model (i.e. the technical configuration).

Alternatively, the translation relationship, which is strictly opposite to refinement but the social concepts cannot model the intended relationships as social concepts cannot be described sufficiently using the available 3Cs primitives. Therefore, another approach has to be adopted to maintain a third model that keeps the two models running together putting into consideration their interconnections and communications. In this perspective, views and viewpoints are advent to reach such a result and therefore the proposed approach adopts architectural views as a mean to model such relationships.

With regards to context modelling, context modelling is limited to the existence of *entry_operation* node, which demonstrates the connection between the environment and the system domain. It can model required behaviour if it appears without a runtime connection with a biddable entity, and if any then it models an enactment of this operation on system components, which, is also an environment event that is beyond the control of the configuration manager.

6.7 Examples & Tools Support

The scope of this chapter is to demonstrate the applicability of the graph-based approaches discussed in the previous chapter, which in turn aims at providing the semantics required for specifying both structural and behavioural properties of socio-technical protocols. This chapter takes into consideration the existing repertoire of GT tools and the specific exploitations of the GT approach that is adopted. Additionally, I shed some lights on the Gastroenterology unit example again to explicate lesson learned from dealing with such a medical case study.

6.7.1 Scope within the Methodology

With regards to the general methodological steps that have been suggested in Chapter 5 (Section 5.4.3), it is clear that tool support is required to assist software engineers to interpret the outputs of step 3 and 4 and animate these inputs graphically to create models. These models specify when and how interactions can be triggered and/or obliged beyond the coordination scope, in order to handle sub-ideal situations and provide runtime evolution. The result would be set of social laws that, once triggered, create normative positions to empower role players to enact the required/permitted interaction by means of reconfiguration facilitations or sanctions impositions.

6.7.2 Case study: The Gastroenterology Unit

In this section, proof-of-concept implementation of socio-technical configuration model is presented. The main concept to prove is the viability of the proposed subset of the extended 3Cs configuration language i.e. social laws and roles. The language constructs are represented in terms of graphical abstract syntax elements that are put forward to provide the operational semantics of the reconfiguration operations that manipulate models of socio-technical protocols to manage biddable interactions within organisational settings. Additionally, I address graphical modelling techniques that allow specifying abstract transformation rules that support generic concepts e.g. roles and task to secure the generality of the approach.

The selected example shows how the proposed approach, which is based on coordination and social laws, deems viable in the design, development and the evolution of a socio-technical model.

The example focuses on a specialised medical unit at a government hospital, Dubai, U.A.E., which provides treatment for digestive diseases. The unit consists of two Endoscopy suites containing each of which has four modern and fully equipped endoscopes with ancillary supporting facilities for patient’s reception, preparation and post-endoscopies recovery rooms.

An example (instance graph) of a socio-technical protocol is shown in Figure 6.9.

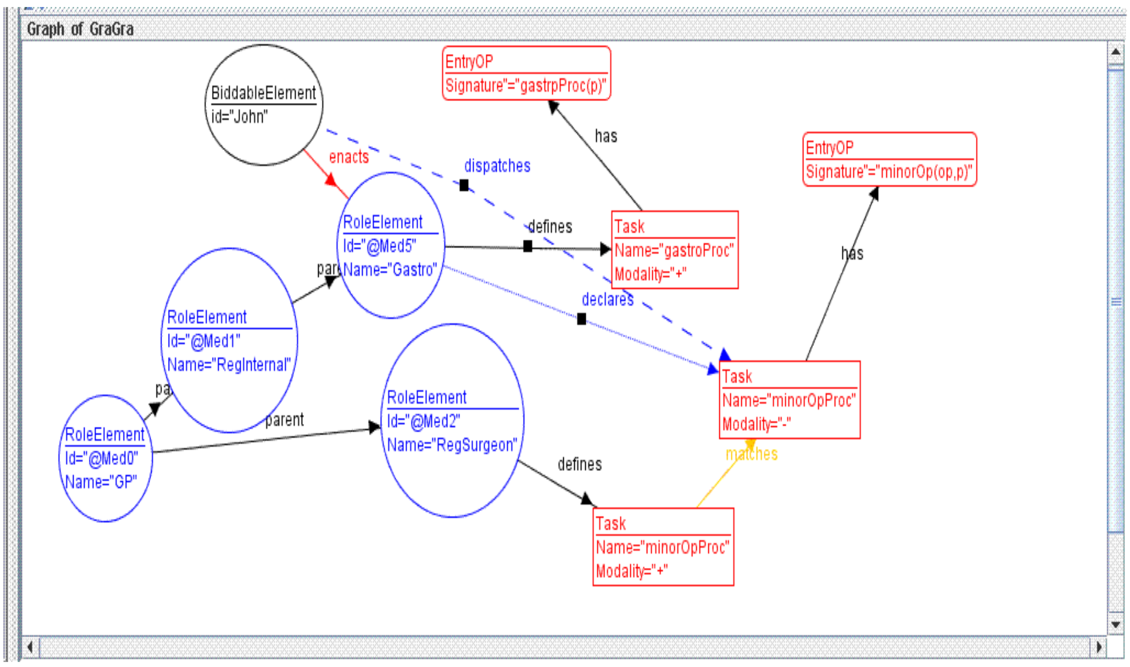


Figure 6.9 A socio-technical protocol (Configuration graph)

6.8 Tool Support

The theoretical work on Graph Transformation has been widely adopted in the category of model transformation approaches. These approaches are formally founded and allow exploiting visual models to represent different approaches of model transformations as explained in (Taentzer, Ehrig *et al.* 2005). I decided to model the graph transformation rules that represent system configurations and their evolution using the AGG tool.

6.8.1 AGG

Using graph transformation for specifying domain specific languages is becoming popular due to the fact the graphs and graph transformations demonstrate visually compelling yet mathematically rigorous models.

The AGG tool supports checking termination and consistency of a graph grammar based on graph constraints. More specifically, it implements the mechanism of critical pair analysis to check termination and confluence of graph grammars to manage inconsistencies during execution. Two graph productions may form a critical pair if they are in conflict, in the sense that they do not preserve the confluence property. This property is needed to guarantee that a rewriting system has a functional behaviour and give the same output graph starting from the same input and GT rules.

6.8.2 Graph Transformation Rules

In what follows I demonstrate a subset of the graph transformation rules that model the generic reconfiguration operation at a higher level of abstraction and the systematic approach to modelling flattened role hierarchies and the rules required to traverse them to reason about the permissions and the capabilities of enacting biddable entities.

6.8.2.1 Dispatching Eligible (DispatchTaskRole)

For simplicity, I present a simple transformation rule that demonstrates task replacement within the space of a certain role without the need to specify domain dependent concepts. This rule can be perceived as an abstract rule that provides the semantics of dispatching a task provided that the task is requested via a captured call to the task's entry operation and the biddable entity is playing a role that sufficiently acquired the sufficient permissions to run the corresponding task. The corresponding reconfiguration step is depicted in figure 7.2.

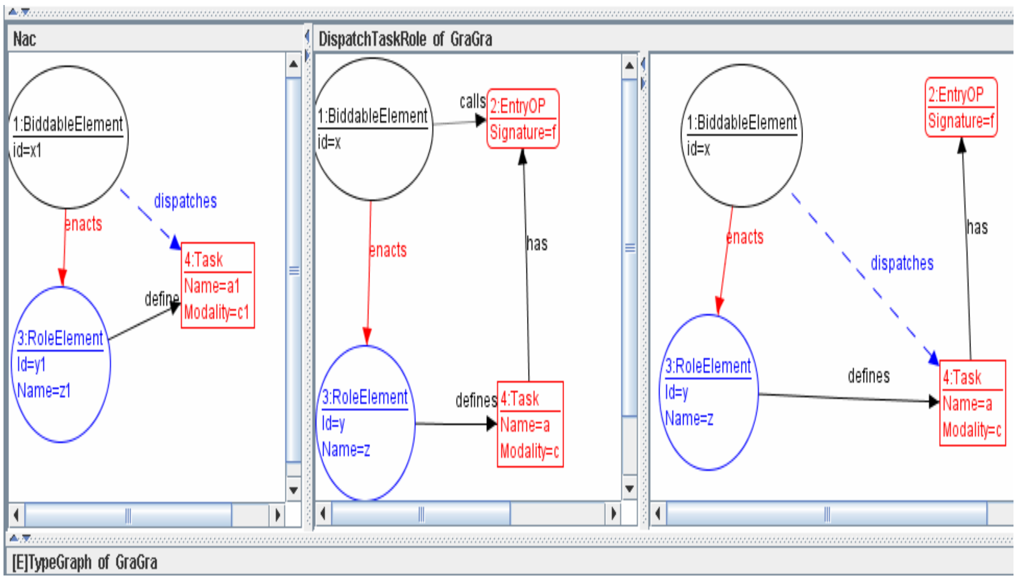


Figure 6.10 DispatchTaskToRole, graph transformation rule

6.8.2.2 Dispatching Eligible Tasks (direct inheritance)

Herein, a more complex task dispatching process through a transformation rule is illustrated to demonstrate how a task replacement can be executed via consulting a parent role node. In this case the current role does not have the permission to execute the task yet its direct parent does. This rule can be perceived as an abstract rule that provides the semantics of dispatching a task through a direct parent provided that:

(1) the parent role node has both the capability and the permission

(2) the task is a recognised as part of the capabilities of initial role.

The corresponding reconfiguration step will attach the biddable entity to the task but there will be no role transition. The transitive closure to search among direct ancestors is given through the recursive application of the rule illustrated in figure 7.3.

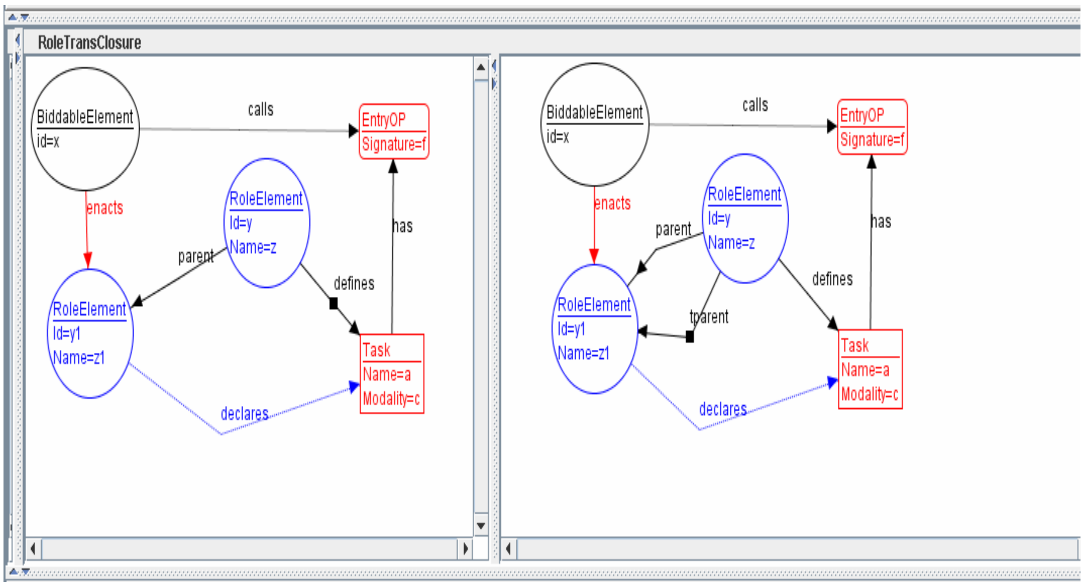


Figure 6.11 The transitive closure rule to compute the ancestors list

6.8.2.3 Dispatching Unqualified Tasks (ReconfigUnqual)

A higher degree of complexity is shown in this rule, which works together with the above rule to handle situations such as the one presented in the Gastroenterology Unit example, where the protocol needs to traverse the role hierarchy recursively to find the most *appropriate role,* with its configuration facilities, to be borrowed to the enacting biddable entities. The gastroenterology example required a *Gastroenterologist* to acquire a *registrar_surgeon's* permissions in a life-saving context.

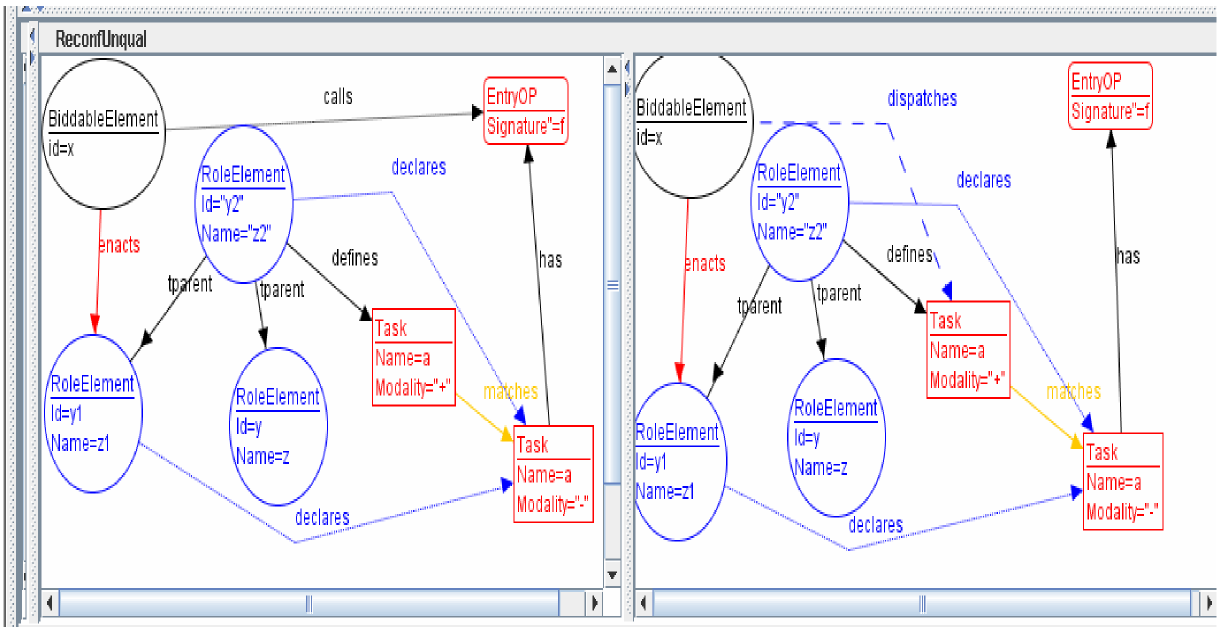


Figure 6.12 ReconfigUnqual role transition

6.8.2.4 Dispatching Via Normal Inheritance

This case is not demonstrated in separate graph because it is identical to the first rule. If a role inherits the capability of executing a task from a parent then it is shown explicitly be a *define* edge and thus it could be dispatched.

6.8.3 Remarks

The semantics provided by the above GT rules specifies the operational semantics of the reconfiguration operations. These rules demonstrate as well how inheritance hierarchies can be applied and traversed. The rules are abstract and generic in the sense they can be applied on any application domain that comprises organisational structures and human-driven processes. Hence, self-adaptation (i.e. reconfiguration) can be perceived as a mechanism that interleaves with the computation and connect both computation and coordination with process management. The proposed approach has taken a step forward towards introducing reconfiguration primitives to collaboration between people and software intensive system by capturing biddable interactions. However, more complex patterns of these interactions have to be considered for future research.

For example, the behavioural aspects of a particular rule pattern in social laws (i.e. a*ctive state and not operation*) needs further refinements. It models a system response to a monitored obligation violation; however, the syntax should explicitly define the "conditional" obligation apart from the social law that monitors its fulfilment. Such obligations should be modelled separately i.e. in a "separate social law". However, the two social laws have to unify "obligation" and "monitoring" laws. The links for such unification are the role player and/or the "violation context."

Chapter 7

Evaluation

"Example is not the main thing in influencing others, it's the only thing."
Albert Schweitzer

7.1 The Scope

This chapter describes several case studies used to evaluate the advantages and the disadvantages of the proposed approach. The objective of these case studies is to substantiate the contributions of the proposed approach and to evaluate its applicability to perform architectural self-adaptation as a mean to fit in biddable human interactions that are vital for addressing sub-ideal situations in a way that ensures the required joint behaviour to preserve the overall system stability.

This chapter is not an attempt to address stability as a quality that emerges from the required joint behaviour of the system components (i.e. social and technical components). Instead, it concentrates on how the self-adaptive approach takes the human biddability into account and on what impact the proposed modelling primitives have on addressing social interactions aspects. The evaluation reported in this chapter explicates strength and weaknesses of the proposed approach, the language and the methodological steps in realising the sort of dynamic adaptation that is required to handle unexpected human interactions within changing organisational context.

I argue that the purposeful self-adaptivity, once achieved, demonstrates the usefulness of the underlying modelling primitives and reconfiguration mechanisms. Thus, demonstrating the required properties of self-adaptivity such as being domain-independent, context-aware and capable of making adaptation decisions at runtime entails the generality of the approach and the expressiveness and the flexibility of the proposed architectural primitives.

Self-adaptivity is introduced as an integral part of the system specification (i.e. a glue of a social laws or connectors in the sense of (Garlan & Shaw 1993)). Self-adaptivity has been explained and differentiated from other self-* approaches such as self-organisation and self-healing in Chapter 4 (Section 4.8).

The proposed method, together with its primitives, is evaluated at three different levels:

(1) *The approach level*: generality and applicability
(2) *The language level*: the evaluation of flexibility and the expressive power
(3) *The implementation level*: the maintainability of the self-adaptivity mechanism

7.1.1 Concerns of Self-adaptivity

If a system is intended to incorporate dynamic application of adaptation, whether anticipated or not, it should include features to support the interface between its constituents and the environment. This interface specifies changes that can be captured and defines ways to respond to these changes through the adaptation logic. Keneey has promoted four concerns for anticipated changes: *when*, *where*, *what* and *how* (Keeney 2004). The approach presented in this thesis has prompted a new concern (i.e. *who*) which plays a vital role in enriching the *when* concern with new semantics (i.e. sub-ideal situations) that overlooks temporal aspects. Determining role players at runtime supports the runtime selections of *what* (i.e. participating components) and *how* (i.e. the reconfiguration mechanism) aspects of self-adaptivity. It should be emphasised that approach overlooks the temporal aspect of the *when* question as well as location aspects captured by *where*.

7.1.2 Runtime vs. Design-time Adaptation

Changes or adaptation in any system configuration (i.e. a population of valid system components) can be specified either at design time or runtime. Generally, changes determined at design time are supported by ECA-based reconfiguration rules that query the configuration state and perform the specified changes in a fully determined way. Conversely, runtime changes need the support of the execution environment (via the runtime infra-structure) to monitor and reflect the properties of the environment (as captured from sensors) and the system dynamic components (as captured by abstract models). Once the required dynamic data is captured and reified decisions about changes can flexibly be made (Di Marzo Serugendo, Fitzgerald et al. 2007, Anderson, De Lemos et al. 2008).

7.1.3 Anticipated vs. Unanticipated Causes of Change

Anticipated changes are driven by expected changes in the dynamic requirements of the target system, and thus, they can be prepared before hand, whereas unanticipated changes are driven by another category of dynamic requirements that include non-determinant properties that cannot be handled before runtime. In the socio-technical systems context, alleviating sub-ideal situations, which are requirements accompanied with non-determinant elements (i.e. biddable social interactions), can only be handled by performing runtime adaptive changes.

The degree to which a certain adaptation is anticipated is related to the prior knowledge about the set of concerns, which have been defined in Section 7.1.1, before performing the adaptation. If these questions (when, what and how) can be deterministically answered before performing the adaptation (i.e. the trigger and the execution mechanism), then this particular adaptation is completely anticipated (e.g. programmed reconfiguration operations in the 3Cs approach).

Unanticipated adaptations are more complicated than anticipated ones as these adaptations must be explained at both design-time and runtime levels. This thesis builds on the *who* concern, which supports figuring out the answers for the other four ones. Moreover, it draws the line between the anticipated and unanticipated adaptations. For example, if the role (i.e. *who*) for a joint interaction (i.e. performing a task) is underspecified then technical elements of the corresponding task can be kept undefined, or the system anticipates a number of alternatives configurations, from which one has to be selected. However, the selections can not be determined until the (*who*) (role-player) relationship is clarified at the runtime.

7.1.4 The Evaluation Criteria

Since self-adaptivity has been adopted as a means to address the biddability of social components, the degree of expressivity and flexibility to address and manage biddable social interactions is central to the discussions of the presented case studies. The discussions will be broken into the following sub-categories:

- The support for the identification of sub-ideal situations (at the process level) staring from crosscutting the configured human-driven organisational processes, role-based norms of the system and the contextual information (at the system physical environment ranges)
- The support for capturing unexpected human interactions
    - Permissions vs. capabilities
- The ability to self-adapt the system in a generalised way
- The flexibility of modelling primitives and the underlying architectural infra-structure
- The maintainability of the modelling primitives after deployment

7.2 Case Studies

7.2.1 Claim 1: Generality

The first case study is meant to demonstrate the generality of the approach. The term general-purpose can be understood in several ways. The term addresses generality with respect to the application domain, problem domain, the independence from any specific programming language or any runtime environment (Keeney 2004). Achieving these aspects will support having an agnostic approach to specific implementation of socio-technical systems. As such, it will be able to incorporate human components randomly to socio-technical protocol either by embowering them as active role players to enact tasks or by alerting and influencing them to obey norms as play the role that they have been assigned.

7.2.1.1 Motivations

This example is meant to make illustrate the generality of the proposed approach in terms of its way of addressing problem and applications domains. By making a departure from the medical domain presented in Chapter 4, the case study demonstrates that the approach does not rely on concepts that are limited to a certain application domain (i.e. the medical domain) or a certain software development environments. The following case study targets a completely different application domain (i.e. flight control systems) where pilots interact seamlessly with an intensive software system.

7.2.1.2 Design

Herein, I introduce an case study of socio-technical interactions within the aircraft flight deck as proposed by (Fields, Harrison & Wright 1997). The interactions between the pilot and the aircraft monitored control system are of a socio-technical nature as they are beyond the traditional and determined HCI interactions where the pilot collaborates with an intensive system that includes software to realise well-defined tasks. Mitigating risks or addressing sub-ideal situations by enacting their recovery tasks requires purposeful interactions from the pilot side in terms and the support of the control system, if the interaction is justified, by means of self-adapting the required task to the existing operation conditions.

In order to express the power of the proposed hierarchical role representation, I enriched the original case study by introducing the co-pilot as sub-role that has a minimal set of tasks yet s(he) is capable of enacting all tasks once they are successfully delegated to her. This alteration in the original case study can be justified as some aircraft accidents were ascribed to co-pilot errors e.g. Egypt Air 990 (NTSB 1999). If the monitoring system was able to distinguish the technical command issuer (i.e. the social component), and correlated him to his assigned role (i.e. the social role), it would have been possible to analyse the command with respect to the system role-based norms (i.e. social laws). With such correlations it would have been possible to suppress the action , impose sanctions on the command issuer or invoke a sub-ideal event that triggers another social law to oblige other social participants (i.e. the main pilot) to recover the situation by a enacting a reverse task to regain the stability of the aircraft.

The case study under focus specifies an aircraft that has two engines each of which has a couple of extinguishers that can be operated independently by the pilots. As described in the aircraft manual, the pilot normally enacts normal tasks during a flight yet he has to give the priority to emergency situation (i.e. sub-ideal situations).

The convergence from sub-ideal situations to a more stable one requires flexible norm-based governance and purposeful adaptation (e.g. obliging or permitting a recovery task), together with the pilot's obedience to the corresponding norm in such situations. The aircraft manual describes the instructions of recovery tasks to which pilots are enjoined to enact to achieve the stability of the system. These instructions are encoded in social laws, which are interpreted by the configuration manager (i.e. the harmoniser), to provide the means for facilitation and sanctions to affect the overall system behaviour.

Engine Fire
1. Reduce engine thrust to Idle 2. Wait 10 seconds 3. Fire shot one 4. If warning clears, shut down engine 5. If warning persist, fire shot two 6. Shut down engine

Table 7.1 Aircraft engine fire procedure – from (Fields, Harrison *et al.* 1997)

From the point of view of the 3Cs approach, the task in hand has to be broken down into fairly simple coordination interfaces. Two coordination interfaces should be created by the system modeller to avail the aircraft control services for shutting down the engine and actuating the fire system: pilot, engine_emergency_shutdown and fire_shot. Additionally, the modeller should devise the actor side from the pilot for calling these services, namely pilot_shutdown and pilot_fireshot.

The implementation of the 3Cs extension to the norm-based configuration entails specifying the recovery task *rightEngineFire*, which requires the detection of the sub-ideal situation via the *engineFireWarning* sensor. This sub-ideal situation is communicated to the pilots by means of an alarm or a flashing light. Achieving the intended joint-behaviour of the system requires the self-adaptivity mechanism to respond the pilot's omissions such as refraining from executing a vital recovery task (i.e. initiating the task). Thus, modelling the self-adaptive behaviour towards the detection of the pilot's ignorance of the obligation to perform such tasks when needed is very critical in the control part (i.e. sanctions) of the adaptation logic of social laws.

```
coordination interface pilot-enginefire
import types  engineType, pilotType, extinguisher
events reduceEngine(e:engineType, p: pilotType)
      wait()
      fireshot1(ex: extinguisher, e:engineType);
      fireshot2(ex: extinguisher, e:engineType);
end interface
coordination interface engine-services
import types  engineType, pilotType
services
      stopEngine()
      haltControl()
      isStopped()
end interface
coordination interface extinguisher-services
import types  engineType, pilotType
services
      fireShot1(e:engineType)
      fireShot2(e:engineType)
end interface
```

To take the pre-condition of the task constituents into consideration, coordination contracts can be used:

```
coordination law fireshot-pocredure
partners p:pilot-enginefire; e:engineType; ex:extinguish-service,
   m:monitor-engine
rules
      when p.fireshot2(e,p)
      with m.ensure_alarm(e);
      do ex.fireShot2(e)
end law
```

However, such a contract cannot help in preventing the omission of the pilot to the obligation of using the extinguisher when fire alarm is on. If a system monitor captures the fact that the alarm is still on and the pilot has omitted the fireshot1 action (because the fire seems minor in his opinion). Activating fireshot2 will be permissible to other pilots. This includes the co-pilot as well, who has less permissions than the first pilot but at such a situation he would be capable of firing the extinguisher shot. The PILOT type is an abstract role that includes both pilots and co-pilots.

```
social law fire-extinguishing
anchor role p:PILOT
type e:engineType, ex:extinguisher
partners
      a:administrator
      m:monitor-procedure
when p.fireshot2(ex,e,p) and a.omitted(fireshot1)
      if m.ensure_alaram(e)
      reconfiguration reconfCoord(p,fireshot2)
      sanction a.record(p,op,"unacceptable jump")
when unqualified p.fireshot2(ex,e,p)
      if m.ensure-alarm(e)and a.omitted(fireshot1)
      reconfiguration reconfUnqual(p,fireshot2)
      sanction a.record(p,ex,"unqualified")
```

7.2.1.3 Discussion

The case study demonstrates the ability of the proposed approach to address different categories of application domains that tackle different tasks. Moreover, this case study shows clearly that the problem that the approach addresses within socio-technical system is not domain specific (i.e. problem of empowering people to act out of their role scope or handling their failure in performing required tasks). The prerequisites for addressing a socio-technical domain are having well-defined tasks, role structure and contextual-awareness mechanisms. With regards to the independence from any programming language, the approach is built on top of the 3Cs framework whose supporting language is a textual specification language that is independent from any known programming language. Although the 3Cs framework was implemented initially in Java and targeted component-based systems running within an event-based systems, it was designed to be a language independent.

The approach relatively achieved a level of independence from particular adaptation. This can be argued as it separates *who* (i.e. the role player) and *when* (i.e.

whether ideal or sub-ideal) from *what* and *how* concerns. This separation makes the adaptation that answers the *who* question (i.e. role transition) purely generic).

7.2.2 Claim 2: Applicability & Flexibility

The first case study is meant to demonstrate the generality of the approach. Any application with an organisational role structure and well defined tasks can be equipped with the proposed self-adaptivity infra-structure without any intrusive preparations in the components of the target application. This is exactly what the following case-study is exhibiting. Putting into consideration the case study presented in Chapter 4, the approach still look to the problem in an abstract way; managing biddable interactions (i.e. when social components perform tasks that are part of their scope and how the system should self-adapt to address such behaviours, particularly at sub-ideal situations. Moreover, the methodological steps that were presented in Chapter 5 is applied to demonstrate the support for the awareness of sub-ideal situations (at the process level) staring from crosscutting the known boundaries of vital contextual information (at the system physical environment ranges) with the identified human-driven organisational processes.

I also demonstrate the flexibility of the reconfiguration language, which entails the capability of modelling and performing unanticipated changes as a response to the interactions of social components, which play well-defined roles in organisational settings. Since the player of the role to be named or how the role player's capabilities will be exploited, demonstrate some features of the unanticipated dynamic adaptation. The ability to incorporate human components randomly to the socio-technical protocol and enable them as adaptation drivers or participants is shown also in this case study.

7.2.2.1 Motivation

The main questions to be answered are whether the proposed approach is applicable to identify the nuances of human interactions within sub-ideal situation that occur in a socio-technical protocol. Additionally, the proposed modelling primitives should be examined to demonstrate whether they successfully and flexibly address the following issues or not:

- dynamically capturing sub-ideal situations and applying normative positions

- performing adaptation on arbitrary social components which have not been explicitly determined
- performing runtime adaptation of roles if necessary and binding these adaptation to select among technical reconfiguration options that are prepared at runtime
- accommodating the non-deterministic reactions of social component to normative positions:
  - by facilitating reconfiguration if the social component's reaction is as expected;
  - or imposing sanctions if the monitored social behaviour does not conform with the normative position

7.2.2.2 Design

The approach provides a mechanism through which a task is operated in an abnormal context to achieve the stability of the system at hand by exploiting the available nature and skills of identified participants (i.e. social components). In this section, I expand the main case study that was presented in Chapter 4 (Section 4.5.4.2) in the light of the methodological steps sketched in Chapter 5 (Section 5.4).

Analysing how people interact with the intensive software system in the Gastroenterology department involves several steps to identify technically oriented recovery tasks, roles, sub-ideal contexts and flexibility points. Further explanations of the case study, as follows, show clearly how the proposed methodological steps support extracting and putting into effect the desired joint behaviours in order to realise the system's stability requirements, in response to the detection of sub-ideal contexts that may occur unexpectedly in a collaboration. This sort of requirements has been identified earlier in (i.e. *soft-goals* or dynamic requirements (Yu & Mylopoulos 1997, van Lamsweerde, Darimont *et al.* 1998, Fontaine 2001)), Such stability requirements have to be addressed sometimes by enabling a social party, as response to detecting the sub-ideal situation, who voluntarily takes the initiative and performs a corresponding recovery task.

However, enabling the required recovery task requires both role and technical level reconfiguration to facilitate the hot-swapping of tasks. Needless to say, the key element of quality in this context is the expressiveness of the reconfiguration language. The more the language minimises the degree of explicit management necessary for constructing the required subsequent evolution whilst preserving the required properties, the more expressiveness is ascribed to it.

Applying the Methodological Steps on Gastroenterology Case Study

Herein, I explicate the methodological steps through which social roles and laws were derived from the documentation of the Gastroenterology department case study as presented in Chapter 5 (Section 5.4).

Step 1: Inputs for starting the step should be prepared, particularly the architecture of system processes. In this case study, the department's medical pathways, the endoscope manuals, job descriptions and written code of norms were prepared.

Step 2: Initially, I analysed human-driven processes (i.e. medical pathway) in the beginning as they are deemed to be more related to general concepts and easier to assess the system's sub-idealites and their stability-related objectives

Step 3: The targeted process is selected (i.e. the normal Gastroenterology process, called *Gastro* for short) and the key actor is identified (i.e. the team leader normally the Gastroenterologist). The process is a routiniesd process since it should be booked earlier and role players and equipment should prepared accordingly. Goals are also stated which includes assessing the interior surface of an organ of a patient but without endangering the patient life.

Step 4: Equipment, software and medical staff interfaces are prepared taking into account the three phases of the gastroenterology procedure: pre-endoscope a (i.e. pre_gastro), the endoscope operation (i.e. gastro_op) and post-endoscope (i.e. post_gastro). The modeller should demonstrate the required configurations for the entire process e.g. use case-like in the light of *required technical configurations* as if the process's technical resources and the actors are properly configured before hand to enact the entire process alternatives (c.f. use cases). As such the configuration is pertained to the gastroenterologist who should guide the process progress then it is human-driven, (i.e. not purely mechanistic). It has been noticed that the process key player (i.e. the gastroenterologist) does not participate in pre-gastro and post-gastro, therefore it is sufficient to present a concentrate on the gastro_op part of the process.

The diagram depicted in Figure 7.1 contains the actors and the components required to execute the process. Identifying the goal of preserving the patient's life incurs adding monitoring capabilities and specifying corresponding indicators.

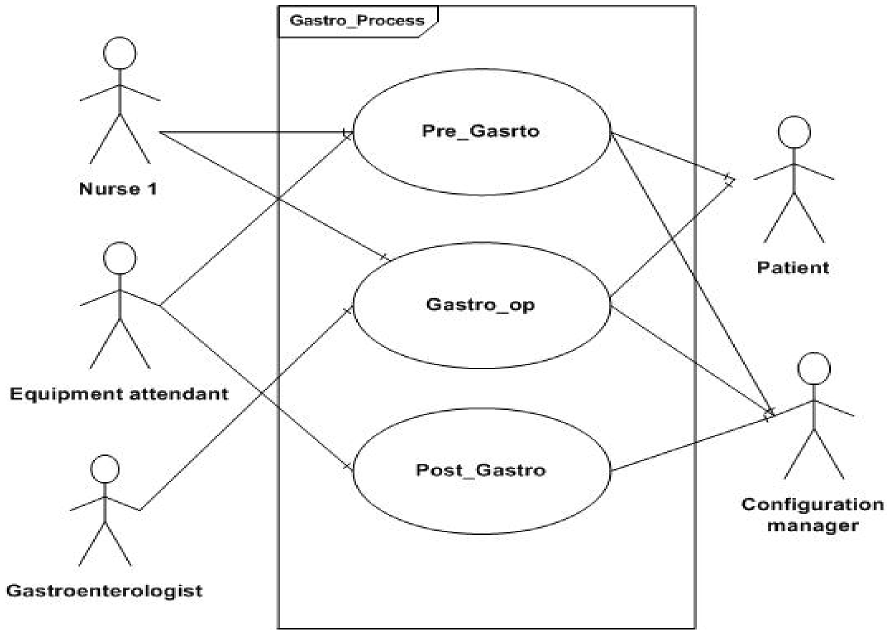


Figure 7.1 An initial use case-like Configuration diagram

Step 5: The global process-view (i.e. the use case-like static configuration) should be divided into purposeful sub-configurations (e.g. a configuration should satisfy a specific functional requirement or a user requirement, which may contain several technological components serving a number of actors. Figure 7.2 illustrates an example of a purposeful sub-configuration that allows accumulating the gastroenterologist interfaces to both the endoscope equipment and the monitoring service in the medical record system. It is purposeful in the sense of achieving a specified sub-goal: performing a Gastro operation while monitoring the patient's vital signs.

This step is divided into sub-operations according to the sequence mentioned in the methodology to produce interfaces, contracts and accumulations of the main actor's interfaces. The latter allows the putting into consideration social (role-based) and technical requirements with respect to needs of the process at hand, from the components or technical view.

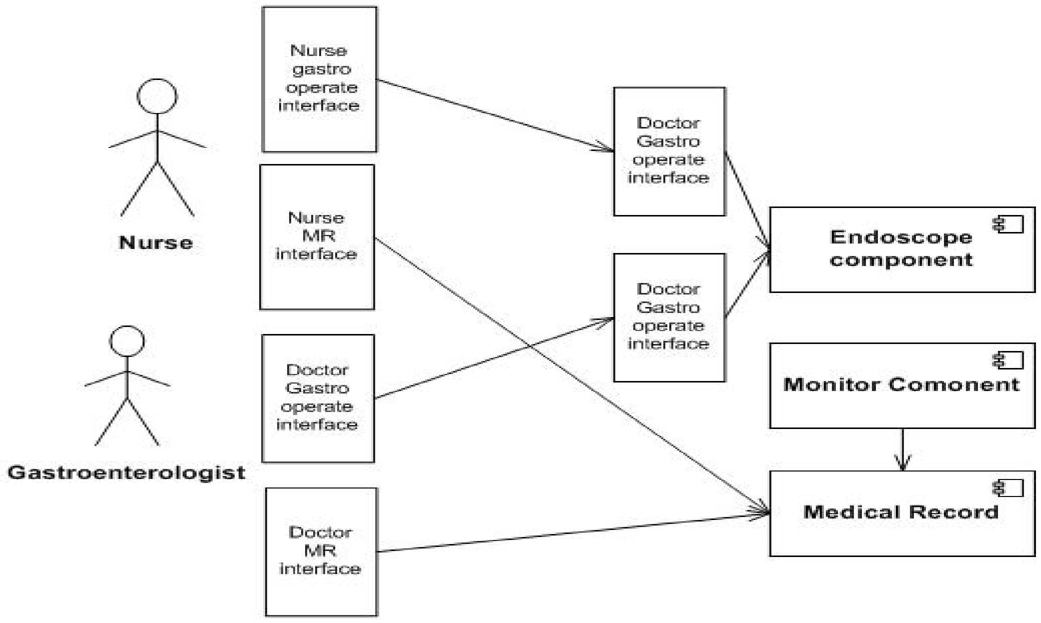

Nurse
Gastroenterologist
Nurse gastro operate interface
Nurse MR interface
Doctor Gastro operate interface
Doctor MR interface
Doctor Gastro operate interface
Doctor Gastro operate interface
Endoscope component
Monitor Comonent
Medical Record

Figure 7.2 A purposeful task within the gastro process (Gastro_op) which allows operating while monitoring vital contextual information

Gastroenterologist interface for Gastro-Procedure

```
coordination interface doctor-gastro_operate
import types  endoscope, doctorType, patient
events operateEndoscope(e:endoscope, p: patient)
     biobsyRequest(e:endoscope, p: patient)
     operateendscopr(e:endoscope, p: patient)
     shutEndoscope(e:endoscope, p: patient);
end interface

coordination interface doctor-consult-MR
import types  doctorType, MR, patient
events requestVitalsigns(p: patient)
     report_observations(p: patient)
end interface
```

Patient's interface for several procedures

```
coordination interface patient-operation
import types operationType, doctorType,patient-operation;
```

```
services
        give_consent(op:opeationType,p: patient-operation)
end interface
```

As specified earlier in Chapter 4, coordination interfaces exhibit a certain business use of a technical or software component (i.e. operation, services and constraints) that are available to be causally governed by coordination contracts that ensure the functional requirements of the system. If a social component is configured with a technical component and their interactions are coordinated with some contract, then one can consider that the social component is conditionally permitted to access this particular technical component in a way that respect the interface constraints and the pre-conditions of the contract.

A Coordination Law: Gastro-procedure

```
Coordination law gastro-operation
partners d:doctor-gastro_operate; e:endoscope-operate p:patient-
operation, a:allocator-do_operation
rules
     when d.operateEndscope(e,p)
     with a.ensure_consent(GASTRO,d,p);
     do e.endoscopeRun(e,p)
end law
```

The recurrence of these steps while traversing each of the process's alternatives, results in a set of interfaces related to the leading actor. These interfaces can be combined gradually in the form of task constructs provided that the accumulated tasks refer to shared technical resources and to a unified unit of work. Also contract specify certain behavioural properties related to the actor under focus.

Step 6: After identifying tasks, they can be matched with the organisation chart to allocate these tasks to appropriate roles. The Gastro process for case study revealed the particularities of the consultant Gastroenterologists who is the natural leader of such a process. The consultant Gastroenterologists role should be connected with other roles representing other medical staff members who share parts of his capabilities or exceed them. These roles have already been explained in Chapter 4.

Step 7: For example, the identified interfaces that are configured to allow the gastroenterologist to operate the endoscope, query and update the medical record can be combined together to fill in the Gastroenterology role. However, we have to distinguish between the two interfaces in terms of the role capability. Querying and reporting to certain information about the patient is a mission that any doctor can perform therefore it is normally placed in the doctor or GP role. Thus, this interface can be accumulated to any task that can be performed by doctors. Conversely, if the doctor-gastro-operate interface is accumulated to a task definition, then the achieved task cannot be performed unless the doctor has already been trained as a gastroenterologist, therefore the interface is accumulated only to tasks that are placed in the gastroenterologist's role.

Step 8: Next, the system analyst should identify inconsistencies related to soft-goals i.e. dynamic goals that might surface as an obstacle to the system stability. As the most valuable asset to maintain is the patient's life. It would be normal to correlate to the patient's vital signs readings with the contextual information that influence the triggering and the management of conditional norms of the system.

During the procedure the team leader (i.e. the consultant Gastroenterologist) delegates the mission of monitoring the vital signs either to an automated software agent or to nurse staff as part of the Gastro process. In both cases if the O2 sign hits the danger level (i.e. 20% decreased oxygen level), the self-adaptivity mechanism (i.e. the harmoniser) interprets the corresponding social law and proactively proceeds to put into effect an obligation imposition and alert the role player to communicate the sub-ideal state. The sub-ideal state's physical information is captured by the framework's sensors and then handled by the harmoniser. The context of the collaboration between participants might be diverted to a life-saving context. Such a hot-swap (as defined in Chapter 5) was discovered while analysing the routinised human-driven Gastro-operation process.

Step 9: Now the system analyst should concentrate on a purposeful configuration step that once configured properly may allow managing the discovered threat to the system stability. Such a purposeful configuration shall include the Gastroenterologist component, the patient's record, the patient-monitor and the monitor/allocator of technical resources in the operation room.

The system shall permit the Gastroenterologist to violate the first norm ("no operation can be undertaken without the patient's permission") to clear the airway of the patient by performing tracheotomy operation. This is an example of a coordination obstacle to realise a purposeful (yet not correct) behaviour.

This behaviour can be achieved by assigning a new role instance to the Gastroenterologists, which embodies a permission to exercise power. This event "fatal decrease in oxygen" *counts as* an emergency context, which turns on some norms and switches off others. Here the system should leave to the Gastroenterologists' judgment whether to exercise this power or call and wait for a surgeon to come. However, the doctor will be responsible for whatever decision he takes.

Therefore, the approach emphasises addressing and analysing the system's code of norms as they may lead to discover inconsistencies in terms of obstacles to achieving purposeful behaviours (i.e. goal-oriented tasks) at sub-ideal situations. In this context, the following norms have been captured from the unit's specifications of medical procedures:

(1) No operation can be undertaken without the patient's permission

(2) Surgical intervention should be carried by *surgeons*[10] only

(3) In the case of emergency, a doctor may commit simple surgical Interventions if surgeons are not available and in a life-threatening situation

Obstacles can be interpreted as follows:

A role obstacle:-

detect_alarm(oxemeter, viatal signs_dangerous) $\rightarrow$ O(do_operation(op, surgeon, patient)).

A coordination obstacle:-

$\neg$ done(ensure_consent(op,patient)) $\rightarrow$ O($\neg$do_operation(op, surgeon, patient))

If the violation is captured and the context is defined (i.e. either ideal or sub-ideal) the sanctions of this violation should be appropriate to the defined context. In the ideal case (no dangerous vital signs): the request for operation is blocked, reports to the supervisor and management will be sent. On the other hand, the modelled sub-ideal case will address the violation and respond in a permissive manner by facilitating extra role instantiations and bid the leader to report on the consequences. Incorporating the tracheotomy procedure, as a hot swap, parallelises the current procedure or terminates it and a new configuration is realised as new roles, equipment and software pieces are instantiated.

The decisive points of specifying interactions in a socio-technical protocol are when the context changes to a sub-ideal state. Such a transition of context has been conveyed to the participating components. In my opinion this can be realised in either of the following ways:

1. Using Passive messages that contain only data and carry no control information (e.g., imply no method invocation). Not implying the exchange

[10] Readers should refer to Figure 4.6, Figure 4.7 and Table 4.1 for roles, tasks and permissions

of any control information makes passive messages more abstract and more flexible than active messages. This approach is adopted by Reo framework (Arbab 2004).

2. Alternatively, a modeller should identify and exploit events (i.e. speech act like in the sense of (Searle 2002)) that have institutional/organisational semantics, which switch the context of the collaborative behaviour.

For example the *oxemeter's* message: (vital signs_dangerous) sent by the monitor brings about a new institutional fact that maps a new obligation toward the surgeon and/or doctor role. A speech act with a declarative point (i.e. alerting a role player) brings about a new state of affairs; in the example above when the *oxemeter* performs a speech act to be interpreted by the reconfiguration manager to consider a sun-ideal state particularly if no response is detected or the absence of the optimal role player is identified.

One of the decisive steps in the methodology is where to locate the least role whose player might be rightly and safely yet exceptionally and temporarily permitted to execute the required interaction as a result of the triggering of a self-adaptation at both role and technical levels. The more abstract the role type within the role hierarchy the more descendants (i.e. role players) will be eligible for participating in such an adaptation. The required interfaces and contracts will be specified for the least role unless there is a need for specialising the task particularities if the task is initiated by a certain descendent of the appropriate role.

Doctor/Surgeon's interface for Simple Surgeries

```
coordination interface doctor-doOperation
import types  operationType, doctorType, patient-operation
services       ensure_consent(op:operationType,
               d:doctorType, patient):Boolean
events
               do_operation(op:opType, p: patient-operation)
end interface
```

Allocator's interface for Simple Surgeries

```
coordination interface allocator-do-operation
import types operationType, doctorType, patient-operation;
events ensure_consent(op:operationType,d:doctorType,
        p:patient-operation):Boolean
       stop (operation:OperationType);
        record(event:eventType, LOG);
        detect(event:eventType);
end interface

coordination interface monitor-procedure
import types doctorType, patient-operation, SignType, event-id;
events      detect_alaram( s:SignType, p:patient-operation)
             report( s:signType,p:patient)
end interface

coordination law emergency-operation
partners d:doctor-surgeon;  p:patient-operation Monitor-Procedure,
         allocator-do-operation; Monitor-Procedure
rules
when       detect_alaram( s:SignType, p:patient-operation)
with    a.ensure_consent(TRAECH,d,,p);
do      d.do_operation(TRAECH, d:doctorType, p: patient-operation)
end law
```

7.2.2.3 Discussion

The applicability of the adopted approach has been demonstrated through going into the details of the methodological steps presented in Chapter 5. The case study illustrated the advantages of addressing sub-idealites through making the difference between routinised transactional view of processes and the ad hoc human-driven view of the same processes. Moreover, the usefulness of the introduced task-based role structure has been clearly explained. Constraints on the applicability of the approach is also exhibited as the existence of a form of a process architecture together with written code of norms is a pre-requisite for using these steps. Last not least, the approach relies heavily on the 3Cs configuration and assumes their existence before hand.

The advocated level of flexibility in the proposed self-adaptive approach matches the sort of flexibility defined and characterised by (Schnenberg, Mans *et al.* 2008)., namely *flexibility by underspecification*: which offer design time configuration options that can be dynamically selected at runtime, among other types of flexibility as shown in Figure 7.3.

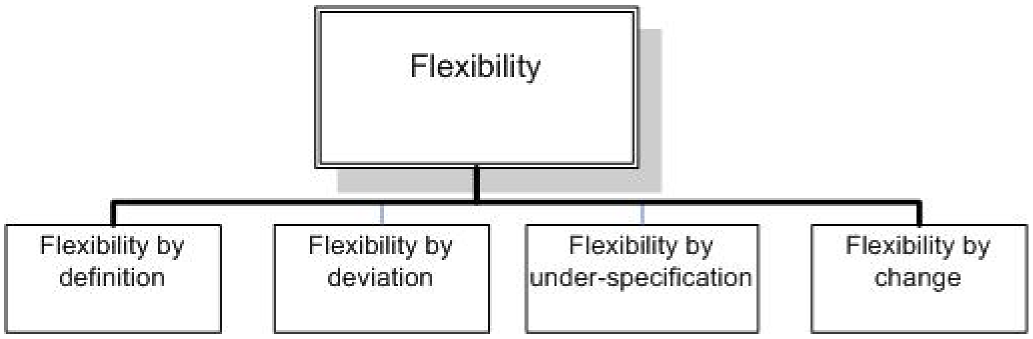


Figure 7.3 Taxonomy of flexibility (Schnenberg, Mans *et al.* 2008)

According to their definition, underspecifying some elements of the social law (i.e. who enacts a task) allows combining design-time and runtime configuration options. At design-time, options for managing the identified sub-ideal situations e.g. what technical configuration are needed, and how they will be configured, are fully determined. However, when combining social laws with runtime instances of the role hierarchy and the role profile of available social components, the selection between these options relies heavily on the actual capabilities and permissions of these components (i.e. role players) whose properties will not be determined until the late binding (i.e. task enactment) is realised. Thus, runtime configuration options (i.e. role transitions) are required to enable the selection between the reconfiguration options (i.e. technical reconfigurations). Additionally, when the social law proactively detects the physical elements of a sub-ideal situation, it voluntarily puts into effect the corresponding normative position, provides the required configuration and communicates the imposition of the obligation of the corresponding recovery task (e.g. through alerting participating humans). The social component's behaviour towards this configuration is undetermined (i.e. either obeying the normative position or not) and can only be captured and dealt with at runtime either by the selected reconfiguration (as mentioned above) or by sanctions.

7.2.3 Claim 3: Maintainability

Maintainability can be refined to modifiability and extensibility (Losavio, Chirinos & Pérez 2002). Extensibility allows designers to alter certain part of the language to their domain specific requirements without affecting other aspects of the modelling language's Meta model. This section builds on the previous case-study and demonstrates and evaluates modifiability and extensibility features of the proposed modelling primitives.

Nevertheless, modifiability plays the key role in boosting the maintainability property since modifiable modelling primitives yield maintainable socio-technical protocols from the organisational structures, norms and tasks perspectives.

7.2.3.1 Motivations

The separation of control and the shallow dependencies between the role view and the component view of the socio-technical protocol architecture allows modifiability at different levels of abstraction. For example existing roles can be added or removed easily to a participant's profile without much burden on the architect or system specifier. Additionally, this case study demonstrates how roles can be added to the hierarchy and how the self-adaptive mechanism treats an interaction that is subject to more than one social law

7.2.3.2 Design

The proposed architectural framework has exploited the role hierarchy and its underlying relationships with social laws to reduce the amount of unpredictability with regards to human interactions within organisational contexts. Figure 7.4 demonstrates the role hierarchy template that is used by social laws to determine at runtime the role space associated to a role player enacting a certain task. During the modelling process of a social law, the system designer is required to determine the least role and the first appropriate role in the hierarchy as the signified entry operation will be anchored on the first and the role configuration mechanism will weaken the binding permissions of the later to allow the role player to enact the task.

During the continuous maintenance of the system, some more roles can be added to the hierarchy (i.e. least role descendants or the first-appropriate role descendants) without any disturbances in the provided services or burdens of rewriting specifications. However, the anchored task can be redefined at any descendant of the first appropriate role to satisfy new requirements.

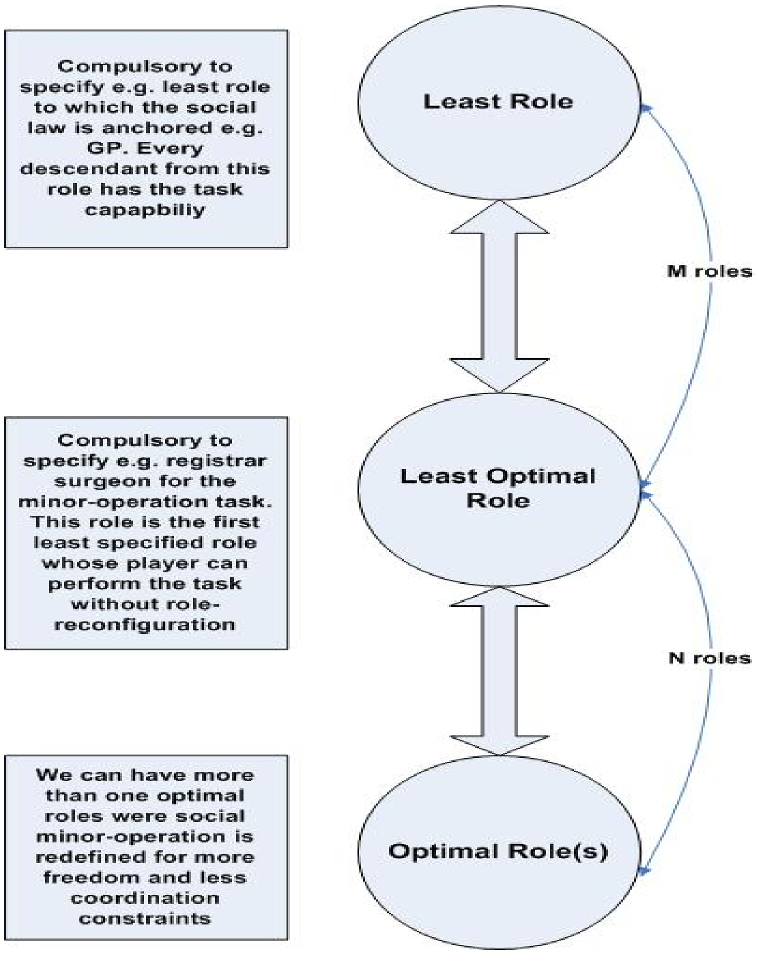

Compulsory to specify e.g. least role to which the social law is anchored e.g. GP. Every descendant from this role has the task capapbiliy
Least Role
M roles
Compulsory to specify e.g. registrar surgeon for the minor-operation task. This role is the first least specified role whose player can perform the task without role-reconfiguration
Least Optimal Role
N roles
We can have more than one optimal roles were social minor-operation is redefined for more freedom and less coordination constraints
Optimal Role(s)

Figure 7.4 The role space with respect of a social law

For example, a task redefinition may weaken some conditions with regard to required resources for a successful task enactment, by the role at hand, making it *optimal* for more specialised emergency needs but at the expense of the eligible number of players who can exploit the new social law that cascades the new task re-definition. In other words, if the role hierarchy is populated with M roles between the least role and least optimal role and N roles between the least optimal role and the optimal role, then the specified social role that cascades the defined task in the least optimal role serves all the player-role enactment in the space of (M+N)-1.

Therefore, the reusability of the proposed architectural approach relies heavily on the height and the density of the capability-based role hierarchies.

With regards to more complex situations, the approach falls short in resolving conflicting social laws when more than one required behaviour are triggered for enablement due to a change in the system context from ideal to multi-level sub-ideal situation. Therefore, only one task becomes enabled. Conversely, when a human participant internalises (i.e. trigger) more than one social law (as they pose the same interaction label, i.e. entry operation), only the social law with the more specialised anchor role is activated. For example, consulting Figure 7.5 shows that, if a *Gastroenterologist* triggers two social laws by enacting their labelled action and the first is anchored on *GP* role and the second is anchored on the *Registrar-internal* one, only the later will be activated. As the approach is based on an assumption that restricts having two different tasks with the same action label, the above example considers only the overlapping redefined tasks over the role hierarchy.

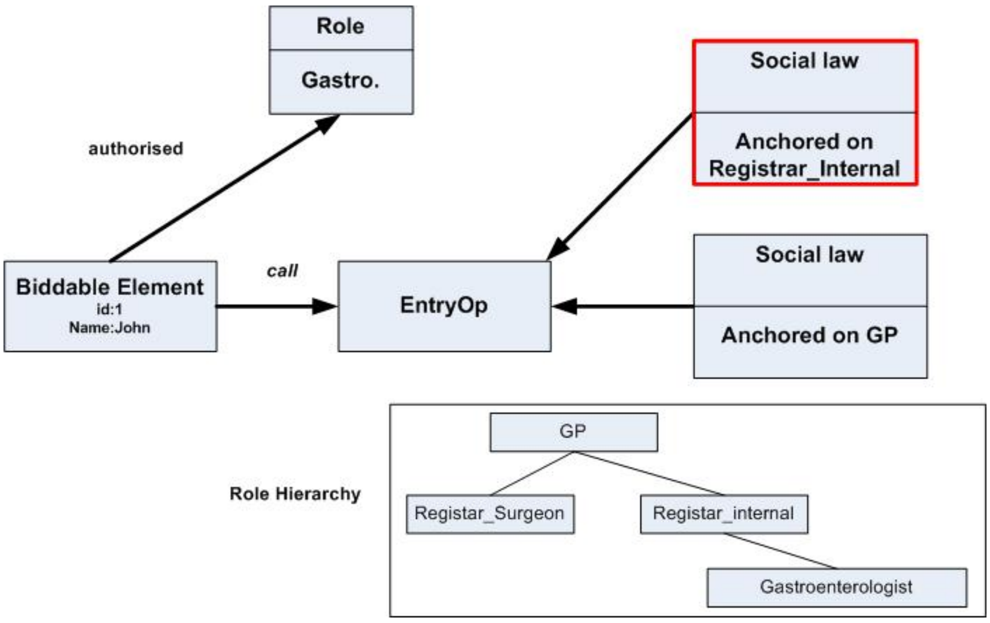


Figure 7.5 Role-based conflict resolution

7.2.3.3 Discussion

This above mentioned examples provided the evidence of the post-deployment features of the proposed modelling primitives. Roles can be assigned to the participant's profiles with out any side-effects on the role structure or social laws. These abilities demonstrate the modifiability of the approach's primitives. An example of the extensibility features is the hierarchical role structure that the approach proposes. Role can be created and included within existing role hierarchies. Tasks also can be extended and added to existing roles without any overheads.

7.3 Comparisons with Other Self-adaptive Systems

Comparing to other self-adaptive technologies with regards to generality, flexibility and maintainability, the proposed approach shows interesting and distinctive features. I present two well-know self-adaptive framework, Aura project (Garlan, Siewiorek *et al.* 2002) and ROAD framework (Colman & Han 2007), in addition to highlighting the adaptivity capabilities of the original 3Cs approach to exhibit the newly introduced features. Aura project presented by Garlan et al. (Garlan, Siewiorek *et al.* 2002) define a framework that put forwards the task concept as a first-class citizen and correspond each task to an identified goal. Their "task-aware" approach presents tasks as: " set of services, together with a set of quality attribute preferences expressed as multi-dimensional utility function, possibly conditioned by context conditions" (Garlan, Siewiorek *et al.* 2002, Colman & Han 2007). However, their task models are meant to capture user goals and intents, which are abstracted away in out approach. Moreover, the approach heavily relies on utility functions to evaluate and find the optimal balance between conflicting goals to suit user needs.

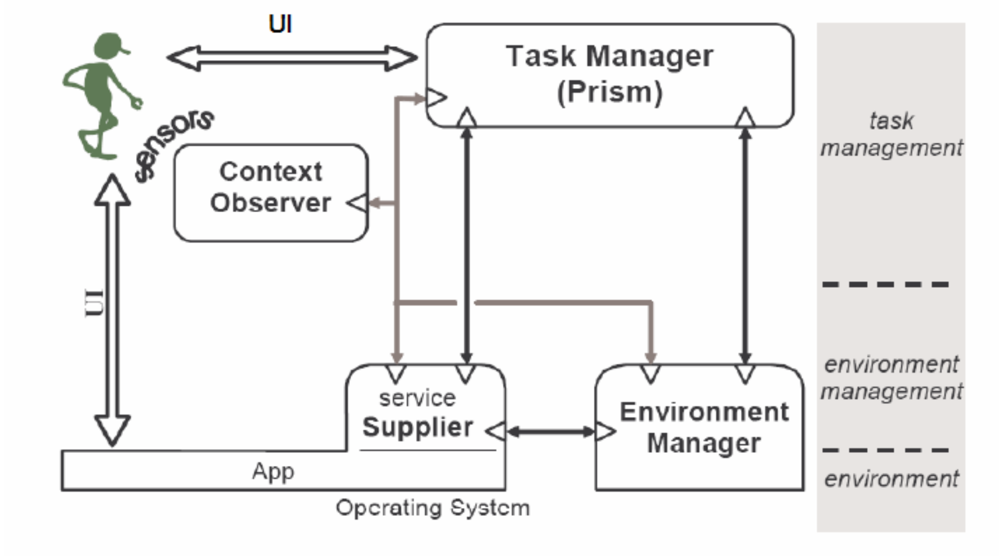


Figure 7.6 The Aura Framework (Garlan, Siewiorek *et al.* 2002)

One of the features of Aura that might bring it closer to our approach is its separation between the Task Management (TM), which determines the user requirements in a specific context putting into consideration his preferences, plans and context dependencies, and Environment Management (EM) which determines how to configure the environment in order to facilitate the user's needs. A general view of the Aura's framework is presented in Figure 7.6.

However, sub-ideality is not central to this approach and utility functions bury many important aspects that the proposed approach emphasises to be externalised. The TM and the EM participate in a control-oriented adaptation pattern in which control loops are required as a means for adapting both the internal systems and its environment. However, users are still treated as external entities to the system, thus flexibilities that the proposed approach introduced are missing in the Aura framework.

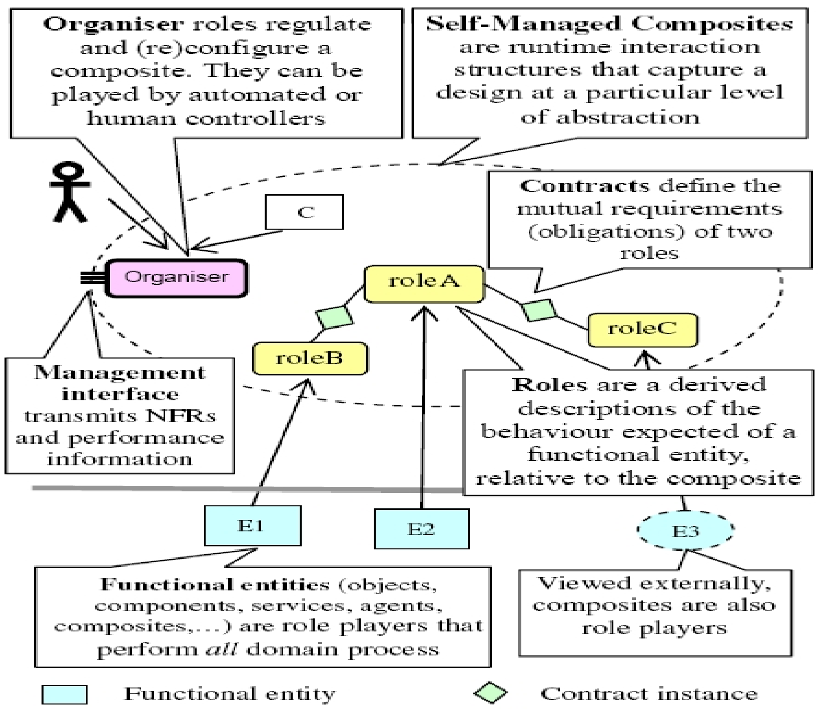

Organiser roles regulate and (re)configure a composite. They can be played by automated or human controllers
Self-Managed Composites are runtime interaction structures that capture a design at a particular level of abstraction
C
Contracts define the mutual requirements (obligations) of two roles
Organiser
roleA
roleC
roleB
Management interface transmits NFRs and performance information
Roles are a derived descriptions of the behaviour expected of a functional entity, relative to the composite
E1
E2
E3
Functional entities (objects, components, services, agents, composites,...) are role players that perform all domain process
Viewed externally, composites are also role players
Functional entity
Contract instance

Figure 7.7 The ROAD Framework (Colman & Han 2007)

The ROAD framework has been introduced earlier in Chapter 4, as it was very useful for the development of the concepts of thesis. Figure 5.7 shows the main components of the ROAD framework.

The following table explains in details the advantages and the disadvantages of the proposed approach comparing with the above mentioned approaches.

	Aura	**ROAD**	**3Cs**	**The 3Cs Extension**
Concepts				
Modelling social participants as integral part of the system	X	X	X	√
Roles as explicit entities	X	√	~ (partners)	√
Abstract roles/role hierarchies	X	X	X	√
Structural compositions at runtime	X	√	√	√
Recursive compositions	X		X	X
Normative concepts	~ (through non-functional requirements)	~ (through non-functional requirements)	X	√
Reconfiguration				
Possible at runtime?	√	√	√	√
Separating human control from environment control	√	√	X	√
Element of control loop adaptivity	√	X	~	√
Externalised management of conflicting rules	X (utility function)	X (utility function)	X	√
Formal support	√	~	~	~
Anticipated adaptation				
Structurally-driven adaptations	X	~	√	√
Temporal aspects	√	√	~ event-based	~ event-based
Unanticipated adaptation				
Non-determinism of executed adaptations	~	X	X	√
Achieving functional requirements	√	√	√	√
Achieving functional requirements	√	√	~ means to encode it	~ means to encode it
Tools				
Runtime environment	√	√	~ (java-based coordination development environment)	X

Table 7.2 A comparison between the proposed approach and its counterparts

The comparison shows that the proposed extension to the 3Cs approach provides means for flexible self-adaptivity, externalised conflict resolution and shows elements of managing unanticipated changes in the environment. Abstract role hierarchies secures the generality of the approach and when instantiated by social components specify the role space that can be evaluated and traversed through role transitions at runtime to support reasoning about an expected interactions of the corresponding social component. Modelling normative positions is the key feature of the approach as it supports the explicit management of sub-ideal situations and accommodates the non-determinism that characterises the human participant's response to it by means of preparing facilitations as well as sanctions.

7.4 Related Work

A unifying view of the engineering of self-adaptive and self-organising systems has been presented by (Di Marzo Serugendo, Fitzgerald et al. 2007) supported by generic framework that integrates both design-time and runtime features, meta data and runtime infra-structures, respectively. They proposed elements of decentralisation and self-organisations to the system components reconfiguration as a vehicle for achieving more freedom in the self-adaptation mechanism. More recent studies in this field emerged in (Cheng, de Lemos et al. 2008a) among which Anderson et al. have proposed a classification of modelling dimensions, with regards to self-adaptive system, that take into consideration the following properties: goals, cause of change, mechanisms and effect.

Addressing self-adaptivity at the requirement level entails complex frameworks particularly when some element of uncertainty is incorporated (i.e. the conformance of social entities to the expected behaviour)(Cheng, de Lemos et al. 2008b). There are several reasons for that including the need to keep an explicit representation of requirements to monitor them at runtime, to correspond them to the environment changes, in order to solve conflicting requirements and process them towards the lower-level architectural infra-structure to guide changes.

The nature of the context change that causes self-adaptation is another distinctive feature for categorising self-adaptive systems. Anticipated changes are easier to capture and handle and the dealing with unanticipated ones (i.e. those not foreseen in the design-time)is still a research challenge, as they cannot be openly and freely handled but rather should be anticipated at some point of time during the system execution (Keeney 2004).

7.5 Concluding Remarks

The examined case-studies demonstrated the generality, the applicability, the flexibility and the maintainability of the proposed approach. These case-studies showed clearly how the new level of configurability (i.e. self-adaptivity) has widen the range of possible adaptation that was present in the 3Cs approach in order to accommodate social interactions and ensure the joint-behaviour of socio-technical protocols in a way that preserves the system stability.

The first case-study exhibits the ability of the proposed approach to address different categories of application domains that tackle different tasks and shows clearly that approach is not domain specific (i.e. problem of empowering people to act out of their role scope or handling their failure in performing required tasks).

The second case-study demonstrates the power of using roles, task and operations as meta data and social laws as connector-like constructs to guide the adaptation of socio-technical models. The separation of control between the technical and the social views allows a participant (i.e. social component) to have double representations in the system. A human component can be sufficiently represented by their static actor-like representation (i.e. coordination interface) when it behaves expectedly. Once the human component is associated to one of the social roles then it will be amenable to its corresponding social laws and thus influenced by obligations and can be empowered to violate norms.

The sort of self-adaptivity that the approach poses shows elements of unanticipated adaptation that can be managed at runtime. It is impossible to anticipate which human component will be adapted at runtime; therefore, it would be almost impossible to determine which role it will acquire. It would be infeasible to attach each human component to its whole set of tasks related to the current role it plays. Instead, as Chapter 5 explained, if a human component is attached to certain role (i.e. it is part of his capabilities), the entry operations of the tasks of this roles are available for enactment yet they are subject to the control of social laws. When a task is permitted for enactment by the social laws then its internal interactions can be governed by the coordination mechanism (i.e. coordination contracts).

Moreover, this research is distinguished by its ability to characterise violations that are not only unavoidable but sometimes necessary and useful, where the agent (role player) enacts a task in order to fulfil a required mission, (not a literal instruction of the task/role). I should state clearly that one cannot claim that the social and collaborative order will be created and maintained mainly by explicit and formal norms, supported by centralised control, formal monitoring and reporting. However, the promoted architectural primitives together with the reconfiguration language and the methodological steps are able to fill the gap caused by the absence of applicable engineering methodologies that address the biddability of social interactions within organisational settings.

Chapter 8

Concluding Remarks and Directions for Future Research

"All the world's a stage
And all the men and women are merely players.
They have their exits and entrances,
And one man in his time plays many parts."
Shakespeare, "As You Like It (II, vii, 139-142)"

8.1 Summary

In this thesis, new concepts and a method have been developed for modelling biddable human interactions within the realm of socio-technical systems engineering. This method borrows its fundamental techniques and primitives from the literature of software architecture, coordination languages and organisational studies.

The examples that were presented in this research pave the way to exploring patterns of norm-based reconfiguration modelling, thus tackling the essence of the "social dimension" within socio-technical (sub)systems. They capture collaborative interactions between social participants, who carry processes in technology-rich environments, and technological components within well-defined organisational settings. However, it can be argued that those examples demonstrate the management over interactions that remain out of reach for the technical boundaries of most of up-to-date medical applications. As such, I refrained from adopting notorious Medical Record (MR) oriented examples that usually focus on information retrieval and related security and privacy issues (e.g. (Crook, Ince et al. 2003)).

This thesis emphasises that the presented medical examples, unlike the examples of security-based approaches to MR, handle collaborative interactions in workplace where the reasoning about unanticipated human-machine interactions is required to guide purposeful self-adaptations in the face of context changeability and/or human participants' deviations from prescribed routines. Moreover, social interactions are becoming viable for capturing and managing due to the rapid advances in the technologies of sensing and monitoring. Medical systems for instance, necessitate more complex intensive systems that include networks of sensors and actuators. The da Vinci surgical system[11] presents an example of the new trend of theatre systems that bring to the fore the issues that have been discussed in this research, (see Figure 8.1).

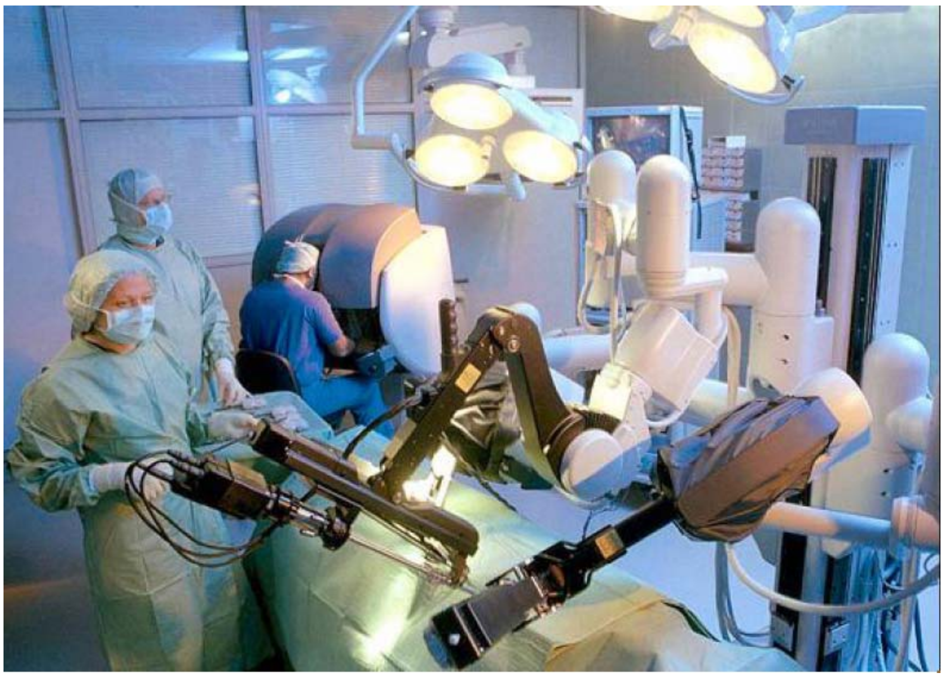


Figure 8.1 The da Vinci surgical system

[11] (Intuitive 2008). Intuitive, Intuitive Surgical, Inc., da Vinci Surgical System, 2008, http://www.intuitivesurgical.com/index.aspx, accessed in 01-05-2008

This research aims to explore the various concepts underlying social laws and roles. It also proposes various means in which it can be related to organisational theory and access control policies, to manage biddable human interactions within socio-technical settings. Notwithstanding the simplicity of the case study examples presented in this thesis, they were successful in presenting crucial properties that are amenable to be generalised and reused in various domains.

Delineated within this thesis, is the normative extension to 3Cs framework that recovers from situations in which collaborative commitments between socio-technical elements have to be reflected by means of new configuration settings, in order to enable or permit the effects of biddable socio-technical interactions. What this research has provided is an architectural approach to the development of socio-technical protocols that exhibit control over biddable interactions through the designated primitives. However, one cannot claim that this approach maintain a complete social order as it cannot be absolutely verified by providing explicit formal norms, centralised control mechanisms and monitoring agents.

8.2 Analysis of the Contributions

The architectural support to socio-technical systems is discussed in the light of the general theoretical areas of focus mentioned in Chapter 2, Three and Four, following the criteria that was presented by (Crook, Ince et al. 2003, Crook, Ince et al. 2005) on evaluating access policies against requirements. With regards to the main aim of this research, i.e. associating the social perspective to software architectures, it has been shown that in order to address this issue a new type of self-adaptivity has to be brought to the fore toward biddable human interactions.

The approach at hand put forward abstractions (i.e. architectural primitives), that are fundamentally different from any applied to date, offering the chance to a relative control over human interactions in socio-technical systems through a runtime re-evaluation of the normative state of the system. The normative state constitutes of the system context, interactions of humans and responsibilities conferred to them. Social laws together with the role system exert influence on acquiring or enabling more practical capabilities to the system participants due to certain circumstances (i.e. sub-ideal situations).

The contributions of this thesis are summarised with respect to the following perspectives:

1. *Problem Identification*: the problem was identified through medical processes examples putting it into an organisational context that includes clear terms of biddable interactions, organisational structures, organisational control and context-ideality.
2. *Concepts:* I identified social and organisational yet domain-independent architectural elements/concepts that can be distinguished by the proposed architectural framework in its conceptual position. Modelling causal relationships within configurations through using ad hoc and programmed configuration scripts of coordination context supports two directions of interactions handling:
   - Causality simply models "It has to work right" in the sense of if a user switched on the key then the light will go on.
   - Fixation of human participation in interaction protocols to conform to the allowed *static* space of permissions. This might lead to situations such as "It isn't my job", as the current configuration (role and coordination) just doesn't cut it here, even if the interaction is required.
3. *Method:* These concepts are concretised by mapping them to well-known software engineering abstractions such as Problem Frames and a rigorous mathematical approach to formalise operational models, i.e. Graph Transformation. These mappings allow giving semantics to the extended architectural framework and organise its process. Additionally, they support and partially automate the constructive development, as well as the runtime evolution of socio-technical systems.
4. *Solution:* The adopted method as well as its supporting graph-based model validation editor/tool fits into existing development methodologies and influences the resulted socio-technical systems in terms of flexibility and responsiveness to changes in the context.

8.2.1 Putting All Together

What this research promises to achieve through this method and its accompanied concepts, framework and methodological steps is to support the emergence of desirable behaviours in the face of changing context through reasoning about possible social interactions that are filtered by multi-dimensions role proxies and then regulated by norm-based social laws. No one can assume that all required adaptation can be known in advanced yet I claim that the neat separation between permissions and capabilities of organisational role player provides the basis for reasoning about the required purposeful adaptations in response to sub-ideal situations, and thus supports assurance of reaching desired states.

In other words, the description of role capacities: capabilities and permission are well-separated and also distinguished from their concrete role-players. This allows redistributing permissions of the players at runtime; enabling dynamic role binding and managing conflict resolution (see the Section 7.6). The dynamic role binding mechanism is dynamic and is supported by social laws at design time through monitoring and self-adaptivity.

8.3 Evaluation

The previous chapter (i.e. Chapter 7) evaluated how well the extension to the 3Cs architectural approach expresses the qualities necessary for maintaining adaptable socio-technical systems. The proposed method is evaluated pragmatically against the objectives that have been mentioned in Chapter 1 using a number of case studies. I concentrated on the conceptual aspects of the reconfiguration language and how they may contribute to specify the operational semantics of purposeful behavioural specifications without delving into the verification issues that can be evaluated through formal proofs of emergent properties.

The evaluation chapter proposes pragmatic evaluation attempts putting into consideration the difficulties of evaluating languages at the conceptual level. I agree with (Cheng, de Lemos *et al.* 2008b) who stated: “It is unclear whether defining such proofs for emerging systems properties is even feasible”. Runtime assurance techniques may rely on demonstratable properties of adaptation like congruence and stability. This sort of the overall system behaviour’s assurance is beyond current conceptions found in existing architectural frameworks, which rely on deducing “correct” behaviour. The presented example-driven evaluation attempts show that the language is expressive enough and viable to address different domains of application. Moreover, the underlying architectural style supports the under-specification of role binding constraints to be determined at runtime.

These case studies substantiated the contributions of the proposed approach and demonstrated its applicability to perform architectural self-adaptation as a mean to fit in biddable human interactions that are vital for addressing sub-ideal situations. The evaluation focused on demonstrating the required properties of self-adaptivity such as being domain-independent, context-aware and capable of making adaptation decisions at runtime as these properties reflected the generality of the approach and the expressiveness and the flexibility of the proposed architectural primitives.

The first case study demonstrated the ability of the proposed approach to address different categories of application domains that tackle different tasks. Moreover, this case study shows clearly that the problem that the approach addresses within socio-technical system is not domain specific (i.e. problem of empowering people to act out of their role scope or handling their failure in performing required tasks).

The applicability of the adopted approach has been demonstrated in the second case study through going into the details of the methodological steps presented in Chapter 5. The case study illustrated the advantages of addressing sub-idealites through making the difference between routinised transactional view of processes and the ad hoc human-driven view of the same processes. Moreover, the usefulness of the introduced task-based role structure has been clearly explained.

8.4 Future Work

Further improvement can be housed to enhance this approach of modelling biddable social interactions within organisational settings through roles, norms and reconfiguration. It would be beneficial to examine the outcomes of this approach to software architectures and benchmark it against industrial case-studies. There is still a room for further abstractions through which the dynamics of socio-technical systems can be incorporated. For example, after a required behaviour is enabled by the reconfiguration manager, it would be propitious to incorporate a temporal operator to provide timeliness system response to the human reaction towards the required behaviour if the enacting participant refrains from fulfilling his directed obligation. Capturing of such violations is time-sensitive in certain domains. With regards to collaboration, the open delegation approach that the approach supports, it could be extended to the standard delegation model that is found in organisations and well-known IS systems.

From the semantics point of view, the approach advocates a graph-based semantics that caters for proving the validity of the introduced concepts and providing tool support. The approach is based on GT rules of a two-view attributed graph-base model, which provides operational semantics of possible reconfiguration operations. A possible enhancement to the semantics of the advocated approach would be through providing *Denotational Semantics* in the sense of *team automata* (ter Beek, Ellis *et al.* 2003) thus complementing the proposed approach to operational semantic. Advances in this direction would support formal specifications of the self-adaptivity manager (i.e. the harmoniser) that was characterised in Chapter 5.

The idea of considering graphs with proactive components (i.e. social components) as part of the state of the system also poses new challenges for analysis techniques. Validation and simulation-based techniques for GT have usually regarded graphs as passive data structures being manipulated. The uncertainty arising from the involvement of social components requires an integration of GT with stochastic analysis techniques— a topic of research where first results are emerging only recently (Heckel, Lajios & Menge 2006).

The presented approach would benefit from currently undergoing research and implementation-driven taskforces. Monitoring as a means for capturing changes in context is via software intensive devices, requires context-aware software mechanisms that mingle with participants' preferences and conditions. This problem is being studied at high level of abstraction (Salifu, Yu & Nuseibeh 2007). Practically, a task force is getting momentum to incorporate human representation within Web Service Business Process Execution Language WS-PEBL definitions.

8.5 Concluding Remarks

Several significant points have been derived through this research, and highlighted as follows:

- Software architecture technologies advanced throughout the last two decade, Self-adaptive approaches to software architecture have recently captured a large audience. Thus, introducing self-adaptive architectures to the development of socio-technical is becoming an important and a direct research area of computer science.

- Almost every socio-technical environment in the world is eager to exploit the most from the capabilities and the rational of its social components, i.e. human participants, particularly at sub-ideal situations. However, many lack the methodology to utilizing architectural technologies to the greatest extent. The principal requirement of such domains, e.g. healthcare environment, is having generic architecture styles that support biddable interactions; promote flexible organisational structures and management control to keep the good behaviour of the entire system through rewards and sanctions.
- Socio-technical systems require better architectures that support them as well as other systems that require flexible representation of the interaction of their human participants. Furthermore, generic architecture styles and models are necessary, to allow domain-free mechanism, facilitate norm-based reconfigurations and role-based transitions (which can be easily specialised and tailored for applications at hand.)
- The study conducted in the Gastroenterology unit showed that concepts that were promoted in this thesis, in terms of architectural primitives for modelling and reconfiguration, do exist in practice and are laid beyond the capabilities of available software development methodologies. Social laws and roles are made to give technical answers to practical social issues, based on the biddability of system's human participants towards other technological components. Otherwise, a very powerful system can be sometimes useless or even become an obstacle to achieving a stable state because of various missing manipulations, that are technically easy but beyond the authorisation of enacting participants. For example, the coordination-based blocking of enacting a life-saving procedure because of a missing patient's approval should be removed to save the patient's life.
- Abstracting roles from coordination interfaces assists in engineering systems that enable the transition of power from automated software control agents to human operators (e.g. shutting auto-pilot system and flying in the manual mode).

- The emphasis on the necessity for differentiating between qualities and permissions is crucial, together with, providing partial task-based delegations and giving system specifier a multi-scaled power to blend rewards and sanctions to manage violating interactions.
- The 3Cs underlying language is not just an ADL-like notation that advocates the separation of concerns between computation, coordination and configuration, but it also constitutes a vehicle for integrating the high-level concepts promoted by this thesis. Norms, roles and role assignments, as high-level concepts, are associated in this framework with: (1) core business functionalities (i.e. coordinated interactions), (2) evolvable UML-like structures of components and connectors that perform scheduled sets of functionalities (i.e. coordination contexts), and (3) technical interfacing complexities between social and technical components that surpass the capabilities of traditional HCI techniques (i.e. relating process-oriented tasks to coordination interfaces).

The main contribution of this thesis is the leveraging of modelling primitives developed for software architectures to cater for interactions that involve human components. The proposed approach primarily takes into account the unanticipated, non-causal nature of human interactions within organisational settings, and provides a mechanism for adjusting role-based permissions and obligations imposed on monitored interactions between technical and human components so as to react and adapt to changes in the environment in which they operate. Similarly, self-adaptations, which may result from monitoring changes in the environment, are capable of capturing sub-ideal situations and applying reconfigurations on both roles and technical aspects. The approach is equipped with a methodology, a graph-based modelling for role-based adaptations and an easy to use modelling notation for reasoning about the features of these self-adaptations in the face of context changeability and the deviations of human participants from their prescribed routines. Thus, the promoted architectural primitives together with the reconfiguration language and the methodological steps are able to fill the gap caused by the absence of applicable engineering methodologies that address the biddability of social interactions within organisational settings.

Appendix A: Bibliography

(Abowd, Allen *et al.* 1995). Abowd, G.D., R. Allen, and D. Garlan, "Formalising Styles to Understand Descriptions of Software Architecture", *ACM Transaction on Software Engineering and Methodology*, **4**(4), 1995, p. 319-364.

(Ajzen 1991). Ajzen, I., "The Theory of Planned Behavior", *Organizational Behavior & Human Decision Processes*, **50**, 1991, p. 179-211.

(Ajzen 2005). Ajzen, I., *Behavioral Interventions Based on the Theory of Planned Behavior*, 2005, http://people.umass.edu/ajzen/pdf/intervention.pdf, accessed in May, 2008

(Akehurst 2000). Akehurst, D.H., Model Translation: A UML-based Specification Technique and Active Implementation Approach, Ph.D., University of Kent at Canterbury, 2000.

(Akrich 1995). Akrich, M., User Representation: Paractices, Methods and Sociology, in *Managing Technology in Society: The Approach of Constructive Technology Assessment*, A. Rip, T.J. Misra, and J. Schot, (eds.), Printer Publishers: London, 1995, p. 167-184.

(Alexander, Ishikawa *et al.* 1977). Alexander, C., S. Ishikawa, M. Siververstein, M. Jacobson, I. Fikdahl-King, and S. Angel, *A Pattern Language*, Oxford University Press, 1977.

(Allen, Deuence *et al.* 1998). Allen, R., R. Deuence, and D. Garlan, Specifying and Analyzing Dynamic Software Architectures, in *Fundamental Approaches to Software Engineering*, Vol. 1382, Springer-Verlag, 1998, p. 21-37.

(Allen & Garlan 1997a). Allen, R. and D. Garlan, "A Formal Basis for Architectural Connection", *Acm Transactions on Software Engineering and Methodology*, **6**(3), 1997a, p. 213-249.

(Allen & Garlan 1997b). Allen, R. and D. Garlan, "A Formal Basis for Architectural Connection", *ACMTransactions on Software Engineering and Methodology*, **6**(3), 1997b, p. 213-249.

(Anderson, De Lemos *et al.* 2008). Anderson, J., R. De Lemos, S. Malek, and D. Weyns, Modelling Dimensions of Self-Adaptive Systems, in *Software Engineering for Self-Adaptive Systems*, B.H.C. Cheng, et al., (eds.), Vol. 5525, Springer-Verlag: Berlin, Heidelberg, 2008, p. 27-47.

(Andrade, Fiadeiro *et al.* 2002). Andrade, L., J.L. Fiadeiro, J. Gouveia, and G. Koutsoukos, "Separating Computation, Coordination and Configuration", *Journal of Software Maintenance and Evolution*, **14**(5), 2002, p. 353-370.

(Andrade, Fiadeiro *et al.* 2001). Andrade, L., J.L. Fiadeiro, and M. Wermelinger, Enforcing Business Policies Through Automated Reconfiguration, in *Proc. of the 16th International Conference on Automated Software Engineering*: IEEE Computer Society Press, 2001, p. 426-429.

(Andrade, Gouveia *et al.* 2002). Andrade, L., J. Gouveia, G. Koutsoukos, and J.L. Fiadeiro, "Coordination Contracts, Evolution and Tools", *Journal on Software Maintenance and Evolution: Research and Practice*, **14**(5), 2002, p. 353-369.

(Andrade & Fiadeiro 2003). Andrade, L.F. and J.L. Fiadeiro, Architecture Based Evolution of Software Systems, in *Software Architectures*, M. Bernardo and P. Inverardi, (eds.), Springer-Verlag, 2003, p. 148-181.

(Arbab 2004). Arbab, F., "Reo: A Channel-based Coordination Model For Component Composition ", *Mathematical Structures in Computer Science*, **14**(3), 2004, p. 329-366.

(Austin 1962). Austin, J.L., *How to Do Things with Words*, Oxford, Oxford University Press, 1962.

(Bacon, Lloyd *et al.* 2001). Bacon, J., M. Lloyd, and K. Moody, *Translating Role-Based Access Control Policy within Context*, in *Lecture Notes in Computer Science*, 2001, accessed in

(Barbuceanu, Gray *et al.* 1999). Barbuceanu, M., T. Gray, and S. Mankovski, "Role of Obligation in Multi-Agent Coordination", *Applied Artificial Intelligence*, **13**, 1999, p. 11 - 38.

(Bardohl 2002). Bardohl, R., "A Visual Environment for Visual Languages", *Science of Computer Programming*, **44**, 2002, p. 181-203.

(Bardohl, Ehrig *et al.* 2003). Bardohl, R., H. Ehrig, J. de Lara, O. Runge, G. Taentzer, and I. Weinhold, Node Type Inheritance Concept for Typed Graph Tranformation, Technical University f Berlin, Berlin, Germany, TR2003_19, 2003

(Bardohl, Ehrig *et al.* 2004). Bardohl, R., H. Ehrig, J. De Lara, and G. Taentzer, Integrating Met-modelling Aspects with Graph Transformation for Efficient Visual Language Definition and Model Manipulation, in *Fundamental Approach to Software Engineering*, *LNCS*, Vol. 2984, Springer: Berlin / Heidelberg, 2004, p. 214-228.

(Baresi, Ghezzi *et al.* 2004). Baresi, L., C. Ghezzi, and S. Guinea, Smart Monitors for Composed Services, in *Proc. of the 2nd International Conference on Service Oriented Computing*, New York, NY: ACM, 2004, p. 193-202.

(Baresi & Heckel 2002). Baresi, L. and R. Heckel, Tutorial Introduction to Graph Transformation: A Software Engineering Perspective, in *Graph Transformation, 1st International Conference, ICGT 2002*, A. Corradini, et al., (eds.), Vol. 2505, Springer, 2002, p. 402-429.

(Baresi, Heckel *et al.* 2006). Baresi, L., R. Heckel, S. Thöne, and D. Varró, "Style-Based Modelling and Refinement of Service Oriented Architectures", *Journal of Sofware and Systems Modelling*, **5**(2), 2006, p. 187-207.

(Barroca, Fiadeiro *et al.* 2004). Barroca, L., J.L. Fiadeiro, M. Jackson, R. Laney, and B. Nuseibeh, Problem Frames: A Case for Coordination, in *the 6th International Conference on Coordination Models and Languages*, *LNCS*, Vol. 2949, Springer: Berlin / Heidelberg, 2004, p. 5-19.

(Bass, Clements *et al.* 1998). Bass, L., P. Clements, and R. Kazman, *Software Architecture in Practice*, SEI Series in Software Engineering, Reading, MA, Addison Wsley, 1998.

(Batista, Joolia *et al.* 2005). Batista, T., A. Joolia, and G. Coulson, Managing Dynamic Reconfiguration in Component-based Systems, in *Software Architecture*, *LNCS*, Vol. 3527, Springer Berlin / Heidelberg, 2005, p. 1-17.

(Berens 2005). Berens, P., The FLOWer Case-Handling Approach: Beyond Workflow Management, in *Process-Aware Information Systems: Bridging People and Software through Process Technology*, M. Duman, W.M.P. van der Aalst, and A.H.M. ter Hofstede, (eds.), John Wiley & Sons, Inc.: Hoboken, NJ, 2005, p. 386-395.

(Berry & Boudol 1992). Berry, G. and G. Boudol, "The Chemical Abstract Machine", *Theoretical Computer Science*, **96**, 1992, p. 217-248.

(Biermann, Ehrig *et al.* 2006). Biermann, E., K. Ehrig, C. Khler, G. Kuhns, G. Taentzer, and E. Weiss, EMF Model Refactoring based on Graph Transformation Concepts, in *Proc. of the 3rd Internatinal Workshop on Software Evolution through Transformations (SETra '06)*, Natal, Brazil, 2006.

(Boella & van der Torre 2003). Boella, G. and L.W.T. van der Torre, Permissions and Obligations in Hierarchical Normative Systems, in *Proc. of the ICAIL*, 2003.

(Brier, Rapanotti *et al.* 2004). Brier, J., L. Rapanotti, and J. Hall, Problem Frames for Socio-technical Systems: Predictability and Change, in *Proc. of the ICSE, 1st.*

International Workshop on Advances and Applications of Problem Frames (WAAPF 2004), Edignburgh, Scotland, 2004, p. 21-25.

(Brier, Rapanotti *et al.* 2006). Brier, J., L. Rapanotti, and J. Hall, Problem-based Analysis of Organisational Change: A Real-world Example, in *Proc. of the 2006 International Workshop on Advanced and Application of Problem Frames*, 2006, p. 13-18.

(Broersen, Dignum *et al.* 2004). Broersen, J., F. Dignum, V. Dignum, and J.J.C. Meyer, Designing a Deontic Logic of Deadlines, in *Proc. of the Deontic logic in computer science*, Madeira, Portugal: Berlin, 2004, p. 43-56.

(Bryle & Giorgini 2006). Bryle, V. and P. Giorgini, "Self-Configuring Socio-technical Systems: Redesign at Runtime", *ITSSA*, **2**(1), 2006, p. 31-40.

(Canal, Pimentel *et al.* 1999). Canal, C., E. Pimentel, and J.M. Troya, Specification and Refinement of Dynamic Software Architecture, in *Proc. of the 1st Working IFIP Conference on Software Architecture (WICSA1)*: Kluwer, B.V., 1999, p. 107-126.

(Carmo & Jones 2002). Carmo, J. and A. Jones, Deontic Logic and Contrary-To-Duty, in *Handbook of Philosophical Logic*, M. Gabbay and F. Guenthner, (eds.), Vol. 5, Springer, 2002, p. 266-344.

(Castelfranchi 2003). Castelfranchi, C., "Formalizing the Informal?: Dynamic Social Order, Buttom-up Social Control, and Spontaneous Normative Relations", *Journal of Applied Logic*, **1**(1-2), 2003, p. 47-92.

(Castelfranchi & Giardini 2003). Castelfranchi, C. and F. Giardini, Silent Agents. Behavioural Implicit Communication for M-A Coordination and HMI, in *Proc. of the 2nd Annual Symposium on Autonomous Intelligent Networks and Systems*, Menlo Park, CA, 2003.

(Cebulla 2004). Cebulla, M., Modelling Sociotechnical Specifics using Architectural Concepts in *Proc. of the Proceedings of the 2004 ACM Symposium on Applied Computing (SAC '04)*, Nicosia, Cybrus: ACM, New York, NY, 2004, p. 1559-1563.

(Checkland 1984). Checkland, P., *Soft Systems Methodology*, Chichester, John Wiley, 1984.

(Checkland & Scholes 2001). Checkland, P. and J. Scholes, *Soft Systems Methodology in Action*, Chichester, John Wiley & Sons, Inc., 2001.

(Cheng, de Lemos *et al.* 2008a). Cheng, B.H.C., R. de Lemos, H. Giese, P. Inveradi, and J. Magee, (eds.), Software Engineering for Self-Adaptive Systems, LNCS Vol. 5525, Springer-Verlag: Berlin, Heidelberg, 2008a.

(Cheng, de Lemos *et al.* 2008b). Cheng, B.H.C., R. de Lemos, H. Giese, P. Inveradi, J. Magee, J. Anderson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustar, A. Finkelstein, C. Gacek, K. Geihis, V. Grassi, G. Karsai, H. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandloa, H. Muller, S. Prak, M. Shaw, M. Tichy, M. Tivoli, D. Wyens, and J. Whittle, Software Engineering for Self-Adaptive Systems: A Research Road Map, in *Software Engineering for Self-*

Adaptive Systems, B.H.C. Cheng, et al., (eds.), Vol. 08031, Internationales Begegnnuns- und Forschungszentrum fuer Informatik (IBFI): Schloss Dagstuhl, Germany, http://drops.dagstuhl.de/opus/volltexte/2008/1500/, 2008b, p. 1-13.

(Chomsky 1956). Chomsky, N., "Three Models for the Description of Language", *IRE Tramsactions on InformationTheory*, **2**(3), 1956, p. 113-124.

(Clements, Bachmann *et al.* 2004). Clements, P., F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*, The SE Series in Software Engineering, 2004.

(Coakes 2002). Coakes, E., Knowledge Management: A socio-Technical perspective, in *Knowledge Management the Socio-Technical World*, E. Coakes, D. Willis, and S. Clarke, (eds.), Springer-Verlag: London, 2002, p. 4-14.

(Coiera 2007). Coiera, E., "Putting the Technical Back into Socio-technical Systems Research", *International Journal of Medical Informatics*, **76**(1), 2007, p. 98-103.

(Colman & Han 2005). Colman, A. and J. Han, "Organizational Abstractions for Adaptive Systems", *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2005, p. 276.

(Colman & Han 2007). Colman, A. and J. Han, "Roles, Players and Adaptive Organisations", *Applied Ontology: An Interdisciplinary Journal of Ontological Analysis and Conceptual Modeling*, **vol 2**, 2007, p. 105-126.

(Corradini, Heckel *et al.* 2000). Corradini, A., R. Heckel, and U. Montanari, Graphical Operational Semantics, in *the Proc. ICALP 2000 Workshop on Graph Transformation and Visual Modelling Techniques*, A. Corradini and R. Heckel, (eds.), Carelton Scientific: Geneva, Switzerland, 2000, p. 324-336.

(Corradini, Montanari *et al.* 1996). Corradini, A., U. Montanari, and F. Rossi, "Graph Processes", *Funamenta Informaticae*, **26**, 1996, p. 241-246.

(Corradini, Montanari *et al.* 1997). Corradini, A., U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe, Algebraic Approach to Graph Transformation, Part1: Basic Concepts and Double Pushout Approach, in *Handbook of Graph Grammars and Computing by Graph Transformation*, G. Rozenberg, (ed.), Vol. 1: Foundations, World Scientific, 1997, p. 163-246.

(Coulson, Blair *et al.* 2004). Coulson, G., G.S. Blair, P. Grace, A. Joolia, K. Lee, and J. Ueyama, OpenCOM v2: A Component Model for Building Systems Software, in *Proc. of the IASTED Software Engineering and Applications (SEA '04)*, Cambridge, MA, 2004.

(Crook, Ince *et al.* 2002). Crook, R., D. Ince, and B. Nuseibeh, Towards an Analytical Role Modelling Framework for Security Requirements, in *Proc. of the 8th International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ' 02)*, Essen, Germany, 2002.

(Crook, Ince *et al.* 2003). Crook, R., D. Ince, and B. Nuseibeh, "Modelling Access Policies Using Roles in Requirements Engineering"*, Information and Software Technology*, **45**(14), 2003, p. 979-991.

(Crook, Ince *et al.* 2005). Crook, R., D. Ince, and B. Nuseibeh, On Modelling Access Policies: Relating Roles to Organisational Contexts, in *Proc. of the RE 2005: Proceedings of the 13th IEEE International Conference on Requirements Engineering*, 2005, p. 157-166.

(Cuesta, Gómez *et al.* 2003). Cuesta, P., J.C. Gómez, and F.J. Rodríguez, A Framework for Evaluation of Agent Oriented Methodologies, in *Proc. of the Taller de Agentes Inteligentes en eltercer milenio (CAEPIA 2003)*, San Sebastian, Spain, 2003.

(Damianou, Dulay *et al.* 2001). Damianou, N., N. Dulay, E. Lupu, and M. Sloman, The Ponder Specification Language, in *Proc. of the Workshop on Policies for Distributed Systems and Networks (Policy 2001)*, HP Labs, Bristol, 2001.

(de Lara, Bardohl *et al.* 2007). de Lara, J., R. Bardohl, H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer, "Attributed Graph Transformation with Node Inheritance", *Theoretical Computer Science Archive*, **376**(3), 2007, p. 111-138.

(Depke, Heckel *et al.* 2000). Depke, R., R. Heckel, and J.M. Küster, Integrating Visual Modelling of Agent-based and Object-oriented Systems, in *Proc. of the 4th International Conference on Autonomous Agents*, Barcelona, Spain, 2000, p. 82-83.

(DeRemer & Kron 1976). DeRemer, F. and F. Kron, "Programming-in-the-Large Versus Programming-in-the-Small", *IEEE Transactions on Software Engineering SE-2*, **2**(June 1976), 1976, p. 321-327.

(Di Marzo Serugendo, Fitzgerald *et al.* 2007). Di Marzo Serugendo, G., J. Fitzgerald, A. Romanovsky, and A. Guelfi, A Generic Framework for the Engineering of Sel-Adaptive Systems and Self-Organising Systems, School of Computer Science, University of Newcastle, Newcastle, UK, Technical Report, 2007

(Dignum & Kuiper 1997). Dignum, F. and R. Kuiper, "Combining Dynamic Deontic Logic and Temporal Logic for the Specification of Deadlines"*, Proceedings of the Hawaii International Conference on System Sciences*, **30//V5**, 1997, p. 336-346.

(Dignum 2003). Dignum, V., A Model for Organizational Interaction, based onAgents, founded in Logic, University of Utrecht, 2003.

(Dignum, Meyer *et al.* 2002). Dignum, V., J.-J. Meyer, H. Weigand, and F. Dignum, An Organisational Oriented Model for Agent Societies, in *Proc. of the International Workshop on Regulated Agent-based Social Systems: Theories and Applications (RASTA '02), AAMAST '02*: Bologna, Italy, 2002.

(Dignum, Meyer *et al.* 2003). Dignum, V., J.J.C. Meyer, F. Dignum, and H. Weigand, "Formal Specification of Interaction in Agent Societies"*, Lecture Notes in Computer Science*(2699), 2003, p. 37-52.

(Dignum, Vázquez-Salceda *et al.* 2005). Dignum, V., J. Vázquez-Salceda, and F. Dignum, OMNI: Introducing Social Structure, Norms and Ontologies into Agent Organisations, in *Programming Multi-Agent Systems: the 2nd International Workshop ProMAS 2004*, R.H. Bordini, et al., (eds.), Vol. 3346, Springer: Berlin / Heidelberg, 2005, p. 181-198.

(Dijkman, Quartel *et al.* 2003). Dijkman, R., D. Quartel, L. Pires, and M. van Sinderen, An Approach to Relate Viewpoints and Modeling Languages, in *Proc. of the 17th International Enterprise Distributed Object Computing Conference (EDOC '03)*, 2003, p. 14.

(Dobson & Martin 2006). Dobson, J. and D. Martin, Modelling Based on Responsibility, in *Trust in Technology: A socio-technical perspective*, K. Clarck, et al., (eds.), Vol. 36, Springer: Netherland, 2006, p. 39-67.

(Donaldson 2001). Donaldson, L., *The Contingency Theory of Organisations*, Foundations for Organisational Science, Sage, 2001.

(Dowling & Cahill 2001). Dowling, J. and V. Cahill, The K-Component Architecture Meta-Model for Self-Adaptive Software, in *Proc. of the 3rd International Conference on Metalevel and Separation of Crosscutting Concerns*, Vol. 2192, Springer-Verlag, 2001, p. 81-88.

(Dustar 2004). Dustar, S., "Caramba—A Process-aware Collaboration System Supporting Ad hoc and Collaborative Processes in Virtual Teams", *Distributed and Parallel Databases*, **15**(1), 2004, p. 45-66

(Edwards 1996). Edwards, W.K., Polices and Role in Collaborative Systems, in *Proc. of the ACM Conference on Computer Cooperative Work (CSCW '96)*, Cambridge, MA, 1996, p. 11-20.

(Ehrig, Engels *et al.* 1999). Ehrig, H., R. Engels, J. Kerowski, and G. Rozenberg, (eds.), Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools, World Scientific, 1999.

(Ehrig, Kerowski *et al.* 1999). Ehrig, H., J. Kerowski, U. Montanari, and G. Rozenberg, (eds.), Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 3: Concurrency, Parallelism, and Distribution, World Scientific, 1999.

(Ehrig, Pfender *et al.* 1973). Ehrig, H., M. Pfender, and H. Schneider, Graph Grammars: An Algebraic Approach, in *Proc. of the 14th Annual IEEE Symposium on Switching and Automata Theory*: IEEE, 1973, p. 167-180.

(Ehrig, Küster *et al.* 2006). Ehrig, K., J.M. Küster, G. Taentzer, and J. Winkelmann, Generating Instance Models from Meta Models, in *the 8th IFIP WG 6.1 International Conference, FMOODS 2006*, *LNCS*, Vol. 4037, Springer-Verlag: Berlin / Heidelberg, 2006, p. 4037.

(El-Hassan & Fiadeiro 2006). El-Hassan, O. and J.L. Fiadeiro, Role-based Architectural Modelling of Socio-Technical Systems, in *Proc. of the 3rd International Workshop on Coordination and Organisation (CoOrg'06)*, Bologna, Italy: ENTCS, 2006, p. 5-17.

(El-Hassan, Fiadeiro *et al.* 2008). El-Hassan, O., J.L. Fiadeiro, and R. Heckel, Managing Socio-technical Interactions in Healthcare Systems, in *BPM 2007 Workshops, the 1st International Workshop on Process-oriented Information Systems in Healthcare*, H.M. ter Hofstede, B. Benatallah, and H.-Y. Paik, (eds.), *LNCS*, Vol. 4928, Springer: Berlin / Heidelberg, 2008, p. 347-358.

(Ellis, Gibbs *et al.* 1991). Ellis, C., S. Gibbs, and G. Rein, "Groupware—Some Isssues and Experiences", *CACM*, **54**(1), 1991, p. 38-58.

(Emery & Trist 1960). Emery, F.E. and E. Trist, Socio-technical Systems, in *Management Science, Models and Techniques*, C.W. Churchman and M. Verhulst, (eds.), Vol. 2, Pergamon, 1960, p. 83-97.

(Engels, Heckel *et al.* 2000). Engels, G., R. Heckel, and S. Sauer, Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML, in *UML 2000*, 2000, p. 323-337.

(Esteva, Padget *et al.* 2002). Esteva, M., J. Padget, and C. Sieera, Formalising a Language for Institutions and Norms, in *Intelligent Agents VIII: the 8th International Workshop (ATAL 2001)*, J.-J. Meyer and M. Tambe, (eds.), Springer: Berlin / Heidelberg, 2002, p. 348-366.

(Falcone & Castelfranchi 2000). Falcone, R. and C. Castelfranchi, Level of Delegation and Levels of Adoption as the Basis for Adjustable Autonomy, in *6th AI*IA*, *LNCS*, Vol. 1792, Springer, 2000, p. 273-284.

(Felici 2003). Felici, M., Taxonomy of Evolution and Dependability, in *Proc. of the Proceedings of the 2nd International Workshop on Unanticipated Software Evolution, (USE '03)*, Warso, Poland, 2003, p. 95-104.

(Ferber & Gutknecht 1998). Ferber, J. and O. Gutknecht, A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems, in *Proc. of the 3rd International Conference in Multi-Agent Systems*, Paris, France: IEEE Press, 1998, p. 128-135.

(Fiadeiro 2004). Fiadeiro, J.L., *Categories for Software Engineering*, Berlin & Heidelberg, Springer-Verlag 2004.

(Fiadeiro 2007). Fiadeiro, J.L., "Designing for Software's Social Complexity", *Computer*, **40**(1), 2007, p. 34-39.

(Fiadeiro, Lopes *et al.* 2003). Fiadeiro, J.L., A. Lopes, and M. Wermelinger, A Mathematical Semantics for Architectural Connectors, in *Generic Programming*, R. Backhouse and J. Gibbons, (eds.), *LNCS*, Vol. 2793, Springer-Verlag, 2003, p.

(Fiadeiro & Maibaum 1996). Fiadeiro, J.L. and T. Maibaum, A Mathematical Toolbox for the Software Architect, in *Proc. of the 8th international Workshop on Software Specification and Design*, 1996, p. 46-55.

(Fiadeiro & Maibaum 1997). Fiadeiro, J.L. and T. Maibaum, "Categorial Semantics of Parallel Program Design", *Science of Computer Programming*, **40**(1), 1997, p. 111-138.

(Fielding 2000). Fielding, R.T., Architectural Styles and the Design of Network-based Software Architectures, Ph.D. Dissertation, Information and Computer Science, University of California, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

(Fields, Harrison *et al.* 1997). Fields, B., M. Harrison, and P. Wright, THEA: Human Error Analysis for Requirements Definition, Department of Computer Science, University of York, York, YCS-294, 1997

(Finin, Labrou *et al.* 1995). Finin, T., Y. Labrou, and J. Mayfield, KQML as an Agent Communication Language, 1995, p.

(FIPA 2000). FIPA, FIPA Communicative Act Library, Doc. XC00037H, http://www.fipa.org, 2000

(Floyd 2002). Floyd, C., Deconstructing, in *Social Thinking—Software Practice*, Y. Ditritch, C. Floyd, and R. Klischewski, (eds.), MIT Press, 2002, p. 1-28.

(Fontaine 2001). Fontaine, P.-J., Goal-Oriented Elaboration of Security Requirements, M.S. Thesis, Dept. Computing Science, University of Louvain, 2001, http://citeseer.ist.psu.edu/fontaine01goaloriented.html.

(Garlan, Cheng *et al.* 2003). Garlan, D., S.-W. Cheng, and B. Schmerl, Increasing System Dependability through Architecture-based Self-Repair, in *Architecting Dependable Systems*, d. Lemos, C. Gacek, and Romanovsky, (eds.), Springer-Verlag, 2003, p.

(Garlan, Monroe *et al.* 1997). Garlan, D., R. Monroe, and D. Wile, ACME: An Architecture Description Language, in *Proc. of the CASCON`97*, 1997.

(Garlan & Shaw 1993). Garlan, D. and M. Shaw, An Introduction to Software Architecture, in *Advances in Software Engineering & Knowledge Engineering*, Ambriola and Tortola, (eds.), Vol. 2, World Scientific Pub Co.: Singapore, 1993, p. 1-39.

(Garlan, Siewiorek *et al.* 2002). Garlan, D., D. Siewiorek, A. Smailagic, and P. Steenkiste, "Project Aura: Towards Distraction-Free Pervasive Computing", *IEEE Prevasive Computing*, **21**(12), 2002, p.

(Gazendam, Jorma *et al.* 2005). Gazendam, H., R. Jorma, and K. Liu, Organizational Semitotics, in *Proc. of the IAASS 2004 Conference, Round Table Workshop 'An Organisational Semiotic View on Interculturality and Globalization'*, Lyon, France, http://www.irc.rdg.ac.uk/Research/Publications.htm, 2005.

(Gelernter 1985). Gelernter, D., "Generative Communication in Linda", *ACM Transaction Programming Languages and Systems*, **7**(1), 1985, p. 80-112.

(Gelernter & Carriero 1992). Gelernter, D. and N. Carriero, "Coordination Languages and their Significance", *Communication ACM*, **35**(2), 1992, p. 97-107.

(Georgakopoulos, Hornick *et al.* 1995). Georgakopoulos, D., M. Hornick, and A. Sheth, "An Overview of Workflow Management: From Process Modeling to Workflow Automation", *Distributed and Parallel Databases*, **3**, 1995, p. 119-153.

(Ghezzi & Picco 2002). Ghezzi, C. and G.-P. Picco, An Outlook on Software Engineering for Modern Distributed Systems, in *Proc. of the the Monterey Workshop on radical Approaches to Software Engineering*, Venice, Italy, 2002, p. 10-17.

(Gill 1991). Gill, K.S., "Summary of Human-Centred System Research in Europe, Part1", *Information Technology and Changes in Organizational Work*, **13**(1), 1991, p. 7-27.

(Goguen 1973). Goguen, J., Categorial Foundations for General Systems Theory, in *Advances in Cyberneic and Systems Research*, P. Pichler and R. Trappl, (eds.), Transcripta Books, 1973, p. 121-130.

(Goguen 1996). Goguen, J., Parameterised Programming and Software Architecture, in *Proc. of the Symposium on Software Reliability*: IEEE, 1996.

(Goguen & Burstall 1992). Goguen, J. and R. Burstall, "Institutions: Abstrsct Model Theory for Specification and Programming", *Journal of The ACM*, **39**(1), 1992, p. 95-146.

(Gomes, Batista *et al.* 2007). Gomes, A.T.A., T.V. Batista, A. Joolia, and G. Coulson, Architecting Dynamic Reconfiguration in Dependable Systems, in *Architecting Dependable Systems IV*, *LNCS*, Vol. 4615, Springer Berlin/Heidelberg, 2007, p. 237-261.

(Grudin 1994). Grudin, J., "CSCW—History and Focus", *IEEE Computer*, **27**(5), 1994, p. 19-26.

(Guindon, Kanser *et al.* 1987). Guindon, R., H. Kanser, and W. Curtis, Breakdown and Processes During Early Activities of Software Design by Professionals, in *Empirical Studies of Programmers: Second Workshop*, G.M. Olsen and E.S. Sheppard, (eds.), Ablex Publishing Corporation: Norwood, NJ, 1987, p. 65-82.

(Hall, Jackson *et al.* 2002). Hall, J., M. Jackson, R. Laney, B. Nuseibeh, and L. Rapanotti, Relating Software Requirements and Architectures Using Problem Frames, in *Proc. of the Requirements engineering*, Essen, Germany: Ieee, 2002, p. 137-144.

(Hall & Rapanotti 2005). Hall, J. and L. Rapanotti, Problem Frames For Socio-Technical Systems, in *Requirements Engineering for Socio-technical Systems*, J.L. Mate and A. Silva, (eds.), Idea Group Inc.: London, 2005, p. 318-339.

(Hall, Rapanotti *et al.* 2004). Hall, J., L. Rapanotti, K. Cox, S. Bleistein, and J. Verner, An Example of Domain Decomposition through Application of the Problem Frames Approach to Complex Problem, Computing Department, The Open University, Milton Keynes, TR#2004/24, 2004

(Handy 1985). Handy, C., *Understanding Organizations*, 3rd edition ed., Penguin Business, 1985.

(Hansen, Pigozzi *et al.* 2007). Hansen, J., G. Pigozzi, and L. van der Torre, Ten Philosophical Problems in Deontic Logic, in *Proc. of the Dagstuhl Seminar 07122, Normative Multi-agent Systems*, http://drops.dagstuhl.de/opus/volltexte/2007/941, 2007.

(Harrison-Broninski 2005). Harrison-Broninski, K., *Human Interactions: The Heart and Soul of Business Process Management*, Meghan-Kiffer Press, 2005.

(Hausmann 2005). Hausmann, J.H., Dynamic Meta Modelling: A Semantics Description Technique for Visual Modelling Languages, PhD thesis, Universität Paderborn, 2005.

(Hay 2003). Hay, D.C., *Requirement Analysis: From Views to Architecture*, Upper Saddle River, NJ, Pearson Education (Inc.), Prentice Hall PTR, 2003.

(Heckel 1998). Heckel, R., Open Graph Transformation Systems: A New Approach to the Compositional Modelling of Concurrent and Reactive Systems, Techncial University, 1998.

(Heckel, Engels *et al.* 1999). Heckel, R., G. Engels, H. Ehrig, and G. Taentzer, A View-based Approach to System Modeling based on Open Graph Transformation Systems, Vol. vol.2; Applications, Languages and Tools, 1999, p. 639-668.

(Heckel & Guo 2005). Heckel, R. and P. Guo, Conceptual Modeling of Styles for Mobile Systems: A Layered Approach Based on Graph Transformation in *Mobile Information Systems*, Vol. 158, Springer Bostion, 2005, p. 65-78.

(Heckel, Lajios *et al.* 2006). Heckel, R., G. Lajios, and S. Menge, "Stochastic Graph Transformation Systems", *Fundamenta Informaticae*, **74**(1), 2006, p. 63-84.

(Hirsch, Inveradi *et al.* 1998). Hirsch, D., P. Inveradi, and U. Montanari, Graph Grammars and Constraint Solving for Software Architectures Styles, in *Proc. of the 3rd International Software Architecture Workshop (ISAW-3)*: ACM Press, 1998, p. 69-72.

(Hirsch, Inveradi *et al.* 2000). Hirsch, D., P. Inveradi, and U. Montanari, Reconfuguration of Software Architecture Styles with Name Mobility, in *Coordination Languages and Models*, *LNCS*, Vol. 1906, Springer-Verlag, 2000, p. 148-163.

(Hirsch 2003). Hirsch, D.F., Graph Transformation Models for Software Architectue Styles, Departmento de Computación Facultad de Ciencias Exactas Naturales, Universidad de Buenos Aires, 2003.

(Hoare 1985). Hoare, C.A.R., *Communicating Sequential Processes*, Englewood Cliffs, New Jersey, Prentice-Hall, 1985.

(Hofmeister, Nord *et al.* 1999). Hofmeister, R., R. Nord, and D. Soni, *Applied Software Architecture*, Boston, Addison-Wesley, 1999.

(Holt, Ramsey *et al.* 1983). Holt, A.W., H.R. Ramsey, and J.D. Grimes, "Coordination System Technology as the Basis for a Programming Environment", *Electrical Communication*, **57**(4), 1983, p. 308-314.

(Hoover 2006). Hoover, H.J., *Freshman Introduction to the Foundations of Computing* 2006, http://www.cs.ualberta.ca/~hoover/, accessed in 15-09-2007

(Hursch & Lopes 1995). Hursch, W. and C.V. Lopes, Separation of Concerns, Computer Science Department, Northeastern University, Boston, MA, 1995

(Intuitive 2008). Intuitive, *Intuitive Surgical, Inc., da Vinci Surgical System*, 2008, http://www.intuitivesurgical.com/index.aspx, accessed in 01-05-2008

(Issarny, Saridakis *et al.* 1998). Issarny, V., T. Saridakis, and A. Zarras, Multi-view Description of Software Architectures, in *Proc. of the ISAW '98: Proceeding of the 3rd international Workshop on Software Architectures*: ACM Press, 1998.

(Jacko & Sears 2003). Jacko, J.A. and A. Sears, *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, London, Lawrence Erlbaum Associates, 2003.

(Jackson 1995). Jackson, M., *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices*, 1995.

(Jackson 2001). Jackson, M., *Problem Frames*, Addison Wesley, 2001.

(Johannesson & Wohed 1999). Johannesson, P. and P. Wohed, "The Deontic Pattern: A Framework for Domain Analysis in Information Systems Design", *Data and Knowledge Engineering*, **31**(2), 1999, p. 135-153.

(Jones & Sergot 1996). Jones, A.J.I. and M. Sergot, "A Formal Characterisation of Institutionalised Power", *Journal- Igpl*, **4**(3), 1996, p. 427-444.

(Joolia, Batista *et al.* 2005). Joolia, A., T. Batista, G. Coulson, and A.T.A. Gomes, Mapping ADL Specifications to an Efficient and Reconfigurable Runtime Component Platform, in *Proc. of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*: IEEE Computer Society, 2005, p. 131-140.

(Kanger 1972). Kanger, S., "Law and Logic", *Theoria*, **38**, 1972, p. 105-132.

(Kanger & Kanger 1966). Kanger, S. and H. Kanger, "Rights and Parliamentarism", *Theoria*, **32**, 1966, p. 85-115.

(Karsai & Sztipanovits 1999). Karsai, G. and J. Sztipanovits, "A Model-Based Approach to Self-Adaptive Software", *IEEE Intelligent Systems*, **14**(3), 1999, p. 46-53.

(Kast & Rosenzweig 1970). Kast, F.E. and J.E. Rosenzweig, *Organization and Management: A System Approach*, New York, McGraw & Hill, 1970.

(Kayser & Nouioua 2004). Kayser, D. and F. Nouioua, "About Norms and Causes", *International Journal of Artificial Intelligence Tools*, **14**(1-2), 2004, p. 7-24.

(Keeney 2004). Keeney, J., Completely Unanticipated Dynamic Adaptation of Software, Ph.D. Thesis, Department of Computer Science,, Trinity College Dublin, 2004.

(Kendall 1999). Kendall, E.A., Role Model designs and Implementations with Aspect-oriented Systems, in *Proc. of the ACM Conference on Object-oriented Systems, Languages and Applications*, Denver, CO, 1999, p. 353-369.

(Klop 1992). Klop, J.W., Term Rewriting Systems, in *Hanbook of Logic in Computer Science*, S. Abramsky, D. Gabbay, and T. Maibaum, (eds.), Vol. 1, Oxford University Press, 1992, p. 1-116.

(Koch, Mancini *et al.* 2002). Koch, M., L.V. Mancini, and F. Parisi-Persicce, "A Graph-based Formalism for RBAC", *ACM Transactions on Information and System Security*, **5**(3), 2002, p. 332-365.

(Koch & Parisi-Persicce 2002). Koch, M. and F. Parisi-Persicce, Describing Policies with Graph Constraintss and Rules, in *Proc. of the First International Conference on Graph Transformation*, *LNCS*, Vol. 2505, Springer-Verlag: London, 2002, p. 223-238.

(Koutsoukos, Gouveia *et al.* 2001). Koutsoukos, G., J. Gouveia, L. Andrade, and J.L. Fiadeiro, Managing Evolution in Telecommunications Systems, in *New Developments on Distributed Applications and Interoperable Systems*, K. Zielinski, K. Geihs, and A. Laurentowski, (eds.), Kluwer Academic Publisher, 2001, p. 133-139.

(Koutsoukos, Kotridis *et al.* 2002). Koutsoukos, G., T. Kotridis, L. Andrade, J.L. Fiadeiro, J. Gouveia, and M. Wermelinger, Coordination Technologies for Business Strategy Support: A Case Study in Stock-trading, in *Advances in Business Solutions*, R. Corchuelo, A. Ruiz, and M. Toro, (eds.), Catedral Publications, 2002, p. 45-56.

(Kramer 1990). Kramer, J., Configuraion Programming - A Frmaework for the Development of Distributable Systems, in *Proc. of the CompEuro '90*: IEEE Computer Society, 1990, p. 374-384.

(Kramer 1994). Kramer, J., Exoskeletal Software, in *Proc. of the 16th ICSE*, 1994, p. 366.

(Kramer & Magee 1998). Kramer, J. and J. Magee, "Analysing Dynamic Change in Distributed Software Architecture", *IEE Proceedings - Software*, **145**(5), 1998, p. 146-154.

(Kristensen 1996). Kristensen, B.B., Object Oriented Modeling with Roles, in *Proc. of the 2nd International Conference on Object-oriented Information Systems (OOIS '95)*, Dublin, Ireland: Springer, 1996, p. 57-71.

(Kruchten 1995). Kruchten, P.B., "The 4+1 view of Architecture", *IEEESoftware*, **6**(12), 1995, p. 40-55.

(Lee & Bae 2002). Lee, J.-S. and C.-H. Bae, "An Enhanced Role Model for Alleviating the Role-Binding Anomaly", *Software- Practice & Experience*, **32**(14), 2002, p. 1317-1344.

(Lehmann 1980). Lehmann, M.M., Programs, Life Cycles, and Laws of Software Evolution, in *Proc. of the IEEE 68*, 1980, p. 1060-1076.

(Lenz & Reichert 2007). Lenz, R. and M. Reichert, "IT Support for Healthcare Processes: Premises, Challenges, Perspectives", *Data & Knowledge Engineering*, **61**(1), 2007, p. 39-58.

(Lindahl 1977). Lindahl, L., *Position and Change—A Study in Law and Logic*, D. Reidel, Dordrecht, Synthese Library 112, 1977.

(Liu 2000). Liu, K., *Semiotics in Information Systems Engineering*, Cambridge, Cambridge University Press, 2000.

(Liu, Sun *et al.* 2001a). Liu, K., L. Sun, A. Dix, and M. Narasipuram, "Norm-based agency for designing collaborative information systems", *Information Systems Journal*, **11**(3), 2001a, p. 229-248.

(Liu, Sun *et al.* 2001b). Liu, K., L. Sun, A.J. Dix, and M. Narasipuram, "Norm-based Agency for Designing Collaborative Information Systems", *Information Systems Journal*, **11**(3), 2001b, p. 229-248.

(Lock 2004). Lock, S., "The Management of Socio-technical Systems using Configuration Modelling", *Human Systems Management*, **23**(1), 2004, p. 29-47.

(Lock 2005). Lock, S., Strider: Configuration Modelling and Analysis of Complex Systems, in *Proc. of the 21st IEEE International Conference on Software Maintenance (ICSM 2005)*, 2005, p. 495-504.

(Loerke 1990). Loerke, W.C., On Style in Architecture, in *Fundamental Issues*, F. Wilson, (ed.), Van Nostrand Reinhold: New York, 1990, p. 203-218.

(Lomuscio & Sergot 2003a). Lomuscio, A. and M. Sergot, "Deontic Interpreted Systems", *Studia Logica*, **75**, 2003a, p. 69-92.

(Lomuscio & Sergot 2003b). Lomuscio, A. and M. Sergot, "Deontic Interpreted Systems", *Studia Logica*, **75**(1), 2003b, p. 63-92.

(Losavio, Chirinos *et al.* 2002). Losavio, F., L. Chirinos, and M. Pérez, "Attribute-Based Techniques to Evaluate Architectural Styles", *Acta Científica Venezolana*, 2002, p. 130-18.

(Löwe, Kroff *et al.* 1993). Löwe, M., M. Kroff, and A. Wagner, An Algebraic Framework for the Transformation of Attributed Graphs, in *Term Graph Rewriting: Theory and Practice*, M.R. Sleep, M.J. Plasmeijer, and v. Eekelen, (eds.), John Wiley & Sons Ltd., 1993, p. 185-199.

(Lupu & Sloman 1999). Lupu, E. and M. Sloman, "Conflicts in Policy-Based Distributed Systems Management", *IEEE Transactions on Software Engineering*, **25**(6), 1999, p. 852-869.

(Magee, Dulay *et al.* 1995). Magee, J., N. Dulay, S. Eisenbach, and J. Kramer, Specifying Distributed Software Architectures, in *Proc. of the 5th European Software Engineering Conference (ESEC `95)*, Sitges, Spain, 1995, p. 3-14.

(Maibaum 1993). Maibaum, T., Taking More of the Soft of Software Engineering, in *Proc. of the 7th International Workshop on Software Specification and Design*, Redondo Beach, California: IEEE Computer Society Press, 1993, p. 2-7.

(Martin & Somerville 2006). Martin, D. and I. Somerville, Ethnography and Social Structures of Work, in *Structure for Dependability: Computer-based Systems from an Interdisciplinary Perspective*, D. Bernard, C. Gacek, and C.B. Jones, (eds.), Springer: London, 2006, p.

(Medvidovic & Taylor 1997). Medvidovic, N. and R.N. Taylor, A Framework for Classifying and Comparing Architecture Description Languages, in *Proc. of the 6th European Software Engineering Conference held jointly with the 5th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Zurich, Switzerland, 1997, p. 60-76.

(Mens & Wermelinger 2001). Mens, T. and M. Wermelinger, "Formal Foundations of Software Evolution: Workshop Report", *ACM SIGSOFT Software Engineering Notes*, **26**(4), 2001, p. 27-30.

(Mètayer 1998). Mètayer, D.L., "Describing Software Architectures using Graph Grammars", *IEEE Transactions on Software Engineering*, **24**(7), 1998, p. 521-533.

(Metha & Medvidovic 2003). Metha, N.R. and N. Medvidovic, Composing Architectural Styles From Architectural Primitives, in *Proc. of the ESEC/FSE*, Helsinki, 2003.

(Meyer 1988). Meyer, J.-J., "A Different Approach to Deontic Logic: Deontic Logic Viewed as a Variant of Dynamic Logic", *Notre Dame Journal of Formal Logic*, **1**, 1988, p. 109-136.

(Mintzberg 1992). Mintzberg, H., *Structure in Fives: Designing Effective Organisations*, Englewood Cliffs, N. J., Prentice Hall, 1992.

(Moazami-Goudarzi 1999). Moazami-Goudarzi, K., Consistency Preserving Dynamic Reconfiguration of Distributed Systems, Ph.D. thesis, Department of Computing, Imperial College, 1999.

(Moffet 1998). Moffet, J.D., Control Principle and Role Hierarchies, in *Proc. of the 3rd ACM Workshop on Role-based Access Control*, 1998, p. 63-69.

(Moffett & Lupu 1999). Moffett, J.D. and E.C. Lupu, The Uses of Role Hierarchies in Access Control, in *Proc. of the Role-based access control*, Fairfax, VA: New York, 1999, p. 153-160.

(Moran, Thomas *et al.* 1990). Moran, T., P. Thomas, and R. Anderson, The Work-a-day World as a Paradigm for CSCW Design, in *Proc. of the Conference on Computer-Supported Cooperatbe Work*, Los Angles: ACM, 1990, p. 381-393.

(Moriconi & Xiaoli 1994). Moriconi, M. and Q. Xiaoli, Correctness and Composition of Software Architectures, in *Proc. of the ACM-SIGSOFT '94*, New Orleans, LA: ACM Press, 1994, p. 164-174.

(Nickles, Rovatsos *et al.* 2002). Nickles, M., M. Rovatsos, and G. Weiß, A Schema for Specifying Computational Autonomy, in *3rd International Workshop on Engineering Societies in the Agent's World (ESAW '02)*, *LCNS*, Vol. 2577, Springer: Berlin, 2002, p.

(Notkin, Garlan *et al.* 1993). Notkin, D., D.G. Garlan, G., and K. Sullivan, Adding Implicit Invocation to Languages: Three Approaches, in *Proc. of JSSST Symposium of Object Technologies for Advanced Software*, *LNCS*, Vol. 742, Springer-Verlag, 1993, p. 489-510.

(NTSB 1999). NTSB, Aircraft Accident Brief: EgyptAir Flight 990 Boeing 767-366ER, SU-GAP, 60 Miles Soucth of Nantucket, Massachusetts, October 31, 1999, , Nationa Trasportation Safety Board, Washington, DC 20594, NTSB/AAB-02/01, http://www.ntsb.gov/publictn/2002/AAB0201.pdf, 1999

(Odell, Parunak *et al.* 2003). Odell, J., H.V.D. Parunak, S. Brueckner, and J. Sauter, "Changing Roles: Dynamic Assignement", *Journal of Object Technology, ETH Zurich*, **2**(5), 2003, p. 77-86.

(Oreizy, Gorlick *et al.* 1999). Oreizy, P., M.M. Gorlick, R.N. Taylor, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf, "An Architectural-Based Approach to Self-Adaptive Software", *IEEE Intelligent Systems*, **14**(3), 1999, p. 54-62.

(Ould 1995). Ould, M.A., *Business Processes: Modelling and Analysis for Re-engineering and Improvement*, Chichester, John Wiley & Sons, 1995.

(Pacheco & Carmo 2003). Pacheco, O. and J. Carmo, "A Role Based Model for the Normative Specification of Organized Collective Agency and Agents Interaction", *Autonomous Agents and Multi Agent Systems*, **6**(2), 2003, p. 145-184.

(Padmanabhan, Governatori *et al.* 2005). Padmanabhan, V., G. Governatori, S. Sadig, R. Colomb, and A. Rotolo, Process Modelling: The Deontic Way, in *Proc. of the 3rd Asia-Pacific Conference on Conceptual Modelling*, Hobart, Australia, 2005, p. 75-84.

(Perry & Wolf 1992). Perry, D.E. and A. Wolf, "Foundations for the Study of Software Architecture", *ACM SIGSOFT Software Engineering Notes*, **14**(4), 1992, p. 40-52.

(Plotkin 1981). Plotkin, G.D., Structural Operational Semantics, Aarhus University, DAIMI FN-19, 1981

(Prakken & Sergot 1996). Prakken, H. and M. Sergot, "Contrary-to-Duty Obligations", *Studia Logica*, **57**, 1996, p. 91-115.

(Reddy, Pratt *et al.* 2003). Reddy, M., W. Pratt, P. Dourish, and M.M. Shabot, "Sociotechnical Requirements Analysis for Clinical Systems", *Methods of Information in Medicine*, **42**(4), 2003, p. 437-444.

(Reenskaug, Wold *et al.* 1996). Reenskaug, T., P. Wold, and O.A. Lehene, *Working with Objects—The OOram Software Engineering Method*, Addison-Wesley/Manning, 1996.

(Rensik & Distefano 2006). Rensik, A. and D.S. Distefano, Abstract Graph Transformation, in *Software Verification and Validation*, *ENTCS*, Vol. 157, Elsevier, 2006, p. 39-59.

(Ricci 2004). Ricci, A., Viroli, M., Omicini, A., Role-Based Access Control in MAS using Agent Coordination Contexts, in *Proc. of the The AAAI-04 Workshop on Agent Organizations: Theory and Practice*, San Jose, California, 2004.

(Ricci 2002). Ricci, A.O., Agent Coordination Contexts: Experiments in TuCSoN, in *Proc. of the AI*IA/TABOO Joint Workshop (WOA 2002)*, Milano, Italy, 2002.

(Riehle & Gross 1998). Riehle, D. and T. Gross, Role Model Based Framework Design and Integration, in *Proc. of the OOPSLA '98*, New York: ACM Press, 1998, p. 117-133.

(Rozenberg 1997). Rozenberg, G., (ed.), Handbook of Graph Transformation and Computing by Graph Transformation, Vol. 1: Foundations, World Scientific, 1997.

(Russel, ter Hofstede *et al.* 2005). Russel, N., A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst, Workflow Data Patterns, Identification, Representation and Tool Support, in *Proc. of the 24th International Conference on Conceptual Modelling (ER 2005)*, L. Delcambre, et al., (eds.), *LNCS*, Vol. 3520, Springer: Klagenfurt, Austria, 2005, p. 216-232.

(Russel, van der Aalst *et al.* 2005). Russel, N., W.M.P. van der Aalst, H.M. ter Hofstede, and D. Edmond, Workflow Resource Patterns: Identification, Representation and Tool Support in *Proc. of the 17th Conference on Advanced Information Systems Engineering (CAiSE '05),*, O. Pastor and J. Falcão e Cunha, (eds.), *LNCS*, Vol. 3520, Springer: Berlin /Heidelberg, 2005, p. 216-232.

(Salifu, Yu *et al.* 2007). Salifu, M., Y. Yu, and B. Nuseibeh, Specifying Monitoring and Switching Problems in Context, in *Proc. of the 15th IEEE International Requirments Engineering Conference*, Habitat Center, New Delhi, India, 2007.

(Sandhu, Coyne *et al.* 1996a). Sandhu, R., E. Coyne, H. Feinstein, and C. Youmann, "Role-Based Access Control Models", *IEEE Computer*, **29**(2), 1996a, p. 38-47.

(Sandhu, Ferraiolo *et al.* 2000). Sandhu, R., D. Ferraiolo, and R. Kuhn, The NIST Model for Role-based Access Control: Towards a Unified Standard, in *Proc. of the ACN Workshop on Role-based Access Control (RBAC-00)*, Berlin, Germany, 2000, p. 47-64.

(Sandhu, Coyne *et al.* 1996b). Sandhu, R.S., E.J. Coyne, H.L. Feinstein, and C.E. Youman, "Role-Based Access Control Models", *Computer*, **29**(2), 1996b, p. 38-48.

(Schnenberg, Mans *et al.* 2008). Schnenberg, H., R. Mans, N. Russell, N. Mulyar, and W.M.P. van der Aalst, Towards a Taxonomy of Process Flexibility, in *Proc. of the CAisSE '08 Forum*, 2008, p. 81-84.

(Scott & Wanger 2003). Scott, S.V. and E.L. Wanger, "Networks, Negotiations, and New Times: the Implementation of Enterprise Resource Planning into an Academic Administration", *Information and Organization*, **13**(4), 2003, p. 285-313.

(Searle 1969). Searle, J., *Speech Acts*, Cambridge University Press, 1969.

(Searle 2002). Searle, J.R., "Speech Acts, Mind, and Social Reality", *Studies in Linguistics and Philosophy*, **79**, 2002, p. 3-16.

(Sergot 1998). Sergot, M., Normative Positions, in *Proc. of the Workshop in deontic logic in computer science; Norms, logics and information systems new studies in deontic logic and computer science*, Bologna, Italy: Ios, 1998, p. 289-310.

(Sergot 2001). Sergot, M., "A Computational Theory of Normative Positions", *ACM Transaction on Computational Logic (TOCL)*, **2**(4), 2001, p. 581-622.

(Sergot 1999). Sergot, M., (ed.), Normative Positions, Norms, Logics and Information Systems, ed. P. McNamara and H. Prakken, IOS Press: Amsterdam, 1999, 289-310.

(Shaw & Clements 1997). Shaw, M. and P. Clements, A Field Guide to Boxology: Preliminary Classification of Architectural Styles, in *Proc. of the 21st Annual International Computer Software and Applications Conference (COMPSAC '97)*, Washington, D.C., 1997, p. 6-13.

(Shaw & Garlan 1996). Shaw, M. and D. Garlan, *Software Architecture*, Prentice Hall, 1996.

(Sheridan, Corker *et al.* 2006). Sheridan, T., K. Corker, and E. Nadler, Final Report and Recommendations for Research on Human-Automation interaction in the Next Generation Air Transportation System, Report No. DOT-VNTSC-NASA-06-05, 2006

(Sloman & Lupu 2000). Sloman, M. and E. Lupu, Policy Based Network Management, in *Proc. of the Networked Planet: Management beyond 2000; NOMS 2000*: IEEE, 2000, p. 1016.

(Smith & Finger 2003). Smith, H. and P. Finger, *Business Process Management: The Third Wave*, Meghan-Kiffer Press, 2003.

(Sousa & Garlan 2002). Sousa, J.P. and D. Garlan, Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments, in *Software Architecture: System Design, Development, and Maintenance (Proc. of the 3rd working IEEE/IFIP Conference on Software Architecture)*, Bosch, et al., (eds.), Vol. 224, Kluwer Academic Publisher, 2002, p. 29-43.

(Sousa, Poladian *et al.* 2005). Sousa, J.P., V. Poladian, D. Garlan, and B. Schmerl, Capitalizing on Awareness of User Tasks for Guiding Self-Adaptation, in *Proc. of* *the International Workshop on Adaptive and Self-Managing Enterprise Applications*, Porto, Portugal, 2005, p. 83-96.

(Spaak 2003). Spaak, T., "Norms that Confer Competence", *Ratio Juris*, **16**, 2003, p. 89-104.

(Stamper 1994). Stamper, R., Social Norms in Requirements Analysis: An Outline of MEASURE, in *Requirements Engineering: Social and Technical Issues*, Academic Press Professional, Inc.: San Diego, CA, 1994, p. 107-139.

(Steimann 2000). Steimann, F., "On the Representation of Roles in Object-oriented and Conceptual Modelling", *IEEE Data and Knowledge Engineering*, **35**(1), 2000, p. 83-106.

(Steinmuller & Safarik 2001). Steinmuller, B. and J. Safarik, Extending Role-based Access Control Model with States, in *Proc. of the International Conference on Trends in Communication, EUROCON '2001*, Bratislava, Slovak Republic, 2001, p. 398-399.

(Summers, Jansen *et al.* 1997). Summers, R., H. Jansen, P.R. Weller, M.V. Gils, and K. Nieminen, Towards an Optimal Data Set for Intensive Care, in *Proc. of the 19th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Chicago, IL, 1997, p. 1025-1028.

(Sutcliffe 2000). Sutcliffe, A.G., "Requirements Analysis for Socio-Technical System Design", *Information Systems*, **25**(3), 2000, p. 213-233.

(Szyperski 1998). Szyperski, C., *Component Software: Beyond Object-Oriented Programming*, Addison Wesley, 1998.

(Taentzer & Beyer 1994). Taentzer, G. and M. Beyer, Amalgamated Graph Transformation and their Use for Specifying AGG—An Algebraic Graph Grammar System, *LNCS*, Vol. 776, Springer, 1994, p. 380-394.

(Taentzer, Ehrig *et al.* 2005). Taentzer, G., K. Ehrig, E. Guerra, J. de Lara, L. Lengyel, T. Levendovszki, U. Prange, D. Varró, and S. Varró-Gyapay, "Model Transformation by Graph Transformation: A Comparative Study", 2005, p.

(Tan & Then 1998). Tan, Y.-H. and W. Then, Modeling Directed Obligations and Permissions in Trade Contracts, in *Proc. of the 31st Hawaii International Conference on Systems Sciences*, 1998, p. 166-175.

(Taveter & Wagner 2001). Taveter, K. and G. Wagner, A Multi-perspective Methodology Based on Business Rules, in *ER (Workshops)*, *LNCS*, Vol. 2465, Springer, 2001, p. 403-416.

(ter Beek, Ellis *et al.* 2003). ter Beek, M.H., C. Ellis, J. Kleign, and G. Rozenberg, "Synchronizions in Team Automata for Groupware Systems", *Computer Supported Cooperative Work*, **12**(1), 2003, p. 317-370.

(van der Aalst & Jablonski 2000). van der Aalst, W.M.P. and D. Jablonski, "Dealing with Workflow Changes: Identification of Issues and Solutions", *Computer System Science & Engineering*, **15**(5), 2000, p. 267-276.

(van Lamsweerde, Darimont *et al.* 1998). van Lamsweerde, A., R. Darimont, and E. Letier, "Managing Conflicts in Goal-Driven Requirements Engineering ", *IEEE Transactions on Software Engineering*, **24**(11), 1998, p. 908-926.

(von Wright 1951). von Wright, G.H., "Deontic Logic", *Mind*, **60**, 1951, p. 1-15.

(von Wright 1964). von Wright, G.H., "A New System of Deontic Logic", *Danish Yearbook of Philosophy*, **1**, 1964, p. 173-182.

(Weiß, Rovatsos *et al.* 2003). Weiß, G., M. Rovatsos, and M. Nickles, Capturing Agent Autonomy in Roles and XML, in *Proc. of the 2nd International Joint Conference on Autonomous Agent and Multi-Agent Systems*, Melbourn, Australia, 2003, p. 105-112.

(Wermelinger 1999). Wermelinger, M., Specification of Software Architecture Reconfiguration, PhD, New University of Lisbon, 1999.

(Wermelinger & Fiadeiro 2002). Wermelinger, M. and J.L. Fiadeiro, "A Graph Transformation Approach to Software Architecture", *Science of Computer Programming*, **44**(2), 2002, p. 133-155.

(Wermelinger, Lopes *et al.* 2001). Wermelinger, M., A. Lopes, and J.L. Fiadeiro, A Graph Based Architectural (Re)configuration Language, in *Proc. of the ESEC/FSE '01*: ACM Press, 2001, p. 21-32.

(Wieringa 2000). Wieringa, R.J., The Declarative Problem Frame: Designing Systems that Create and Use Norms, in *Proc. of the 10th International Workshop on Software Specifications and Design (IWSSD '00)*: IEEE Computer Society, 2000, p. 75.

(Winograd 1994). Winograd, T., "Designing a Language for Interactions", *Interactions*, **1**(2), 1994, p. 7-9.

(Winograd & Flores 1986). Winograd, T. and F. Flores, "Response to Reviews of Understanding Computers and Cognitions", *Artificial Intelligence*, **31**, 1986, p. 250-261.

(Yao, Moody *et al.* 2001). Yao, W., K. Moody, and J. Bacon, A Model of OASIS Role Access Control and its Support for Active Security, in *Proc. of the SACMAT '01*, Chatilly Verginia, 2001.

(Yellin & Storm 1994). Yellin, D. and R. Storm, "Interfaces, Protocols, and Semi-Automatic Construction of Software Adaptors", *ACM SIGPLAN Notices*, **29**(10), 1994, p. 176-190.

(Yu & Mylopoulos 1997). Yu, E. and J. Mylopoulos, Modelling Organizational Issues for Enterprise Integration, in *Proc. of the Interntaional Conference on Enterprise Integration and Modelling Technology*, Turin, Italy, 1997, p. 529-538.

(Zambonelli, Jennings *et al.* 2003). Zambonelli, F., N. Jennings, and M. Wooldridge, "Developing Multi-Agent Systems: The Gaia Methodology", *IEEE Transactions on Software Engineering and Methodology*, **12**(3), 2003, p. 317-370.

(Zhang, Xu *et al.* 2005). Zhang, S., Y. Xu, and N. Gu, OSM: An Organizational State Machine Model for CSCW Systems, in *Proc. of the 9th International Conference on Computer Supported Cooperative Work in Design*, 2005, p. 883-888.

Appendix B: List of Abbreviations

This appendix contains a key of the most common acronyms and abbreviations used in this thesis. These include both technical terms and popular product names.

3Cs	►	Computation , coordination and Configuration (Business architecture
ADL	►	Architectural Definition Language
AGG	►	The Attributed Graph Grammar System
BIC	►	Behavioural Implicit Communication
BPM	►	Business Process Management
CDE	►	Coordination Development Environment
CSCW	►	Computer Supported Cooperative Work
CTD	►	Contrary To Duty(obligation(s))
DSL	►	Domain-Specific Language
ECA	►	Event-Condition-Action (rules)
EMF	►	Eclipse Modelling Framework
GME	►	Generic Modelling Environment
GMF	►	Graphical Modelling Framework
GP	►	General Practitioner
GT	►	Graph Transformation
GUI	►	Graphical User Interface
HCI	►	Human-Computer Interaction
HIM	►	Human Interaction Management
MDA	►	Model Driven Architecture
MOF	►	Meta-Object Factory
OOAD	►	Object-Oriented Analysis & Design

OOP	►	Object-Oriented Programming
PSM	►	Platform-Specific Model
RBAC	►	Role Based Access Control (models)
RNS	►	Roles, Norms and Sanctions
SDL	►	Standard Deontic Logic
TPB	►	Theory of Planned Behaviour
UML	►	Unified Modelling Language

Appendix C: Glossary

No	Concept	Definition
1	3Cs	An architectural approach to systematic software development, that separates Computation for coordination and Configuration
2	Ad hoc Reconfigurations	Unforeseen changes at design time but which are nevertheless constrained by invariants specified at the ADL level.
3	Behavioural Implicit Communication	The authors of this work believe that Behavioural Implicit Communication is the easiest way to achieve collaboration without explicit communication. BIC does not require special or specialized signals could be the best manner to improve coordination in H-M and MAS domains.
4	Competency	An individual’s actual performance in a particular situation or the ability to integrate, knowledge and skill to perform a task under the varied circumstances of the real-world context
5	Component	A unit of composition with contractually specified interfaces and explicit context dependencies only
6	Configuration	A purposeful collection of inter-related components working together to achieve some common objective. The properties and behaviour of system components are inextricably inter-mingled.
7	Context	A set of assertions representing the cognitive state of the individual or a group and situation state of the world at certain time
8	Contextual Obligations	Interpreting obligations as being relative to a context.
9	Coordination Contexts	Having mechanisms for evolving systems is not the same as prescribing when and how these mechanisms should be applied. Evolution is a process that needs to be subject to rules that aim to enforcing given policies of organizations over the way they wish or require. Also proposing a primitive– for modelling the circumstances in which reconfiguration can and should take place.
10	Coordination Contracts	A set of analysis techniques, modelling primitives, design principles, and patterns that have been developed to externalize interactions into explicit, first class entities that can be dynamically superposed over system components to coordinate their joint behaviour.
11	Coordination	A set of analysis techniques, modelling primitives, design

	Technologies	principles and patterns that have been developed to externalize interactions into explicit, first class entities that can be dynamically superposed over system components to coordinate their joint behaviour.
12	Domain-Specific Modeling (DSM)	A software engineering methodology for designing and developing systems, most often IT systems such as computer software. It involves systematic use of a graphical Domain-specific programming language (DSL) to represent the various facets of a system. DSM languages tend to support higher-level abstractions than General-purpose modelling languages, so they require less effort and fewer low-level details to specify a given system.
13	Dynamic norm compliance	Punishments and rewards required where agents have the freedom to decide whether to conform or violate norms.
14	Dynamic workflows	A model where each participant implements his own workflow, very dynamic and self catered. A specific participant is dynamically chosen for a role in the workflow.
15	Emergent properties	A system exhibits emergent properties when those properties are more than the sum of its parts' properties.
16	Human Interaction Management System (HIMS)	A process modelling and enactment system that provides native support for the six Role Activity Theory object types (Role, Entity, Activity, User, State and Interaction).
17	Indicative mood	What the environment has or will have regardless of the behaviour of the machine.
18	Norms	Normative sentences, entities of sort similar to propositions except they lack truth values.
19	Operational semantics	Abstract machine that treats a program as an instructions sequence on the states of machines.
20	Optative mood	properties we would like a machine to bring about or maintain
21	Programmed Reconfigurations	Foreseen reconfigurations at design time and designed at ADL level.
22	Reification	The act of making an abstract concept or low-level implementation detail of a programming language accessible to the *programmer*. *Actually we abstract the real world as a configuration graph and then we use this model as a reification by the configuration manager*
23	RNS	A formal schema for specifying boundaries of autonomous agent behaviour. It consists of roles. Norms and sanctions.
24	Roles	1) Patterns of behaviour agents must follow in order to respect the dictates of electronic institutions." An analysis

		of agent speech acts in institutional actions." 2) The set of policies for which an automated device is the subject e.g. a BSC reconfiguration agent. An ODP role is an "identifier for a behaviour, which may appear as a parameter in a template for a composite object, and which is associated with one of the component objects of the composite object. 3) in RBAC, a role is properly viewed as a semantic construct around which access control policy is formulated, bringing together a particular collection of users and permissions, in a transitory way (Sandhu et al. 1996). 4) A job function within the context of an organization with some associated semantics regarding the authority and responsibility conferred on the user assigned to the role
25	Self-repair systems	Systems with the ability to adapt themselves at runtime to handle such things as resource variability, changing user needs and system faults.
26	Software Architecture	Software Architecture is the level of software design that addresses the overall structure and properties of software systems
27	Software Component	A unit of composition with contractually specified interface and explicit context dependencies only.
28	Software Engineering	Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software
29	Software Intensive Systems	Software-intensive systems are those complex systems where software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole.
30	Software Practice	Social activity that allows for variety of social science approaches to be applied and takes a close look at how humans act in relation to software."
31	Stigmergy	A functional form of behavioural communication where the communicative end cannot be represented in the agent's mind intention) but it is a functional effect selected by evolution or built in by a designer.
32	Structural Operational Semantics	Recovers compositionality through the syntactic structure of the language., transition is an encoding of its deduction tree.
33	Subsumption	Premises of syllogism or incorporate something under more general category

34	Validations	Early inspection of implications is the best guarantee that the system being developed is actually the system the specifier intends (we call such 'checks' concerning conformity with the specifiers intentions),
35	Verifications	conceptual errors and internal inconsistencies are discovered (we call such checks '')
36	Workflow Norms	One approach to workflow management is to attempt to ensure its compliance by formalizing the business process as with in the automated workflow system, that is codifying the process norms. This works fine in certain incorporate environments, many workplace studies have found the complex work processes are typically carried out within richer ecological settings. In contrast a descriptive codifying of norms, which contribute to work processes can allow us to asses the robustness of the process. Level: social, legal and cultural context

Appendix D: Mapping the 3Cs Extension to the PBT Conceptual Framework

Conceptualising the Approach

With regards to human interactions or contextual changes that cannot be captured by causal architectural primitives in the 3Cs framework, a different view put into consideration the following situations:

- *Events originated from social participants:*
    - If the interaction is not part of the participant's current role configuration yet it is part of his/her role set
    - If the interaction is part of his/her role configuration but:
        - No superposed contract is bridging his/her social component with the technical component
        - Or, the existing contract has coordination rules that block the interaction due to the values of the queried properties associated to the rule's guard condition
    - If the interaction is not part of a role within his eligible role set but he/she has the capability to enact a task that is labelled with this interaction.
- *Events originated from Contextual Changes*
    - Changes that are signified as indicators for sub-ideal situation confer obligation(s) to enact certain human-driven tasks
        - Roles here are considered slots to be filled with presumable or available social components.

The system response towards such events includes:

- Reconfigurations at the role level
    - Authorising an enacting social component to fill in a role slot
    - Unifying an available social component with the role slot as means for the enablement of the task.

- Reconfigurations at the coordination (component level) to equip a role player with the needed resources of the allocated task
  - Adding and removing technical components
    - Software components (e.g. GUIs)
    - Hardware components (e.g. equipment)
  - Adding and removing connectors (contracts)

Mapping the Ideas to PBT Conceptual Framework

I have selected the theory of planned behaviour of (Ajzen 1991, Ajzen 2005) as an analytical framework for ideas proposed in this thesis. The original derivation of the theory of planned behaviour (Ajzen 2005) defined intention and its other theoretical constructs in terms of *trying* to perform a given behaviour rather than in relation to the actual performance. More details about the theory have been explained in Chapter 2. The behaviour in the proposed model is the process of attempting to perform a given behaviour and measures that deal with the actual performance of the behaviour.

The theory of planned behaviour distinguishes between three types of beliefs—and between the related constructs of attitude, subjective norms and perceived behavioural control. Why the necessity of this distinction? The answer is that all beliefs associate the behaviour of interest with an attribute of some kind, be it an outcome, a normative expectation or a resource needed to perform the behaviour. A similar view is maintained with regards to the proposed architectural framework: All types of triggers whether originated by the configuration manager, coordinated or non-coordinated interactions associate the behaviour of interest with an attribute of some kind. This could be an activation of a service call, a normative expectation or a resource reconfiguration needed to perform the behaviour.

Interventions (reconfigurations) directed at behavioural, normative or control beliefs may change the attitude towards behaviour, subjective norms and the perception of behavioural control. These, in turn, change the intention “human attempts” to the desired direction. Interventions designed to change the behaviour can be directed towards one or more of its determinants: attitudes, subjective norms or perception of behavioural control and given adequate control over behaviour. The new intention should be carried out under appropriate circumstances. This theory has been taken as a vehicle to materialise the effects of the reconfigurations and biddable social interactions on the overall system behaviour.

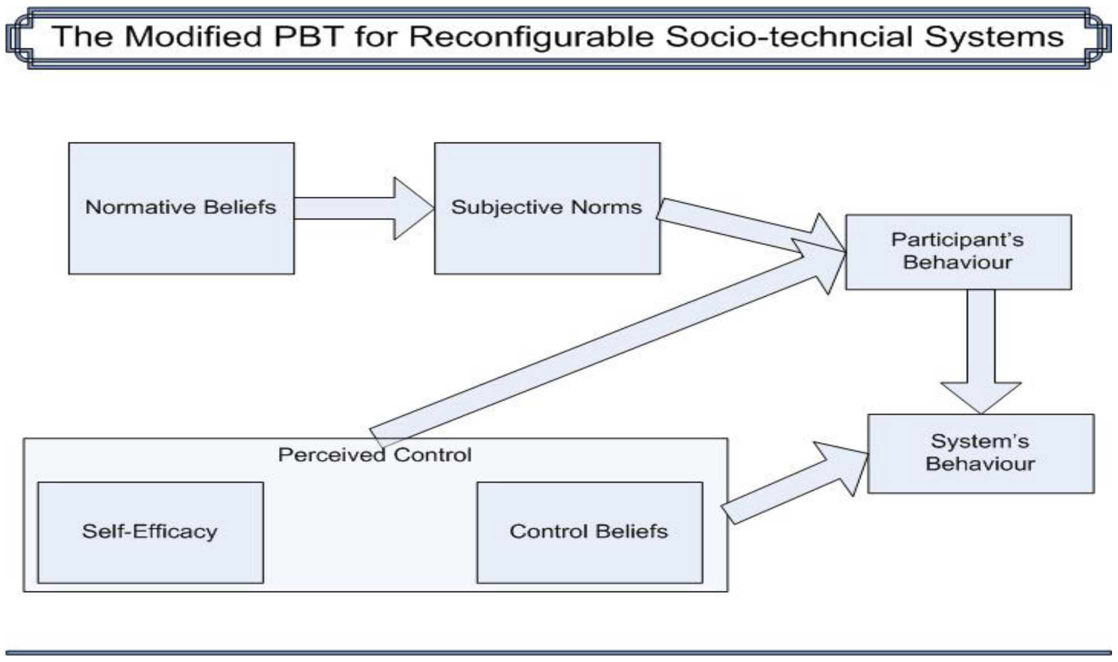


Figure The modified PBT for reconfigurable socio-technical systems

Table 4.2 provides the mapping between out framework concepts and the (PBT) counterpart concepts and Figure 4.9 depicts the relationship between these concepts.

No.	**Planned Behaviour Theory components**	**Socio-architectural concepts**	**Rational for matching/abstraction**
1	Behaviour: human successful	System response	Both constitute the final result
2	Behavioural beliefs: personal beliefs about interactions whether desirable or not	Abstracted away	Agent matter
3	Attitude toward a Behaviour: personal judgment	Abstracted away	Cannot be recorded, differs from a participant to another
4	Normative Beliefs: set of beliefs about what constitute desirable behaviour as defined by individuals and groups	Modelled as social laws with typed anchor roles, obligations, interdictions and contextual constraints	the intention is to avoid agent model's components, norms are codified to be interpreted by reconfiguration manager but still well-known to humans and affect their decisions
5	Subjective Norms: The specific behaviour norms that a individual sets for him/self; what an individual believes that he/she should do	Instantiated social laws through instance roles played by social components and enacted or enabled tasks entries	In the original the subjective norms are usually subsets of normative beliefs. Social laws are always put in place, yet triggers activates them
6	Perceived Behavioural Control: The individual's perception of the ease (or difficulty) of performing a specific behaviour. It has two main components: Self-efficacy and Control beliefs	The perceived effects that result from triggering a social law on a configuration, which result from either contextual changes and/or human intervention	Individual perceptions of ease and difficulties are reduced in the architecture to two levels:1) role and configuration view 2) facilitations and sanction (elaborate)
7	a) Self-efficacy: Confidence of performing the task satisfactorily	Modelled within the type of connector that link the social component with the service and its instance role view including permissions and obligations	Confidence here likelihood of the successful task allocations smoothly modelled through connectors (coordination, social) and capabilities with the role model
8	b) Control beliefs: the likelihood that factors what might prevent a successful completion of the action/task	Factors that might prevent the task allocation depend on the harshness of negative configuration control (i.e. sanctions) (and/or resource allocation difficulties	Participants might decide to refrain from an obligation or enact to a forbidden task as they find the price (sanction) is cheaper than bearing the weight of a sub-ideal situation

The modified PBT for reconfigurable socio-technical systems