REQUEST FOR COMMENTS
Evaluation Framework for Guidelines, Techniques and Modelling Artifacts at the Analysis Stage of AOSE Methodologies to Deal With Complexity*

Joaquín Peña and Rafael Corchuelo

The Distributed Group Seville
University of Seville
ETSI Informática
Avda. de la Reina Mercedes, s/n. Sevilla 41.012 (Spain)
Phone: +34 954 55 38 65, Fax: +34 954 55 71 39
E–mail: joaquinp@tdg.lsi.us.es, web page: www.tdg-seville.info

Abstract Complexity is one of the main challenges of the Agent Oriented Software Engineering (AOSE) field. In this request for comments, we have selected a set of guidelines, techniques and modelling artifacts which we think should be appropriate for improving the ability of current approaches to deal with complex systems. Some of these features are present in some methodologies, has been only identified or even have not been identified.
With this request for comments, we intend to establish some consensus in the elements that AOSE community see appropriate. With this purpose we have developed an on–line form to weigh each feature up available at http://www.tdg-seville.info/AOSE-RFC. The purpose of this document is to detail and motivate each of these elements in an abstract and methodology–independent way.

1 Introduction and Motivation

Structuring models into different abstraction layers and developing them in a top–down and/or bottom up approach are well established techniques in traditional software engineering (hereafter SE) when facing complex systems.

Agent Oriented Software Engineering (hereafter AOSE) is a youth field where researches have focus on properly modelling the features of agents. Unfortunately, most approaches provide plain models not documenting how layered models can be performed. Although this research is rather valuable, more complex systems could be faced if these techniques are introduced in this field.

* The work reported in this article was partially supported by the Spanish Ministry of Science and Technology under grant TIC2003- 02737-C02-01.

Performing models using abstraction layers helps to deal with complexity by facing it incrementally, iteratively and systematically. Top layers can focus on most relevant aspects of the system, while bottom layers represent detailed views of its sub–parts. Layers can be developed incrementally and iteratively limiting the designer scope to certain parts of the system or certain aspects by means of top–down and bottom–up approaches. Top–down approaches start with top layers and refine them iteratively until reaching the level of detail to implement the system. Bottom–up approaches start analysing in detail most important parts of the system developing bottom layers. Later, models are abstracted in order to get an overview of the system, that is to say, top layers. Usually, both approaches are performing in parallel zooming in an out in the system model.

However, although this process is well defined in SE, these techniques cannot be directly applied in the AOSE context. The main problem is that the features of applications that SE currently produce are rather different from the agent's ones. The main difference is that most SE approaches focus the modelling process on data while interactions present a predominant role in the AOSE arena, e.g. [18,26]. This fact forces to perform the development process of MASs from a completely different point of view than in SE.

Interactions are being deeply studied in AOSE, but unfortunately, by the best of our knowledge, the AOSE community has not faced modelling them in a top–down or bottom–up approach or performing layered–models yet. Fortunately, there exist a small set of approaches in SE that focus the modelling process on interactions whose main ideas, adapted to agents, could result quite valuable in this field.

2 Structure of the RFC

We have carefully study current AOSE methodologies and SE methodologies that focus on interactions. As a result of this study, we have selected the set features we see appropriate to develop agent–based applications with a higher level of complexity than current approaches enable. We have also identified several new features. These features are classified into the following categories:

Modelling artifacts: Icons with a concise semantic which are used to graphically represent the system.

Techniques: Procedures that allows to transform models, prove properties of models, analyse them, *etcetera*.

Guidelines: Show the way in where modelling artifacts and techniques should be used, that is to say, best practices to deal with complexity. They can be used as the main rules to establish the software process needed to develop a complete methodology.

When studying related work, we have focused on selecting the features that allow:

1. focusing the modelling process on interactions without taking into account structural constraints

2. modelling a system using different abstraction layers by means of top–down and bottom–up approaches
3. to provide modular descriptions
4. to improve reuse
5. maintaining traceability between requirement and analysis, and between analysis and design
6. modelling open systems

Current methodologies present rather different modelling languages and software processes. Thus, to ease their extension, we do not intend to present a methodology but a set of abstract and methodology–independent features which can be integrated with current methodologies. Hence, we do not base on a certain modelling language and we do not propose any software process.

The final purpose of this document is to establish which of these features are relevant to deal with complexity and which is their relevance degree is. Later, we are going to study current methodologies along with the results of this RFC to establish quantitatively the degree of complexity coverage that they provide. From this study, we intend also to determine quantitatively which is the impact of extending a certain methodology to deal with more complex applications. With this purpose, we have developed an on–line form to obtain feedback from other researches about the features we propose (it can be found at www.tdg-seville.info/AOSE-RFC).

This paper is structured as follows: Section 3 shows the features of systems where this work can be applied. Section 4 summarises the way in where complexity can be conquered. Section 5 shows the modelling artifacts we have selected; Section 6 shows the techniques we propose; Section 7 shows the guidelines to use modelling artifacts and techniques. Section 8 show our main conclusions. And finally, Section 10 is a glossary of terms used in this document. We encourage readers to refer to it since some terms have a different meaning in this document due to the context in where this work is framed, e.g. *behaviour*, *interaction*, etc.

3 Applicability Context

The applicability context of this work can be analysed from different perspectives: (i) regarding complexity and predictability, (ii) the features of agents, (iii) the stages of modelling in where it is applicable.

3.1 Context Regarding Complexity and Predictability

In the field of complex organisational knowledge exchange, decision–making, strategy, and policy–making, Snowden and Kurtz proposed the Cynefin Framework [35]. This framework clarifies the complexity term providing a taxonomy of knowledge–based organisations regarding complexity and predictability. This taxonomy divides an organisation into the following domains whose main features are summarised in Figure 1:

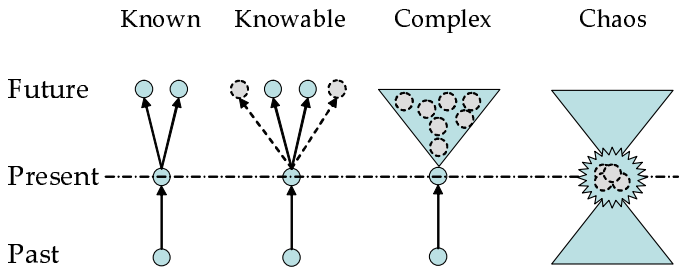

Figure 1. time-cause-efect-complexity

1) Ordered Domain: Stable cause and effect relationship exist. In this domain the sequence of doings of the organisation can be established as a cause/effect chain. It represents the predictable part of the system. This domain is further divided into:

1.2) Known Domain: As can be observed in Figure 1, every relationship between cause and effect are known. The part of a MAS in this domain is clearly predictable and can be easily modelled.

1.1) Knowable Domain: While stable cause and effect relationships exist in this domain, they may not be fully known. In general, relationships are separated over time and space in chains that are difficult to fully understand. The only issue is whether we can afford the time and resources to move from the knowable to the known domain. As can be observed in Figure 1 this is represented by some future states represented using dashed lines that are not known but could be discovered.

2) Un–ordered: Un–Stable cause and effect relationship exist between interactions in the system. It represents the unpredictable part of the system. This domain is also further divided into:

2.1) Complex Domain: There are cause and effect relationships between the agents, but both the number of agents and the number of relationships defy categorization or analytic techniques. Unfortunately, relationships between cause and effect exist but they cannot be predicted. This domain presents retrospective coherence. That is to say, coherence can be only established by analysing past history of the system. Unfortunately, future directions, although coherent, cannot be predicted. As can be observed in Figure 1, past can be understood as a single chain of cause/effects, but when we try to predict future changes, the solution space is too wide to analyse it and we can not establish the order of apparition of future states.

2.2) Chaos Domain: There are no perceivable relationships between cause and effect, and the system is turbulent; we do not have the response time to investigate change.

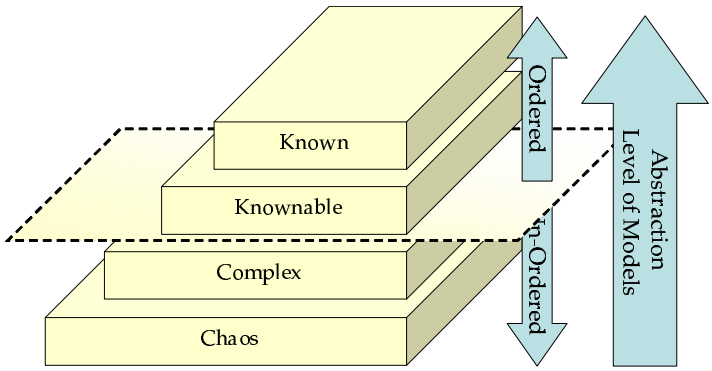

Figure 2. Domain of a problem depending on the abstraction level of models

In addition to this framework, we must introduce another dimension in the categorization of complexity: the level of abstraction of models. As we have depicted in Figure 2, depending on our purpose when studying a certain problem, we can need more or less details of it. That is to say, there exists a certain level of abstraction that provides us only the level of detail we need, and no more. In our categorization, this fact is important since depending on the level of abstraction which we observe a MAS, it can be categorized in the known domain, using the highest level of abstraction, or even in the chaos domain, using the lowest level of abstraction.

As we justify later, we must focus on the analysis phase to conquering complexity. At analysis, the level of abstraction needed is determined by our purpose in this phase: understanding the system. The abstraction capability that current approaches provide is limited and most of them produce models in a single abstraction layer or do not study deeply how these layers can be obtained and maintained. In consequence, systems that fall at the complex or chaos domain can not be modelled by current approaches at the point of view needed for analysis.

Systems that falls in the complex or knowable domain at abstraction level required for analysis can be brought to the knowable domain and even to the known domain using the elements we have selected. This is done by performing a layered model of the system where models are developed systematically and incrementally using different abstraction levels until reaching the level of detail needed for analysis. While current approaches do not provide any modelling artifacts, technique or guideline to do so, but they provide these elements only for performing a complete plain model of the system managing it complexity basing on the analyst experience. Thus, in best cases, we help to bring complex problems from the point of view of current approaches to the known or knowable domains.

3.2 Context Regarding Features of the System

Parunak *et al.* emphasises the importance of *Congruence* in the AOSE field [29]. Congruence is the property that a MAS presents when its system goals are fulfilled and related someway with agents goals. The importance of this property relapses on the fact that a system must cover the purposes with which it has been built. As a result, the congruence of a system measures the utility of the system. Authors claim the importance of this feature in the field and suggest that future research on AOSE must integrate this concept in all phases of developing a MAS. Therefore, we take it into account adapting and extending the related work.

In [29], authors provide an extensive framework of MASs which also contemplate other properties. In addition to congruence, we take into account the following features: we focus on multi–agent systems where exist *Correlation* between agents (exists joint information) and whose agents acts *Coordinately* (implies a causal process where communication between agents exists either directly or indirectly through the environment). We take into account agent's and system's goals, thus covering system that coordinates by *Contention* (agents that coordinate with contradictory goals) or by *Cooperation* (agents with non–contradictory goals). System must also present a certain degree of *Congruence* (agents goals fulfil system goals even when a *Contention* mechanisms exits). Hence, agents must relate *Coherently* (the relation among the agents that yields *Congruence* is *Coherence*).

The features we have selected are also applicable to open systems where we know interaction patterns at modelling time but not the concrete agents who participate on them.

3.3 Context Regarding Modelling Stage

Finally, notice that a MAS organisation can be observed from two different points of views [6,39]:

The interaction point of view: it shows us the organisation as the set of interaction relationships between agents.

The structural point of view: it shows us agents belonging to sub–organisations, groups, teams. In this view agents are also structured into hierarchical structures showing the social structure of the system.

The former is called *Acquaintance Organisation*, and the later is called *Structural Organisation*. Both views are intimately related, but they show the organisation from radically different point of views: a relationship between several agents in one of them does not necessarily implies a relationship in the other.

The relationship amongst both resides in that interactions between agents must take place in whichever structural organisation. We can say that the acquaintance organisation is always contained in the structural organisation. If we determine first the acquaintance organisation and we start from the constraints required for the structural organisation, the acquaintance organisation

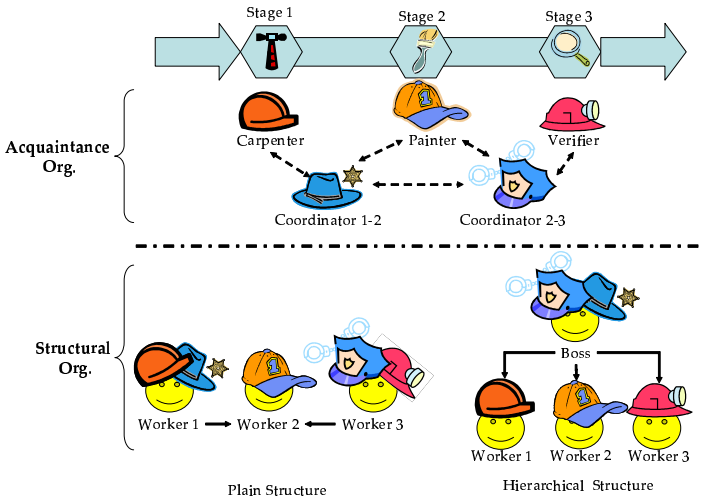

Figure 3. Acquaintance *vs.* Structural Organisation: Manufacturing Pipeline Example

can be mapped onto a certain structural organisation: assigning roles to agents [39]. Thus, any acquaintance organisation can be modelled independently from structural organisation [21].

In Figure 3, we present a simple version of the manufacturing pipeline example presented in [39, pag. 10]. In this example, each stage is performed by agents and we need to coordinated speed of all stages. At the acquaintance organisation, it implies to have a set of roles and interactions between them. At the structural organisation, these roles can be structured to form several organisational structures. For example, as it is shown in Figure 3, we can map the acquaintance organisation over a plain structure, a hierarchical structure, etcetera. This shows that the structural organisation of the manufacturing pipeline is different from its structural organisation.

As other authors, we think that the modelling process should be focused on interactions without taking into account structural constraints, which may further complicate our task. Focusing on interactions and avoiding structural details is an approach ratified by the research done in other more mature fields: i) in the component world, Syzpersky and D'Souza also emphasises the importance of focusing on interaction in stead of architecture (structure in MAS) in complex systems [36, pag. 124][11]; ii) in traditional software engineering there exist also approaches that focus on interactions improving reuse and the ability to manage complexity [33,37]; iii) and finally, the last version of UML provide modelling artifacts to perform interaction–centered modelling [27] although they do not provide guidelines on how using them.

As most complexity is located in the acquaintance organisation, since it focuses on the main source of complexity, interactions, this document only focus

on modelling it. As it is shown in Sections 6.5 and 7.4, some of the techniques and guidelines we present can be also used to map the acquaintance organisation over the structural organisation.

4 Dealing with complexity

Complexity is one of the main challenges of AOSE. Many authors agree on that the complexity of MASs is consequence of interactions [18,26]: *Complexity is caused by the collective behaviour of many basic interacting agents.* James Odell [26]. Thus, as we have presented in Section 3, we focus only on them.

In addition to previous fact, we have to determine how complexity can be managed. In this sense, in [18], Jennings *et al.* adapt the three main principles to manage complexity proposed by Booch in [4] in traditional SE: Abstraction, Decomposition and Organisation/Hierarchy[1] [18]:

Abstraction: It is based on defining simplified models of the systems that emphasises some details avoiding others. It is interesting since it limits the designer scope of interest and the attention can be focused on the most important details at a given time.

Decomposition: It is based on the principle "divide and conquer". It helps to limit the designer scope to a portion of the problem.

Composition: It consists on identifying and managing the inter–relationships between the various subsystems in the problem. It makes possible to group together various basic components and treat them as higher-level units of analysis, and, provides means of describing the high–level relationships between several units.

Unfortunately, we think that current AOSE solutions do not put the proper emphasis on these principles. We have carefully studied previous work in the component world, traditional software engineering and other fields such as distributed systems regarding the previous principles. As a result, we summarise following the strategy we have selected, improved and adapted to the agent field for each principle. We also show their main advantages.

4.1 Abstraction

Abstraction is a powerful tool in software engineering. In traditional SE, classes are the main abstraction since modelling focused on data and processes to transform it. However, as we have shown before, this abstraction does not fit with the features of agents.

In agency, interactions are the main source of complexity; therefore, they must be seen as the main abstraction. The modelling artifact we should provide for them must allow to model interactions of whichever complexity as a single

[1] Notice that hereafter we call it *Composition* in order to differentiate it from the *organisation* term

Abstraction	A.1 Multiparty Interactions Relationships
	A.2 Structure of an Interaction
	A.3 Role Models
	A.4 Behaviour descriptions based on multiparty interactions
	A.5 Behaviour model of Roles and Behaviour model of Role Models
	A.6 Abstract roles and services
	A.7 Model of knowledge consumed/produced by interactions
	A.8 Ontologies of an interaction
	A.9 Abstract knowledge and ontologies
	A.10 ``Classification" association for parameterised role models and structural models
	A.11 ``Generalisation" association for interactions
	A.12 Multiparty interactions must be first class modelling elements
	A.13 Details on how interactions are carried out must be optional at first stages of modelling
	A.14 The modelling artifacts presented omitting behaviour descriptions allows us to model abstractedly and graphically following a BDI approach
	A.15 The modelling artifacts, techniques and guidelines presented allows us to model abstractedly and graphically electronic institutions

Modelling Artifacts Technique Guidelines

Table 1. Abstraction Summary

element. That is to say, as complexity of models depends on the level of abstraction, it must be able to model interactions at whichever level of abstraction. Later, the level of abstraction can be decreased thanks to decomposition and composition.

The purpose of the analysis stage is to understand the system without taking into account implementations details. At the beginning of design the system has to have been properly understood to add implementation details[2]. Therefore, complexity should be managed from the beginning of analysis until conquering it as much as possible.

A plain model of the system, as many current methodologies provide, may contain a huge number of modelling elements and interactions relationships. These models could become unmanageable when the model traverses the frontier between known and knowable domains or even worst if it falls into the complex domain. In these cases, interaction abstractions are not enough powerful since interaction abstraction may allow us to perform a model of the system in the known domain, but its level of abstraction usually will be not enough for our purpose.

This problem has been solved in traditional SE by maintaining a set of models of the system structured in several abstraction layers. In AOSE, we must also provide tools to maintain models consistent and techniques to abstract, divide and conquer, refine and modularise them.

[2] Notice that for complex system analysis and design are usually performed iteratively, thus no real sequential process has to exist

In a layered model of a system, top layers show us abstract models which provide an overview of the system. Bottom layers give us means for detailing top layers nearing our model to a code model. In SE, the completion of layers is usually done in an iterative way where abstract layers are refined to produce bottom layers and bottom layers are abstracted to produce top layers. That is to say, modelling following a top–down approach or a bottom–up approach. It follows that we must provide modelling artifacts, techniques and guidelines to produce layered descriptions. Next two sections summarise how decomposition and composition can help to perform a top–down approach and a bottom-up approach.

In addition, in Table 1, we show a summary of all the features identified for the abstraction principle which are presented in detail in Sections 5, 6, and 7. Notice that each feature in this table, and the followings, are the subject of the on–line poll.

4.2 Decomposition

Abstraction mechanisms may result insufficient to deal with complexity when we model large MASs. In these cases, we can decompose too large models to obtain a set of simpler ones which can be managed easily. Using decomposition in this way we can maintain several layers of abstraction where higher level layers represent abstractedly complex problems and bottom layers store detailed descriptions of sub–part of top layers models obtained by decomposition. Thus, trying to reach the level of abstraction needed for our purpose. That is to say, modelling in a top–down approach.

This separation requires for a set of modelling artifacts, techniques and guidelines to: (i) maintain several separated models and (ii) to determine feasible separations.

The "Role" abstraction is the most important modelling artifact to support decomposition. When a complex structural organisation, formed by agents, is decomposed from the acquaintance point of view, some of their agents have to be also decomposed seeing them from different points of view. Each of these points of view is materialised as the role that the agent play in each decomposed model. We use roles in all models since this concept allows decomposition and abstract us from the concrete agents that we will have in the structural organisation. Consequently, the agent concept is relegated to design where the structural organisation is built by assigning roles to agents (see Composition). Notice that some methodologies mix both concepts since in stead of providing a sequential software process where both stages are performed iteratively (roles and agents are defined iteratively), they provide a sequential process where both stages are mixed explicitly, e.g. [5,6]. In addition, roles also allows us to model the acquaintance organisation orthogonally to the structure emphasising interaction [21].

Finally, reuse and modularity are indispensable features of whichever methodology. Isolated models can be easily reused than complex and large models. Fur-

Decomposition	D.1 Instantiation rules of role models: constraints on the identity of agents who play roles
	D.2 Parameterised Role Models
	D.3 Roles
	D.4 Structure of a Role
	D.5 Initiator and Responder Roles
	D.6 Environmental Roles: environment decomposition
	D.7 Traceability models: traceability between requirement, analysis, and design
	D.8 ``Aggregation" associations for interactions
	D.9 ``Aggregation" association for interaction elements
	D.10 ``Redefinition" association for interactions
	D.11 ``Redefinition" association for interaction elements
	D.12 Techniques to obtain the behaviour of a role from the behaviour of a role model
	D.13 Decomposition by Requirement Goals
	D.14 Decomposition by dependency analysis
	D.15 Technique to extract the static aspect of a role
	D.16 Technique to automatically decompose dynamic aspect of a role
	D.17 Technique to sequence decomposed interactions obtained by goal decomposition
	D.18 Technique to sequence decomposed interactions obtained by dependency decomposition
	D.19 Technique to test inheritance conformance at behavioural aspect
	D.20 Technique to check instantiation rule at runtime
	D.21 Acquaintance organisation must be modelled before structural organisation
	D.22 Decomposition of problems allows us to assign isolated sub-problems to different teams of analysts
	D.23 To perform a decomposition maintaining the level of abstraction we must use interaction model decomposition
	D.24 When we need to refine models we must use interaction decomposition
	D.25 Interactions must be linked with requirement system goals to provide traceability between requirements and analysis and to improve system congruence
	D.26 Hierarchical goal diagrams guide the decomposition
	D.27 Dependency Decomposition must be applied when we are not able to perform more goal decompositions
	D.28 To improve reusability and decrease complexity of decomposed models they must be modelled without taking into account details on other problems
	D.29 Instantiation rules allows to model open system where interaction pattern are known at analysis but not concrete agents who participate on them
	D.30 Instantiation rules plus guards allows to model complex non predictable systems where interaction patterns are known at design time
	D.31 ``Generalisation" association and behaviour inheritance technique provide means for modelling open system in a very flexible way

Modelling Artifacts Technique Guidelines

Table 2. Decomposition Summary

thermore, isolated models allow us to modularise the system description and assigning each sub–part to different teams of analysts.

The techniques to decompose models can be classified into two levels of decomposition depending on their relationship with the interaction abstraction discussed on Section 4.1:

Model decomposition: A decomposition where a complex problem is divided into several separate models without decomposing interactions. For example, a model where we represent a MAS for an e–book store where the agents has to search books and to sell the books that the client select. It can be divided into two models: one containing the problem of searching books and another for modelling how books are purchased. Notice that adding details to decomposed models, where we limit the designer scope, is easier than in complete models. Hereafter, we refer it as *Role model decomposition* (see glossary for role model definition).

Interaction decomposition: A decomposition where abstract interactions are broken into finer grain interactions (see glossary for interaction definition). For example, we can divide the interaction *search_books* into several refined interactions: for example, *establish_preferences*, *selection_of_dealers*, *get_results* and *eliminate_duplicate_results*.

In addition, in Table 2, we show a summary of all the features identified for the decomposition principle which are presented in detail in Sections 5, 6, and 7.

4.3 Composition

A set of models obtained by decomposition of sub–part of the system offers a tour by the system specification but do not offers the big picture of it. Designers/implementers not involved in the analysis phase must be able to get an overview of it. We can compose detailed models to abstract them and represent most relevant features in a single abstract model. Therefore, composition represents the tools needed to perform a bottom–up approach.

Another important fact is that a complete decomposition of a complex problem into a set of orthogonal sub–problems is not usually possible. When several problems have been isolated in order to study them separately, we are ignoring the interdependencies between them. That it is to say, the whole is greater than the sum of its parts. The lost elements may contain crucial features of the system. Hence, composition of models is an important tool to discover such elements when isolated problems have been properly studied. This reduce the complexity we are concerned since only interrelationships has to be managed since problems to be composed have been previously studied.

Finally, as by composition several roles can be merged and an agent is an aggregation of roles, we can use composition to help building the structural organisation. That is to say, models created at analysis can be mapped onto the real world organisation structure as we have shown in the pipeline example.

Composition	C.1 Techniques to obtain the behaviour of a role model from its behaviours of roles
	C.2 Techniques to test behaviours
	C.3 Techniques to merge interactions and their elements
	C.4 Techniques to compose the behaviour models of several role models
	C.5 Techniques to compose several behaviour models of roles
	C.6 Behaviours composition can be done by: sequential composition, parallel composition and composition by interleaving
	C.7 Composition helps us to discover how sub--parts of the system modelled by different teams of analysts relates
	C.8 To identify instantiation rules that involves several models we must compose them
	C.9 To identify coordinator agents we must use composition
	C.10 Composition and Decomposition must be documented in traceability models by associations recommended
	C.11 Behaviours composition must be based on multiparty interactions of a certain granularity when role interleaving is needed
	C.12 Behaviour composition must be done over behaviour model of a role model if most roles have to be composed
	C.13 Behaviour composition must be done over behaviour of roles if few roles have to be composed and most remain unchanged

Modelling Artifacts Technique Guidelines

Table 3. Composition Summary

In addition, in Table 3, we show a summary of all the features identified for the composition principle which are presented in detail in Sections 5, 6, and 7.

5 Modelling Artifacts

Complex systems require for tailored modelling artifacts that allows us to build abstract models, decompose and compose them. We also need modelling artifacts to document the relations amongst the set of models we produce and to structure them into a set of abstraction layers. Following, we present the modelling artifacts we have selected to cover these requirements.

5.1 Interactions

– ***A.1 Multiparty Interactions Relationships:*** *Interaction relationships must be multiparty since it allows us to model high–level social relationships and performs a layered model of the system*
The acquaintance organisation of a large MAS may contain a huge number of relationships between roles. Furthermore, these knowledge level relationships usually relate more than two roles. If we model relationships between roles by means of biparty links, we are forced to decompose mentally n–party relationships where $n > 2$. Thus, decreasing the level of abstraction from the beginning.

For example, a relationship relating three roles can be modelled as a single modelling artifact using multiparty interactions. However, if we limit to binary, we have to use at least two links associations, thus, decomposing mentally it. This forces us to enter into the relationship details.
This motivates the need for relationships that allows relating an arbitrary number of roles. This kind of relationships are provided by many modelling languages. Different terms are used to refer to them: Collaborations in UML 2.0 [27], object diagrams and use cases [4], process models [7], message connections [8], data-flow diagrams [34], collaboration graphs [38], scenario diagrams [33], collaborations [11]. In MAS many authors have also proposed abstraction to model coordinated actions such as, protocols [39], nested protocols [3], interactions [6] or micro-protocols [23]. Hereafter, we refer "multiparty interactions relationships" as interactions.
We must also provide multiparty interactions relationships with the capability of being described internally by means of other interactions in order to provide the possibility of using an arbitrary number of abstraction layers.

– ***A.2 Structure of an Interaction:*** *In order to fully describe a multiparty interaction a set of elements must be attached to them.*
We have selected the following elements to decorate multiparty interactions as a result of carefully studying the bibliography.

system goal it fulfils: A system goal results on an interaction between several agents when it is enough complex to require more than one agent to be achieved. This argues for linking interactions with goals providing a direct transition between requirements and analysis. Goals that are enough simple to be achieved by a single agent, represented as the role it plays, can be linked to them (see D.4).
It allows us to model coherent systems since system goals are directly related with the goal of roles that participates on it [29]. It also allows us to have an artifact that directly represents the functionality required by the system and that provides a direct traceability with goal–based requirement documents.

knowledge consumed and produced in the interaction: It shows the knowledge required by the interaction to fulfil its goal and to generate some new knowledge [6,39]

ontologies of an interaction: They describes the knowledge that is processed in the interaction. It allows to provide a precise specification [5,11]

coordination type its participants follows: It allows to include in models the Parunak's taxonomy of interactions [29]

instantiation rule of the interaction: it shows the constraints over the agents that may play the roles in the interaction (see D.1) [39]

Finally, notice that an interaction relates the set of roles that participate on it; notice that we do not include the agent concept since we leave it for the structural organisation. However, it could be also used in structural models.

– ***A.3 Role Models:*** *Acquaintance sub–organisation must be described abstractedly and provide the capability of being detailed in several abstraction*

layers. If we represent acquaintance relationships using multiparty interactions, role models can be used with this purpose
Multiparty interactions, in stead of binary associations, must be used to represent all the relationships between roles in a certain sub–organisation. This allows modelling whichever acquaintance organisation as abstract as we need. Later, we can apply decomposition to refine models describing its interactions by means of binary associations (see Sections 6.2 and 6.3). We can also use composition to abstract models and identify interrelationships between sub–organisations (see Section 6.5).
Several agents form a certain sub–organisation in order to fulfil a certain system goal that it is complex enough to require several agents. As a result, role models represent how a certain system goal is fulfilled, and its interactions represents the sub–goals of the role model goal. This allows modelling congruence at several levels of abstraction.

– ***A.4 Behaviour descriptions based on multiparty interactions:*** *Behaviour description must be able to express the order of execution of multiparty interactions [14]*
Using messages at the dynamic aspect, although they can be abstract, limit descriptions to relationships among two participants. This hinders the relation between the multiparty interaction–based role models and its behaviour model. It also forces analyst to decompose mentally multiparty relations in role models to a set of binary messages. Therefore, messages–based descriptions decrease the level of abstraction we are able to represent and hinder the traceability between models in both aspects.
Notice that using multi–party interactions, behaviours descriptions can be abstract. Using them we represent the social behaviour, that is to say, which joint tasks (notice that uni–party interactions can be also used) has to be performed to achieve system goals.
We call these behaviour descriptions *abstract joint behaviours*. Hereafter, we refer them as *behaviours* for shortening.

– ***A.5 (i) Behaviour model of Roles and (ii) Behaviour model of Role Models:*** *Behaviour descriptions must be modelled from a single role point of view and from a whole role model point of view*
Modelling artifact to represent a complete description of the behaviour of an acquaintance sub–organisation and the portion of behaviour belonging to a single role may be also appropriate. Both views allow to focus the modelling process on single roles or the MAS/sub–organisation itself. Thus, applying decomposition and improving our capability for dealing with complexity.
The behaviour model of a role model helps us to understand its behaviour abstractedly (they are based on multiparty interactions). A model representing the behaviour of a single role helps us to understand its behaviour in the acquaintance organisation without taking into account details on other roles that do not interact with it.
The behaviour model of a role can be also useful when assigning several roles to the same agent since it help us to group such roles which presents a similar behaviour in the same agent.

– ***D.1 Instantiation rules of role models:*** *Role models must be decorated with instantiation rules: constraints on the identity of agents who play roles [39]*
It must be emphasized that constraints on the identity of agents who play each role are lost in role models. For example, using an agent model (a model that represents the structural organisation) is easy to show that a paper's author agent cannot review its own paper, but when the reviewing process is modelled separated from the submission process using roles, this constraint is lost. Instantiation rules help us to model the constraints on how a certain role model can be instantiated over a certain group of agents. They also guide the mapping of the acquaintance organisation over the structural organisation.
The identification of instantiation rules can be helped by composition. When we assign several role models to a set of agents or when we compose several roles, new instantiation rules can be identified (see Section 7.4 for further details).
Notice that GAIA also proposes to express dynamic restrictions using instantiation rules. That is to say, restrictions on the order in where roles must be played. These restrictions limit us to a certain structural organisation. We leave them for the last steps of analysis when the mapping over the structural organisation is performed. We think that these restriction can be represented easily in the behaviour model of a composite role.
– ***D.2 Parameterised Role Models:*** *Roles, knowledge and services of interactions can be set as parameters to ease the reuse of interactions. Instantiation rules are also crucial elements which must be attached to parameterised role models to properly reuse them [27,2]*
We can parameterise roles, knowledge and services of a role model to describe a parameterised pattern of interaction which is suitable to be reused. Instantiation rules must be attached to parameterised role models in order to show the restrictions on their use.

5.2 Roles

– ***D.3 Roles:*** *We must use roles to be able to decompose the system at static and dynamic aspects. Roles are linked with one or more multiparty interactions and describe the knowledge and services offered on these interactions*
A MAS can be decomposed into several isolated models, each representing how a certain problem is solved, that it is to say, how each system goal is fulfilled. As a result, the same agent may be involved in several models. Therefore, the activity of an agent may be seen from several points of views, one for each problem resolution it participates in. These views are represented by means of the role that the agent plays in each problem.
In [21], Kendall enumerates the main advantages roles in AOSE. We think that these advantages also help us to deal with complexity:
1. Roles emphasise how agents interact. Thus, they allow focusing on the main source of complexity.

2. Roles are orthogonal to agents, that is to say, they allow modeling the acquaintance organisation orthogonally from the structural organisation.
3. Models based on the role concept can be instantiated, generalised, aggregated and specialised into compound models. This improves reuse and enables the use of composition and decomposition to deal with complexity.

– ***D.4 Structure of a Role***
As a result of studying related work, we have selected the following structure of a role:

role interfaces: knowledge and services provided by each role
goals of roles: It improves the traceability between system goals and the sub–goals that each role fulfil to achieve the system goal
cardinality of roles: number of agents that may play it
initiator/s and responder/s role/s: See D.5
guard per pair interaction/role: it shows if the agent playing a role wants to participate in the interaction or not. It also allows us to model proactive agents.

– ***D.5 Initiator and Responder Roles:*** *The agents playing the roles in a role model can be responsible of jointly initiating an interaction or simply participating on it when they are required to do so.*
The agents which play an initiator role are which initiate the interaction. The agents playing the responder roles are required to participate in the interaction. An agent can decide whether participating in an interaction or not by means of guards.

– ***D.6 Environmental Roles:*** *Environment must be also modelled by means of roles using "environmental roles" allowing see environment from different isolated points of view.*
This distinction has been proposed in traditional software engineering [33]. We think it is also appropriate in the agent field since all the elements present in models have not to be agents and they should be also decomposed in separated views.
In a MAS there may exist entities with a low degree of proactivity or even totally passive. These entities can be seen as environmental resources or passive objects. As such environmental entities may be also used to solve several problems; we should also provide different views of them. For example, in a conference management system, a paper database can be used for several purposes: in the reviewing problem we are only interested on the topic of papers, and in the notification process we are interested on authors' e–mails and in the score of the papers. Thus, this database can be represented as two different environmental roles, one for each sub–problem.
This separation can be done using environmental roles. This also helps us to perform a uniform model where few modelling artifacts have to be managed.

– ***A.6 Abstract roles and services:*** *Abstraction mechanisms must be provided for roles and their services*
Defining an interaction involves the definition of the set of roles that participate on it. We have also to show the services they must provide to the rest of

participant, the knowledge that each participant manage and the common ontologies. Therefore, we must also provide abstract modelling artifacts for these elements in order to model interaction abstractedly:

Role abstractions: It consists on a role played by an agent which represents an entire organisations, other system, a legacy systems, and so on. For example, a role banker may represent a complex organisation which when refined may involve several agents. Recursive agents or holonic agents are good example of this kind of abstraction mechanism [17,28].

Services Abstractions: It consists on abstracting the definition of services provided by roles by means of pre and post–conditions (Notice that services can be viewed as interactions with a unique performer). For example, banks agents may provide an *update_funds* service to manage the modification of funds which when refined may require several queries and updates.

5.3 Knowledge and Ontologies

– ***A.7 Model of knowledge consumed/produced by interactions.*** *Multiparty interactions must be decorated with the knowledge consumed/produced by each role [6,39].*
Since interactions amongst agents take place at the knowledge level, we should describe the knowledge and resources consumed/produced by each role in the interaction. Notice that it provides some information about the interactions internals, but without detailing how the process is carried out.

– ***A.8 Ontologies of an interaction:*** *The model of knowledge consumed and produced by each role in the interaction must be precise, that is to say, every knowledge entity used in an interface must be defined in the corresponding ontologies*
These knowledge entities or resources can be modelled abstractedly until we have enough understanding of the system to refine them; however models must be precise at any abstraction layer. Representing the knowledge consumed/produced without the ontologies that define this knowledge is meaningless, thus we must attach a set of ontologies per interaction. The ontologies of an interaction describe the common knowledge that agents must manage to be able to engage in the problem resolution represented by it.
We have selected ontologies as an important element of models as a result of studying PASSI [5], and [11] in the component arena.

– ***A.9 Abstract knowledge and ontologies:*** *Abstraction mechanisms must be provided for knowledge and ontologies*
Defining an interaction involve the definition of the set of elements. In order to model interaction abstractedly we must provide abstract modelling artifacts for knowledge and ontologies:

Knowledge Abstractions: It consists of representing the knowledge owned by each agent in an abstract way at stages where details are not known. For example, banks' agents may own the knowledge about their account

which we can represent abstractly as knowledge entity called *account*. *account* represents abstractly all the knowledge that agents must manage about accounts which refined my contain knowledge about the account number, the funds, the credit availability, etc.

Ontologies: Ontologies must be also abstracted.

5.4 Traceability

– ***D.7 Traceability models:*** *A complex system modelled in several abstraction layers produces a high amount of models. We must provide traceability models in order to maintain them ordered and to provide traceability between requirement, analysis, and design [11].*
The main source of complexity is the interaction. As we have shown previously, interactions can be related with system goals, thus with functional requirements of the system. In a complex system, we can have a huge number of interactions and role models using them. Furthermore, each role model is detailed in subsequent layers by means of other finer–grain interactions. Such amount of models has to be managed carefully. A traceability model is the solution we have selected and adapted to solve this problem. They are based on refinement documents of Catalysis [11].
Each node in a traceability model represents an interaction and it is related with finer and coarse–grain interactions by means of the associations in next points. As interactions are linked with system goals and they are used in role models, this model provides us a direct traceability with goal–based requirement documents and it shows us which models details how each functional requirement is achieved. Notice that each model is related with the model in subsequent layers since they are obtained by composition or decomposition of other models (see Section 6). Traceability models are used to show these relations and also show us which interactions we have used in each abstraction layer.
In addition, in a top–down or bottom–up approach, we must transit from requirements to analysis and from analysis to design iteratively. We can use traceability diagrams in order to clarify this transition documenting and justifying analysis and design decisions clearly. That is to say, we can decide between several available decompositions or compositions basing on functional and non–functional requirements and document such decisions in traceability diagrams.
Finally, as this model can provide us functional traceability, it also helps to identify clearly which roles or sub–organisations are affected by future extensions/modifications of the system.

– ***A.10 "Classification" association for parameterised role models and structural models***
Parameterised role models can be instantiated to obtain a concrete role model. The relation between them is characterised by "classification" association. We must document instantiations of parameterised role models in traceability models.

Notice that another level of instantiation arises when we instantiate a role model over a certain set of agents, that it is to say, a certain structural organisation. These relationships can be also represented using classification. Both kinds of classification associations can be distinguished by a stereotype or a different graphical representation.

- ***D.8 "Aggregation" associations for interactions:*** *Multiparty interactions can be decomposed, thus we must relate bottom layer interactions with top–layer interactions by the "aggregation" association*
  If we decompose interactions into several finer grain interactions we obtain a new layer. Relating high–level interactions with the interactions resultant of decomposition provides us traceability from requirements to design. Abstract interactions must be related with the set of interactions that refine it. The association we propose is "aggregation" to be used in traceability diagrams. For example, we can decompose the interaction *search_books* to obtain the finer grain interactions: *establish_preferences*, *selection_of_dealers*, *get_results* and *eliminate_duplicate_results*. All of these finer–grain interactions can be related with the interaction *search_books* by "aggregation".
- ***D.9 "Aggregation" association for interaction elements:*** *As a result of decomposing an interaction, its elements can be also decomposed. Hence we must relate these modelling artifacts by means of a "aggregation" association*
  Notice that when an interaction is decomposed it also implies to decompose it roles, thus roles must be also related by means of an "aggregation" association. Services, knowledge, instantiation rules and ontologies can be also decomposed into finer–grain elements. They could be also related by this association.
- ***D.10 "Redefinition" association for interactions:*** *As a result of decomposing or composing interactions, we may have to redefine them. Hence we must relate interactions that are redefined by means of a "redefinition" association*
  As a result of decomposing or composing models, we may discover new requirements that force us to redefine some interactions. In consequence, we must relate original interactions with redefined ones in traceability diagrams.
- ***D.11 "Redefinition" association for interaction elements:*** *As a result of decomposing or composing interactions (see Sections 6.2 and 6.3), its elements can be also redefined. Hence we must relate these modelling artifacts by means of a "redefinition" association*
  As a result of a decomposition or composition, we can easily focus on its sub–parts and find inaccuracies in the elements of initial interactions. In consequence, these elements must be redefined and we must be able to represent their relation in traceability diagrams.
- ***A.11 "Generalisation" association for interactions:*** *Generalised role models can be defined in abstract layers. Later these role models can be extended by inheritance to fit certain situations. The relation between these models can be documented by generalisation association.*
  Notice that this association implies inheritance at static and dynamic aspects. Regarding static aspect, OO inheritance concept can be applied (in-

terface inheritance). But at dynamic aspect, we must work with behaviour inheritance [24]. We say that a behaviour specification B specialises A, if the traces B produces are contained in the traces A produces. In this context, a trace is defined as the sequence of interactions that role model behaviour may produce.

6 Techniques

Modelling using abstraction layers should be helped by a set of techniques that help us to systematically obtain top layers from bottom layers (a bottom–up approach) or bottom layers from top layers (a top–down approach). Furthermore, inaccuracies and mistakes may appear even more in complex systems. Discovering them is a tedious and time–expensive task, thus, if possible, we should also provide techniques to help this task.

6.1 General Techniques

- ***C.1 Techniques to obtain the behaviour of a role model from its behaviours of roles.***
  Notice that when using multiparty descriptions both behaviour models are equivalent [32]: one is distributed over roles and the other is centralised as the role model behaviour. Maintaining both descriptions consistent manually may be a difficult task. Thus, mechanisms to transform automatically all the behaviours of roles in a role model to the behaviour of the role model is rather valuable to conquer complexity by automating this transformation.
- ***D.12 Techniques to obtain the behaviour of a role from the behaviour of a role model.***
  We have to work with the behaviour model of roles or with the behaviour model of the whole role model depending on the technique we are applying. The mechanism to transform automatically the behaviour of a role model into a set of behaviours of roles is also rather valuable to conquer complexity.
- ***C.2 Techniques to test behaviours*** *In complex system behavioural descriptions may result on difficult to understand models. We can palliate this problem using the modelling artifact shown in previous sections. But, we can also provide automatic algorithms that help us to ensure their correctness*
  Behaviour descriptions may contain mistakes, for example, situations not taken into account, and deadlocks. Detecting this kind of mistakes is expensive. Thus, we could provide automatic techniques to test behaviour descriptions.

6.2 Techniques to Determine Feasible Decompositions

Decomposing a problem into a set of sub–problems, may result a hard task if we do not provide techniques for identifying such divisions.

Following, we discuss on two techniques for determining feasible decompositions. The former, goal decomposition, can be used for performing interaction and role model decomposition, and the latter, dependency decomposition, can be used to perform interaction decomposition.

– ***D.13 Decomposition by Requirement Goals:*** *We can decompose the system by means of system goals or use cases (functional decomposition). It improves system congruence*
Since agents are limited to some environment and have limited abilities, complex problems are usually solved by a set of agents. For example, in a conference management system where a system goal is to select papers, a blind reviewer agent is only concerned with revision of its papers and it is not able to send the results of the review to the authors or decide if the paper is accepted or not. Hence, other agents are needed to fulfil the system goal.
If a complex system goal has to be fulfilled, we need several agents to achieve it and hence, at least one interaction between them. That is to say, a system goal results on an interaction between several agents when it is enough complex to require more than one agent to be achieved. This argues for linking interactions with goals providing a direct transition between requirements and analysis. Simpler goals can be also linked with roles.
Identifying system goals and decomposing them into finer–grain goals provide us means for an initial functional division of the system where interactions and their roles can be identified. This argue for goal–oriented requirements techniques that provides us hierarchical system goal diagrams [9,10,20,39]. By them, we can perform a functional decomposition of the system which may be also performed by analysing requirements document based on use cases, but with a higher effort.
We can perform the following types of decompositions using goal–based requirement documents:
Interaction Goal Decomposition: When a requirement goal is refined by several goals it implies that several sub–goals must be fulfilled to achieve it (by "and" decomposition of goals). This shows us how to decompose an interaction which represents a high–level goal into a set of finer–grain interactions, one per sub–goals to be fulfilled. In "or" decomposition we must fulfil whichever of them.
Without techniques to perform and guide this kind of decomposition, the refinement of multiparty interactions to biparty primitives, such as those used at design and code level, may result on intuitive decisions without any guidance.
By decomposing a complex interaction, we also divide and conquer complex problems. Interaction decomposition decreases the number of participants of interactions, which lead us to easier implementations (the protocol to coordinate n parties is more difficult than such for two parties) as it is shown in [1,15,30]. Furthermore, the complexity of the knowledge processed in each interaction decreases, thus easing their internal design.

Role Model Goal Decomposition: We can also perform a decomposition where interactions are not divided. If we have a model where several sub–goals are represented by means of interactions in a single model, and they are related by "and" of several high–level goals, each of the high–level goals shows us a set of interactions that can be separated into a decomposed model.

For example, if we have two system goals of the conference management system: *review_papers* and *communicate_results*, each of them can be refined by and decomposition by several sub–goals. If we deal with a single role model which contains the interactions linked with the sub–goals of both, our model can be decomposed into two role models, one to show how the sub–goals *review_papers* is achieved, and another for the sub–goals *communicate_results*.

– ***D.14 Decomposition by dependency analysis:*** *When goal decomposition has been exploited, system can be decomposed by analysing the dependencies between knowledge/services required/consumed by each role in a role model [30]*

It consists on grouping roles that highly depend inside in the context of an interaction. Since roles relate in order to exchange knowledge and/or problem–solving capabilities, we can analyse the dependencies regarding them. Each group of dependent roles can be related by a new set of finer–grain interaction, thus reducing the complexity of models and providing means for identifying refined acquaintance sub–organisations. In [13,30,31] authors describe several techniques to decompose multiparty interactions.

For example, in a *purchase* interaction, which is not further detailed at requirement documents, several agents may be involved: a *seller's bank* agent, a *buyer's bank* agent and a *point of sales* agent. *Seller's bank* depends on the knowledge provided by the *point of sales*, *buyer's bank* also depends on the *point of sales*, and *buyer's bank* and *seller's bank* do not depend. Hence, we can decompose the *purchase* interaction into two interactions: *determine_and_pay_goods* between *buyer's bank* and *point of sales*, and *store_profits* between point of sales and *seller's bank*.

6.3 Techniques to Support Decomposition

– ***D.15 Technique to extract the static aspect of a role:*** *When we perform a decomposition, roles have to be also decomposed. Techniques to extract static features are needed*

The process of engineering a software application is done in an iterative way. In consequence, we have to deal with descriptions where a set of roles may be involved in the resolution of a certain problem which can be further decomposed. Techniques that allow us isolating the static and dynamic aspect of a role regarding a certain problem are also quite appropriate.

At the static aspect, we need techniques to extract those acquaintance relations (interactions) which are used to solve the sub–problem we are isolating.

In addition, if a role participates in several interactions, we need to separate the part of the interface used by the isolated interactions.

- ***D.16 Technique to automatically decompose dynamic aspect of a role:*** *Techniques to automatically decompose the dynamic aspect of roles from a complex model given a certain decomposition are needed. This is a hard task if we perform it manually. Automating it helps us easing decomposition and conquering complexity*
  At the dynamic aspect, we need techniques to decompose the behaviour of a role regarding the sub–problem we are isolating. That it is to say, extract the order of execution of such interactions in where the role is involved in the problem we are isolating.
- ***D.17 Technique to sequence decomposed interactions obtained by goal decomposition:*** *We must provide techniques to sequence a set of decomposed interactions automatically if possible*
  By requirement–goal decomposition we can break an interaction into several finer–grain interactions. We need techniques to determine how decomposed interactions sequence. In this case, requirement document may help us in this task.
- ***D.18 Technique to sequence decomposed interactions obtained by dependency decomposition:*** *We must provide techniques to sequence a set of decomposed interactions automatically if possible*
  In dependency decompositions we know how knowledge and/or services relate. When an interaction works with the result of other interactions (it exists dependencies between their knowledge and/or services) it must be executed after their inputs have been calculated in the other interactions. As a result, we can automatically sequence them analysing their dependency graphs. Therefore, we can establish automatically how interactions sequence in most cases.

6.4 Techniques to Support Open Systems

- ***D.19 Technique to test inheritance conformance at behavioural aspect.***
  We must provide techniques to ensure that the behaviour of an inherited model conforms to the behaviour of its ancestor. It allows us to support open systems since new roles can be added to the system if their behaviour inherit from a generalized one.
- ***D.20 Technique to check instantiation rule at runtime.***
  We must provide techniques that allow checking if instantiation rules are fulfilled when a certain role model is instantiated by a set of agents.
  At runtime we can also add new interactions, but we have to check if they fulfil constraints imposed by instantiation rules.

6.5 Techniques to Support Composition

- ***C.3 Techniques to merge interactions and their elements***

We must provide techniques to perform composition at the static aspect since roles and interactions can be merged in composite role models. For example, when we merge several interactions, we have to determine the new initiator, the participants (may be the sum of all of them or not) and so on. When merging several roles, their interfaces must be also merged and their links with interactions.

- ***C.4 Techniques to compose the behaviour models of several role models:*** *Several behaviour models of several role models must be merged to obtain the behaviour model of the composite role model*
  Techniques to compose the behaviour models of several role models have to be provided in order to determine the behaviour model of the composite role model. It can be used to obtain the big picture of several problems modelled isolated or to map several role models to a certain structural organisation.
- ***C.5 Techniques to compose several behaviour models of roles:*** *Several roles that are seen as the same role in a composite role model must be merged*
  When several roles are composed, their behaviour description (the sequences of interactions they execute) must be also merged. As a matter of fact, techniques to compose the behaviour of several roles that are seen as the same role in a composite role model are quite valuable.
  Furthermore, in a structural organisation several roles are mapped onto the same agent, thus we need a way of merging the behaviour of several roles into single agent behaviour.
- ***C.6 Behaviour composition can be done by: sequential composition, parallel composition and composition by interleaving***
  We need techniques to compose behaviour descriptions (behaviour models of role models and behaviour models of roles). Several techniques can be identified:
  Sequential: The behaviour of each model is executed atomically in sequence with others.
  Parallel: The behaviour of each model is executed in parallel. It is the less common in acquaintance organisation since models that can be executed in parallel are independent and thus, we do not usually compose them.
  Interleaving: It is the most common since the composition of several role models that are highly related implies to highly interleave them. Notice that the proper interaction granularity must be used for this kind of composition (see C.11).

7 Guidelines

In previous sections, we have presented modelling artifact and techniques to deal with complexity. However, we have not coped with another important dimension that a methodology must provide, its software process. As we intend to provide general solution not coupled with any methodology, this section presents a set of guidelines that a methodology should follow in its software process to deal with complexity:

7.1 General Guidelines

– ***D.21 Acquaintance organisation must be modelled before structural organisation.***
Separating agents and roles allows us to be able to distinguish between two architectural levels:

Acquaintance Organisation it is described by means of role models. It specifies the acquaintance relationships in the system by means of interactions and roles.

Structural organisation it represents a concrete instantiation of a set of role models over a set of agents (this process is guided by instantiation rules). It specifies the organisation structure based on the real organisation where the system has to be deployed. That it is to say, it is the mapping of the acquaintance relationships identified at analysis onto a concrete organisational structure.

Since interactions are the main source of complexity, we should not bother about organisation structure at analysis. This eases the process of understanding the complex behaviour of a MAS. Furthermore, we should not take into account organisation structure at the beginning of analysis.
Notice that if we start analysis with a certain organisational structure (by means of agents) based on the real organisation where the MAS is going to be deployed, the initial subdivision in interactions and roles may not be optimal. The main reason is that real organisation is not always modelled from the interaction point of view. For example, teachers in the group of teachers of a certain subject are grouped because they teach the same subject, but it does not imply any acquaintance relation between them.
Another risk of starting analysis with the real structural organisation in mind is that we mimic its mistakes. The mistakes of organisations has been well studied in economics [25]: agents coordinated by more than one agent, agents introduced to cover the relations between several sub-organisations, agents with same profile placed in different sub–organisations when only one agent and a proper structure can be used, etcetera . Thus, when real organisations impose constraints on the system architecture, abstract organisations may be modelled by means of role models at analysis to later map them onto concrete agents structured as the real organisation by the composition principle.

7.2 Abstraction Guidelines

– ***A.12 Multiparty interactions must be first class modelling elements***
Interactions must be seen as first class modelling elements since they are the main source of complexity and one of the most important features of agents.

– ***A.13 Details on how interactions are carried out must be optional at first stages of modelling***
Details on the relationships between roles are not accurately known at first modelling phases. For example, we do not accurately know how is going to

be carried out a purchase relationship at the first approach to the problem. If we have to model how it is carried out, we need a deep knowledge of the agents involved what is not possible at analysis stage and counterproductive to maintain an abstract description. This argues for descriptions techniques that do not force us to describe how they are carried.

- ***A.14 The modelling artifacts presented omitting behaviour descriptions allow us to model abstractedly and graphically following a BDI approach and, in conjunction with techniques and guidelines, it enables the construction of a methodology for complex BDI systems***
  Modelling artifacts we present can be used to model BDI and BDJI (Belief-Desire-Joint-Intention Architecture) [16,19]. In addition techniques and guidelines can help us in the modelling process [22].
  Believes can be modelled as knowledge entities in interfaces of roles and data of interactions.
  Desires can be modelled as goals of roles, or even using goals of interactions if we follow the Belief-Desire-Joint-Intention approach.
  And *intentions*, can be modelled as instantiation rules and guards showing the situations where each interaction is part of the plan to achieve the agent's desires. In addition, we can use behavioural models when following a BDJI approach.
  Notice that using the modelling artifacts, techniques, and guidelines in this document we are able to model BDI systems abstractedly and following a layered approach. For example, coarse grain interactions can be used to describe intentions abstractedly, and finer–grain interactions can be used to detail them producing several layered models of the system.
  In this way, we can produce BDI models of the system in several abstraction layers gaining all the advantages presented in this paper. Later these models can be refined by a traditional BDI formal approach. Abstract models produced using the elements on this paper represent a good starting point for system where performing a BDI model from the beginning falls in the complex or chaos domain, thus helping us to deal with complexity.
- ***A.15 The modelling artifacts, techniques, and guidelines presented allow us to model abstractedly and graphically electronic institutions, and it enables the construction of a methodology for dealing with more complex institutions***
  The electronic institutions approach model a MAS basing on social concepts [12]. The main concepts used in this approach present a direct correlation with elements presented in this document. Following, we summarise how the elements presented in this document can help the development of first phases of electronic institutions:
  Electronic institution separate *roles and agents* as we show in this document.
  Dialogical frameworks defines the interaction framework for a set of agents. With this purpose are defined the ontology that defines the common language that agents use to interact, and the common language used for communication. It also defines *illocution schemes*. An illocution scheme represents

an interaction among two roles which represents statically an acquaintance relationship between roles in the institution. We argue for modelling interactions in different layers of abstraction. Illocution schemas can be graphically represented using role models with a single mRIs and its ontologies since it represents the static relationships that may appear between roles. We can also use role models with several mRIs and its corresponding ontologies, where each mRI represents an illocution scheme. Notice that we are able to model constraints on the use of illocution schemes using guards of mRIs and that we can also use parameterised role models when a higher level of reuse is needed.
A *scene* defines how a set of roles interact in an institution describing the available order of use of illocutionary particles over time. That is to say, scenes represents the dynamic aspect of acquaintance relationships. In our approach, both behaviour descriptions, behaviour model of a role model and of a role, can be used to describe scenes gaining all the advantages described for them.
The *performative structure* represents the next–level–of–abstraction description of the behaviour of the institution regarding scenes. It represents how the scenes defined in the institution appear over time. This represent a good abstraction mechanisms that, as authors argue, is required for developing large institutions. In our approach, we also base on this solution but extending it to an unlimited number of level of abstractions. We also provide the techniques and guidelines to maintaining them and we use the same semantics for all layers.
Normative rules specifies the obligations that an agent acquires in a certain scene because of its participation in another scene. They can be modelled by instantiation rules, guard and behaviour models that limit the available set of interactions for a role in a certain execution time. Postconditions of interactions can show how each role changes because of executing the interaction.
Finally, it must be emphasised that our approach does not replace electronic institutions, but it helps to perform the first stages of modelling of complex institutions where a low knowledge of its intricacies is available. Later, when a initial layered model is developed following the recommendations in this document, more formal models can be produced following the electronic institution approach.

7.3 Decomposition Guidelines

- ***D.22 decomposition of problems allows us to assign isolated sub–problems to different teams of analysts.***
  It help us to deal with complexity since each team can focus on a simplified problem without taking into account restrictions of related parts of the system.
- ***D.23 to perform a decomposition maintaining the level of abstraction we must use role model decomposition.***

The former technique helps us to perform a functional decomposition of the system where we maintain the level of abstraction.

- ***D.24 When we need to refine models we must use interaction decomposition.***
  The later helps us to refine models defining abstract interactions internally (by means of finer–grain interactions) thus obtaining a new layer with a refined model.
- ***D.25 Interactions must be linked with requirement system goals to provide traceability between requirements and analysis and to improve system congruence.***
  This process also improves the system *Congruence* since system goals are taken into account and agents goals can be determined by a decomposition of a system goal into a set of sub–goals assigned to agents (notice that such decomposition may have been identified at requirement stage).
  Using system goal diagrams, we can determine interactions in the system. A goal results on an interaction between several agents when it is enough complex to require more than one agent to be achieved. This argues for linking interactions with goals providing a direct transition between requirements and analysis. Goals that are enough simple can be linked with agents.
- ***D.26 Hierarchical goal diagrams guide the decomposition. We can apply "role model decomposition" or "interaction decomposition".***
  Hierarchical goal diagrams provide a description of system goals where we can see which sub–goals are needed to fulfill a higher level system goal. This information can be used to perform both kinds of decomposition as it is described in D.13.
- ***D.27 Dependency Decomposition must be applied when we are not able to perform more goal decompositions.***
  Finer system goals identified at requirement stage may result insufficient to reach a level of detail enough to step to design easily. Notice that in complex system, we should minimize the complexity of interactions at analysis as most as possible. If interactions are too much complex at design it will be harder to design them internally. Dependency decomposition can be applied at the interaction level, thus, performing *interaction decomposition*.

7.4 Composition guidelines

- ***C.7 Composition help us to discover how sub–parts of the system modelled by different teams of analysts relates.***
  It help us to deal with complexity since modelling composing several sub–parts of the system studied separately only implies to model the relationships between models that have been studied and understood properly. Thus, we do not have to deal with the whole problem at once.
- ***C.8 To identify instantiation rules that involves several models we must compose them***
  Furthermore, building a new role model from several isolated ones allows us to identify new constraints on the agents who may play each of them. Thus,

composition of role models is a crucial tool to identify instantiation rules that traverse the frontier of one role model as we show in previous section: a paper's reviewer must be different from the paper's author but this instantiation rule does not belong to the reviewing process nor the submission process, but to both.

- ***C.9 To identify coordinator agents we must use composition***
  When several role models are merged we can obtain a complex organisation. In these situations, coordinators can be added to control the sub–organisation.
- ***C.10 Composition and Decomposition must be documented in traceability models by associations recommended.***
  Since interactions and their elements can be composed, we must use "aggregation" association to document it in traceability models. If redefinitions are performed, we must use "redefinition" association.
- ***C.11 Behaviours composition must be based on multiparty interactions of a certain granularity when role interleaving is needed***
  Notice that if we have to interleave several behaviour descriptions (behaviours of a role model or behaviours of a role), we have to do it at the proper interaction granularity. For example, in a computer science department a role professor has to be interleaved with a role head of department. In this situation high level interactions could not be useful. For example, if we model the behaviour of each role by a loop executing a single abstract interaction, *manage_department* for *head* and *teach* for *professor*, we are only able to produce a behaviour description where both roles are alternated. However, we are able to perform a higher interleaved behaviour if we refine both behaviour models of roles using finer–grain interactions.
  As a matter of fact, when specifying behaviours, we have to determine the granularity of interactions to be used in the model.
- ***C.12 Behaviour composition must be done over behaviour model of a role model if most roles have to be composed.***
  When most behaviour of roles has to be composed or their behaviour is affected by new interactions we recommend merging the behaviour model of role models. If most roles in the system change, it is easier to understand these changes in a centralised description than in a distributed description. For example, consider a role model to airline booking and another to calculate expenses which have to be merged to provide information on the prices and where every behaviour model of role changes. We can compose the behaviour model of each role model to obtain the composite behaviour model of both (see C.1 and D.12).
  Later, if we are interested on the behaviour of one of the composed roles, we can extract it by techniques in point D.12.
- ***C.13 Behaviour composition must be done over behaviour of roles if few roles have to be composed and most remain unchanged.***
  When most roles remain unchanged, it is easier to compose only affected roles since they represent a partial model of the system. Thus, we have to consider only such interactions where affected roles are involved (not all).

For example, if we have to compose a *search_book* role model with a *purchase_books* role model, we have only to compose the role *searcher* in the former with the role *shopper* in the later. Merging the behaviour of these roles will be easier than building a new behaviour model of the role model since these roles are involved in a sub–group of the interactions that we should manage when dealing with the behaviour of the role model.
Later we can transform the affected roles behaviour along with the roles unchanged to obtain the behaviour model of the composite role model by techniques in point C.1 and D.12.

7.5 Guidelines for Improving Reuse at Analysis and Runtime: Open Systems

- ***D.28 To improve reusability and decrease complexity of decomposed models they must be modelled without taking into account details on other problems***
  It is important to point out that some features are lost when problems are studied isolated. Some interactions may be lost. For example, if a MAS to search and purchase items is decomposed into a search model and a purchase model, the way in how the search is restricted to such dealers that admit the credit card of the user do not belong to the scope of any problem, but both. Notice that problems can be modelled keeping in mind constraints with other problems or interactions. However, if we want to reuse a certain acquaintance model it should be as general as possible. If we decompose models isolating them from other models constraints, we produce more general models. This promotes reuse since descriptions do no model dependencies with other problems in the system and can be reused in the same system or others adding specific dependencies. Furthermore, ignoring interdependencies with other models eases the modelling of the problem, thus, decreasing complexity.
- ***D.29 Instantiation rules allows to model open system where interaction pattern are known at analysis but not concrete agents who participate on them***
  Instantiation rules of role models can be used as we do in Object Oriented paradigm when instantiating an object through a parameterized constructor. This shows a correlation between Classes and Role Models and between Object and Agent Models. Therefore, the parameterisation of role models easy the reuse of them and even the instantiation at runtime. It allows to model open systems where role models are instantiated dynamically at runtime as we do when creating a new object at runtime in the OO paradigm.
- ***D.30 Instantiation rules plus guards allows to model complex non predictable systems where interaction patterns are known at design time.***
  Instantiations rules can be used to model the organisation rules, and guards can be used to represent when agents are interested on performing a certain action with other agents. In this way, we can model rules and guards basing on BDI, ecosystems or electronic institutions.

This allows us to model BDI (Joint intention framework), electronic institutions, and so on, using the same elements proposed previously. This also allows applying the rest of techniques and guidelines in this document, but not behaviour descriptions based on interactions (see Section 8).

– ***D.31 "Generalisation" association and behaviour inheritance technique provide means for modelling open system in a very flexible way***
We can define generalised roles in the system which can be extended by inheritance (interface inheritance and behaviour inheritance). This allows that unknown agents enter in the system if they play a role that extends some of the roles defined in the system.

8 Conclusions

In this RFC, we have attempted to present the modelling artifacts, techniques, and guidelines needed to deal with complexity in the AOSE field. We hope to have provided enough insight so that readers can decide which of them they see appropriate so that the comments provided by readers can be studied later to determine how current methodologies can be improved in the interaction plane to deal with more complex systems than currently.

9 Acknowledgment

We would like to thank James Odell for reviewing this document and for the fruitful discussion we have maintain with him and the improvements made as a result of it. We also have to thanks Massimo Conssentino, Renato Levy, Dominic Greenwood, Juan Pavón, Juan Carlos Moreno, Antonio Moreno, Josep Lluis Arcos and the rest of people we have contacted for their accurate comments and interest.

10 Glossary

We define here the main concepts used in this document:

Abstract joint behaviour: Describes the order of execution of joint actions (*multiparty interactions*) that a *role* performs over time

Acquaintance organisation: Organisation from the interaction point of view. It is formed as a set of *role models*

Behaviour Model of a Role: Model that represents how the set of *multiparty interactions* in where a role is involved can be ordered over time.

Behaviour model of a role model: A model that represents how the set of interactions in a *role model* sequence

Behaviour: this term is used with a different meaning than in most methodologies. Most approaches see behaviour as a design term where implementation details are involved. In this paper, behaviour is a synonym of *abstract joint behaviour*

Dynamic aspect: involve models that show time–dependent features. For example, the order of apparition of *multiparty interaction relationships* over time

Interaction granularity Abstraction level of an interaction. Finer grain interactions represent simple joint processes, while coarse–grain interactions represent abstractly complex joint process

Interaction: This term is used as a synonym of *multiparty interaction relationship* and has a entirely different meaning than in current research.

Modelling Artifact: Graphical representation of a certain artifact of the system in a model

Multiparty interaction relationship: Relationship between an arbitrary number of roles established to fulfil some system goal defined abstractly. It comprises all interactions[3] and tasks performed to fulfil the system goal it pursues. In this paper, we refer them as *interactions* for shortening

Role Goal: A goal that is pursues by a single agent playing a role in the context of a certain multiparty interaction

Role Model: A set of roles related by means of multiparty interactions that jointly fulfill a system goal by contention or cooperation[4]

Role: A role is partial view of an agents which represent its features regarding a certain *multiparty interaction/interactions*. A role defines also the interface that the agent offers to the rest of participant by means of services and knowledge and pursues a *role goals*, one per multiparty interaction in where it is involved

Static aspect: Set of models that show time–independent features. For example, the acquaintance relationships between *roles* in a *role model* or their interfaces.

Structural organisation: it represents structural relations between agents grouping them in departments, teams, relating them by subordination relationships, etcetera

System Goal: A goal enough complex for requiring several agents to be achieved at a certain level of abstraction

Technique: Systematic procedure that is used to transform, calculate a piece of a model or a complete new model, or to proof or check properties of models

References

1. R. Bagrodia. Synchronization of asynchronous processes in CSP. *Transactions on Programming Languages and Systems*, 11(4):585–597, October 1989.
2. B. Bauer, J. Mller, and J. Odell. Agent uml: A formalism for specifying multiagent interaction. In *Proceedings of the First International Workshop on Agent-Oriented Software Engineering AOSE 2000*, number 1957 in LCNS, Limerick, Ireland, 2001. Springer–Verlag.

[3] as defined in FIPA Methodology TC glossary
[4] Following the definition on [29]

3. B. Bauer, J. Muller, and J. Odell. Agent uml: A formalism for specifying multiagent interaction. In M. Wooldridge and P. Ciancarini, editors, *Proceedings of 22nd International Conference on Software Engineering (ISCE)*, LNCS, pages 91–103, Berlin, 2001. Springer–Verlag.
4. G. Booch. *Object-Oriented Design with Applications*. Benjamin/Cummings, Redwood City, CA, 1990.
5. P. Burrafato and M. Cossentino. Designing a multi-agent solution for a bookstore with the passi methodology. In *Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002). CAiSE'02*, Toronto, Ontario, May 2002.
6. G. Caire, F. Leal, P. Chainho, R. Evans, F. Garijo, J. Gomez, J. Pavon, P. Kearney, J. Stark, and P. Massonet. Agent oriented analysis using MESSAGE/UML. In *Proceedings of Agent-Oriented Software Engineering (AOSE'01)*, pages 101–108, Montreal, 2001.
7. D. de Champeaux. Object-oriented analysis and top-down software development. In *Proceedings of the European Conference on Object-Oriented Programming, ECOOP'91*, volume 512 of *Lecture Notes in Computer Science*, pages 360–375. Springer–Verlag, 1991.
8. P. Coad and E. Yourdon. *Object-Oriented Analysis*. Computing Series. Yourdon Press, Englewood Cliffs, NJ, 1990.
9. A. Dardenne, A. van Lamsweerde, and S.Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3–50, 1993.
10. S. A. DeLoach, M. F. Wood, and C. H. Sparkman. Multiagent systems engineering. *The International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001. World Scientific Publishing Company.
11. D'Souza and A.C. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison–Wesley, Reading, Mass., 1999.
12. M. Esteva, J. A. Rodriguez-Aguilar, C. Sierra, P. Garcia, and J. L. Arcos. On the formal specification of electronic institutions. In F. Dignum and C. Sierra, editors, *Agent–mediated Electronic Commerce (The European AgentLink Perspective)*, number 1991 in LNAI, pages 126–147. Springer–Verlag, 2001.
13. N. Francez and I. Forman. Synchrony loosening transformations for interacting processes. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of Concurr'91: Theories of concurrency – Unification and extension*, number 527 in LNCS, pages 27–30, Amsterdam, The Netherlands, August 1991. Springer–Verlag.
14. N. Francez and I. Forman. *Interacting processes: A multiparty approach to coordinated distributed programming*. Addison–Wesley, 1996.
15. N. Francez and I. R. Forman. *Interacting Processes*. Addison–Wesley, 1996.
16. Mike Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Mike Wooldridge. The belief-desire-intention model of agency. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 1–10. Springer-Verlag: Heidelberg, Germany, 1999.
17. A. Giret and V. Botti. Towards an abstract recursive agent. *Integrated Computer–Aided Engineering*, 11(2), 2004.
18. N. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.
19. N. R. Jennings. Specification and implementation of a belief-desire-joint-intention architecture for collaborative problem solving. *Intl. Journal of Intelligent and Cooperative Information Systems*, 2(3):289–318, 1993.

20. E. Kendall, U. Palanivelan, and S. Kalikivayi. Capturing and structuring goals: Analysis patterns. In *Proceedings of the* 3^{rd} *European Conference on Pattern Languages of Programming and Computing*, Germany, July 1998.
21. E. A. Kendall. Role modeling for agent system analysis, design, and implementation. *IEEE Concurrency*, 8(2):34–41, April/June 2000.
22. D. Kinny, M. Georgeff, and A. Rao. A methodology and modelling technique for systems of BDI agents. In Rudy van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.
23. J. Koning, M.Huget, J. Wei, and X. Wang. Extended modeling languages for interaction protocol design. In M. Wooldridge, P. Ciancarini, and G. Weiss, editors, *Proceedings of Second Internationa Workshop on Agent-Oriented Software Engineering (AOSE'02)*, LNCS, Montreal, Canada, May, 2001. Springer–Verlag.
24. B. Liskov and J. M. Wing. Specifications and their use in defining subtypes. In *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, pages 16–28. ACM Press, 1993.
25. H. Mintzberg. *The Structuring of Organizations*. Prentice-Hall, 1978.
26. J. Odell. Agents and complex systems. *Journal of Object Technology*, 1(2):35–45, July-August 2002.
27. Object Management Group (OMG). Unified modeling language: Superstructure. version 2.0. Final adopted specification ptc/03–08–02, OMG, August 2003. www.omg.org.
28. H. Van Dyke Parunak and James Odell. Representing social structures in UML. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 100–101, Montreal, Canada, 2001. ACM Press.
29. H. V.n D. Parunak, S. Brueckner, M. Fleischer, and J. Odell. A design taxonomy of multi-agent interactions. In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, *IV International Workshop on Agent-Oriented Software Engineering (AOSE'03)*, volume 2935 of *LNCS*, pages 123–137. Springer–Verlag, 2003.
30. J. Peña, R. Corchuelo, and J. L. Arjona. A top down approach for mas protocol descriptions. In *ACM Symposium on Applied Computing SAC'03*, pages 45–49, Melbourne, Florida, USA, 2003. ACM Press.
31. J. Peña, R. Corchuelo, A. Ruiz-Cortés, and M. Toro. Towards an automatic method for detecting synchrony loosening anomalies in the context of multiparty interactions. In *Actas del II taller de trabajo sobre Desarrollo de Software Preciso. VI Jornadas de Ingeniería del Software y Bases de Datos (JISBD'01)*, Almagro (Ciudad Real, Spain), November 2001.
32. J. Peña, R. Corchuelo, and J. L. Arjona. Towards Interaction Protocol Operations for Large Multi-agent Systems. In M. Hinchey, J. Rash, W. Truszkowski, C. Rouff, and D. Gordon-Spears, editors, *Proceedings of the Second International Workshop on Formal Approaches to Agent-Based Systems (FAABS 2002)*, volume 2699 of *LNAI*, pages 79–91, NASA-Goddard Space Flight Center, Greenbelt, MD, USA, 2002. Springer–Verlag.
33. T. Reenskaug. *Working with Objects: The OOram Software Engineering Method*. Manning Publications, 1996.
34. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object–Oriented Modeling and Design*. Prentice Hall, Schenectady, New York, 1991.
35. D. Snowden and C. Kurtz. The new dynamics of strategy: Sense-making in a complex and complicated world. *IBM Systems Journal*, 42(3):35–45, 2003.

36. C. Szyperski, D. Gruntz, and S. Murer. *Component Software: Beyond Object-Oriented Programming*. Addison–Wesley, second edition edition, 2002.
37. R. Wirfs-Brock and A. McKean. *Object-Oriented Design: Roles, Responsibilities, and Collaborations*. Addison–Wesley, November 1990.
38. R. Wirfs-Brock and B. Wilkerson. *Designing Object-Oriented Software*. Prentice–Hall, August 1990.
39. F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the GAIA methodology. *ACM Transactions on Software Engineering and Methodology*, to be published 2003/2004.