A Framework for the Definition of Metrics for Actor-Dependency Models

Xavier Franch, Gemma Grau, Carme Quer
Universitat Politècnica de Catalunya (UPC)
C/Jordi Girona 1-3, UPC-Campus Nord (C6), Barcelona (Spain)
{franch, ggrau, cquer}@lsi.upc.es
http://www.lsi.upc.es/~gessi/

Abstract

Actor-dependency models are a formalism aimed at providing intentional descriptions of processes as a network of dependency relationships among actors. This kind of models is currently widely used in the early phase of requirements engineering as well as in other contexts such as organizational analysis and business process reengineering. In this paper, we are interested in the definition of a framework for the formulation of metrics over these models. These metrics are used to analyse the models with respect to some properties that are interesting for the system being modelled, such as security, efficiency or accuracy. The metrics are defined in terms of the actors and dependencies of the model. We distinguish three different kinds of metrics that are formally defined, and then we apply the framework at two different layers of a meeting scheduler system.

1. Introduction

Goal-oriented analysis methods and languages such as KAOS, *i**, GRL or TROPOS [1, 2, 3] are widespread in the requirements engineering community as cornerstones for the refinement and decomposition of the customer needs into concrete goals in the early phase of the requirements specification [4].

Goal-oriented models define a network of actors and dependencies and their decomposition into simpler concepts. In this paper, we are especially interested in the consideration of goal-oriented models without analysing decomposition aspects. In other words, we focus on the static nature of a goal-model, showing the actors and dependencies that exist at one layer of the system. We call *actor-dependency models* this restricted class of goal-oriented models. An example of actor-dependency models would be *i** Strategic Dependency (SD) models [2], whilst *i** Strategic Rationale (SR) models are not objective of our current work.

Actor-dependency models are very interesting by themselves. In the context of requirements engineering, they are the formalism to express a first, organizational view of the system. Once the actor-dependency model exists, it can be used in different ways. For instance, we may analyse the model itself to reason about the modelled world, focusing on properties such as opportunity and vulnerability [2] to explore different alternatives. When the actor-dependency model is considered complete it may be used in further activities usual in goal-oriented modelling: we may undertake a deeper analysis of its actors by making explicit their rationale and form a more complete goal-oriented model by decomposition; or we may use the model to formulate the prescriptive system requirements using classical artefacts such as use cases [5].

In this paper we are interested in the structural analysis of actor-dependency models. More precisely, we want to take profit of the structure of actor-dependency models to analyse the modelled system with respect some designated properties of interest using some adequate metrics.

The use of metrics for analysing the adequacy of proposed system models is well-known in other type of models. For instance, there are some suites of metrics in the field of object-oriented models [6, 7], which refer to structural properties like cohesion and coupling. Properties referring to the system itself, such as security, efficiency or cost, which mainly fall in the category of non-functional or organizational requirements, appear when considering models of the system architecture [8]. These metrics are usually defined in terms of the components, nodes, connections, pipes, etc., that compose the final configuration of the system.

In this paper, we propose the use of metrics defined over actor-dependency models to analyse non-functional and organizational goals. As a result, we obtain metrics that are closer to the universe of discourse than to the internal structure of the system, since their definition will be given not in terms of

connectors and pipes but of actors and dependencies. We articulate our proposal as a general framework for defining these kind of metrics. We introduce the framework in a formal way, first characterising the concept of actor-dependency model and then proposing three different categories of metrics. All three are built upon the same idea, namely the use of functions that give weights to actors or dependencies of the model; the weight may be given individually to each element, or else we may give the same weight to elements of the same kind or exhibiting a particular property. Depending on the element of interest, i.e. actors or dependencies, we obtain actor-dependency or dependency-based general forms of metrics.

The paper is structured as follows. In section 2, we introduce our notion of actor-dependency models and provide formal definitions for them and their components. In section 3, we identify three categories of metrics and we give general forms for each of them. In sections 4 to 6, we study the applicability of the framework in a particular, well-known case study, a meeting scheduler [4, 9]. Last, in section 7 we provide the conclusions and some future work. Throughout the paper, we use *i** strategic dependency (SD) models [2] to provide examples and draw system models.

2. A Definition of Actor-dependency Models

An *actor-dependency model* comprises two types of elements, the *actors* themselves and the *dependencies* among them. Actors are intentional entities, that is, there is a rationale behind the activities that they carry out. Dependencies connect source and target actors, called *depender* and *dependee* respectively. Altogether form a network of knowledge that allows understanding "why" the system behaves in a particular way [10].

In this paper, we consider two types of actors, namely roles and agents. According to [2], we define a *role* as an abstract characterization of the behaviour of a social actor within some specialized context or domain of endeavour, whilst an *agent* is an actor with concrete, physical manifestations that plays a role.

Actors and dependencies may exhibit *attributes*[1], such as product fabricant, physical location, etc., or others more oriented to their use during the requirements engineering activity, as priority or importance.

For our purposes, it is helpful to consider that agents and dependencies belong to one *sort* that group elements of the same kind; therefore we can talk about human agents, goal dependencies, and so on. Concerning roles, we are more flexible and we allow them to belong to zero, one or more sorts. This rule fits with the case that although some roles are inherently of a given sort, others are more open; for example, a meeting scheduler role may be covered by human or software agents and therefore we classify it as belonging to the human and software roles as well.

Definition 1 formalizes the concept of actor-dependency model. To simplify matters, we do not consider in the rest of the paper the case of dependencies with multiple dependers or dependees, whose treatment in our context is straightforward but technically cumbersome.

Definition 1. Actor-dependency model.

An actor-dependency model is a pair $M = (A, D)$, being A a set of actors and D the dependencies among them, such that:

1) The set A has a mapping $type_A$: $A \rightarrow$ {role, agent} that classifies model actors into roles and agents. When convenient, we consider A as a pair $A = (A_{\text{role}}, A_{\text{agent}})$.

2) The set A has a mapping $sort_A$: $A \rightarrow \text{P}(TA)$, being TA the permissible sorts of actors. It holds that:

$$type_A(a) = \text{agent} \Rightarrow \| sort_A(a) \| = 1$$

3) The actors in the set A may have attributes modelled as mappings $\{ prop_{A,i}: A \times K_i \rightarrow V_i \}$.

4) D is defined as a set of ordered pairs of actors with the name of the dependency, $D \subseteq A \times A \times string$.

5) The set D has a mapping $sort_D$: $D \rightarrow TD$, being TD the permissible sorts of dependencies.

6) The dependencies in the set D may have attributes modelled as mappings $\{ prop_{D,i}: D \times K_i \rightarrow V_i \}$.

Fig. 1 presents an example of actor-dependency model expressed with the *i** notation, namely a Strategic Dependency (SD) model, for a meeting scheduler system in the ACME company, based on that of [2]. There are a few changes; remarkably, we introduce an actor that plays the part of directory. The 3 actors are roles; although not in the example, other types of actors, namely agents and an intermediate concept called *position*, are supported in *i**. About dependencies, SD models allow 4 sorts:

- *Goal*. The depender depends upon the dependee to bring about a certain state in the world. See *Attend Meeting* in the figure.
- *Task*. The depender depends upon the dependee to attain a goal in a particular way. We have not included any in the example.
- *Resource*. The depender depends upon the dependee for the availability of a physical or informational entity (see *Agreement Date* in fig. 1).

[1] Probably "property" would have been a better term than "attribute", but we are using "property" throughout the paper with another meaning, then we have preferred to avoid confusion.

- *Soft goal*. The depender depends upon the dependee to meet some non-functional requirement. We have not included any in the example.

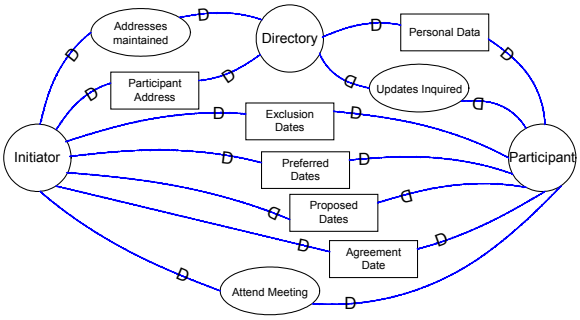

Figure 1. An *i** SD model for a meeting scheduler.

Actor-dependency models can be refined from the starting system model passing through different refinement steps, each intermediate model capturing a notion in the requirements engineering activity. We can say that actors from one model are assigned to actors from the other model, and the dependencies go with them. Eventually, we could reach an state where all the actors are agents, representing how physical entities play the system roles. Many times we do not need to reach such a level of detail, but just two or three different layers of vision of the system. In this paper, we will address 2 different layers of interest in our case study, representative enough of our vision of actor-dependency models application and well-suited to apply our metrics framework.

3. Metrics for Actor-Dependency Models

In this section, we propose the use of *structural metrics* for analysing the *model properties* of an actor-dependency model, i.e., those properties that depend on the form of the model and the types, sorts and attributes of its elements. Structural metrics are valuable for both analysing a highly abstract model of a system of any kind, composed basically by roles, and for comparing different feasible realizations of this abstract model (which take the form of actor models too, but composed basically by agents) with respect to the most relevant criteria established in the modelled world.

For a given model property object of measure, it may be the case that all its elements (actors and dependencies) influence the metric. However, it is more likely that just elements of some particular sorts affect this property. Furthermore, some individual elements may be identified as especially relevant for the property; in the most general case, all the elements may have a different weight in the metric. We need then to take into account all these situations if we aim at having a widely applicable metrics formulation framework.

We distinguish three types of structural metrics. *Global structural metrics* take the model as a whole and produce a single measure for the property of interest. *Local structural metrics* focus on the individual elements of the model, producing then a set of values that can be examined looking for thresholds or weak points of the model. On top of local structural metrics we define *sensitivity metrics*, finding out the element that maximizes the values of a local structural metric. All types of metrics rely on two fundamental concepts, *actor evaluation* and *dependency evaluation*.

3.1. Evaluation of actors and dependencies

The atomic concept in our metrics framework is the evaluation of the individual elements that are in the actor-dependency model. The evaluation of actors and dependencies are defined as functions that yield a value in the interval [0, 1]. In both cases, the evaluation is computed as the multiplication of two factors, the first one taking into account just the type, sorts and attributes of the element itself, and the second one considering the other model elements it relates to, i.e. the dependencies stemming from, or going onto, the actor in actor evaluation; or the actors linked to the dependency in dependency evaluation.

Definition 5. Actor evaluation.

Given a model property P, an actor-dependency model $M = (A, D)$ and an actor inside the model, $a \in A$, the actor evaluation of a for P over M is of the form:

$$P_{M,A}(a) = f_A(a) \times g_{A,D}(a))$$

being f_A: $A \rightarrow [0, 1]$ a mapping that assigns a weight to every actor of the model, and $g_{A,D}$: $A \times D \rightarrow [0, 1]$ a mapping that corrects the weight of an actor considering the dependencies stemming from or going onto it.

Definition 6. Dependency evaluation.

Given a model property P, an actor-dependency model $M = (A, D)$ and a dependency inside the model, $d=(a, b, x) \in D$, the dependency evaluation of d for P over M is of the form:

$$P_{M,D}(d) = f_D(d) \times g_{D,A}(d)$$

being f_D: $D \rightarrow [0, 1]$ a mapping that assigns a weight to every dependency of the model, and $g_{D,A}$: $D \rightarrow [0, 1]$ a mapping that corrects the weight of a dependency considering the depender and dependee actors, respectively.

3.2. Global Structural Metrics

Definitions 7 and 8 provide a generic framework for the definition of global structural metrics. There we define two different types of global structural metrics, actor-dependency and dependency-based, depending on which type of element the metric focuses on. The metrics just sum the evaluations of its elements, and make a final normalization of the value taking into account the number of actors or dependencies of the system that satisfy a particular condition. Sometimes the metric value must be considered as-is without any normalization and then the $limit_P$ function yields 1.

Definition 7. Actor-dependency global structural metrics.

Given a model property P, an actor-dependency model $M = (A, D)$ and a function $limit_P$: $A \rightarrow [1, ||A||]$, an actor-dependency global structural metric for P over M is of the form:

$$P_M = \frac{\Sigma a \in A: \mathrm{P}_{M,A}(a)}{\mathrm{limit}_P(A)}$$

Definition 8. Dependency-based global structural metrics.

Given a model property P, an actor-dependency model $M = (A, D)$ and a function $limit_P$: $D \rightarrow [1, ||D||]$, a dependency-based global structural metric for P over M is of the form:

$$P_M = \frac{\Sigma d \in D: \mathrm{P}_{M,D}(d)}{\mathrm{limit}_P(D)}$$

3.3. Local and Sensitivity Structural Metrics

The definition of local structural metrics follows the same layout than global ones but keeping track of the measures of individual elements. We use this kind of metrics for having a complete analysis of the individual actors or dependencies of an actor-dependency model. Sensitivity metrics are a kind of summary providing the maximum value of a local structural metrics.

Definition 9. Actor-dependency local and sensitivity structural metrics.

Given a model property P, an actor-dependency model $M = (A, D)$ and a function $limit_P$: $A \rightarrow [1, ||A||]$, an actor-dependency local structural metric for P over M is of the form:

$$P_M: A \rightarrow [0, 1] \text{ such that } \mathrm{P}_M(a) = \mathrm{P}_{M,A}(a)$$

We define the actor-based sensitivity structural metrics bound to P as:

$$PMax_M = \max a \in A: \mathrm{P}_M(a)$$

Definition 10. Dependency-based local and sensitivity structural metrics.

Given a model property P, an actor-dependency model $M = (A, D)$ and a function $limit_P$: $D \rightarrow [1, ||D||]$, a dependency-based local structural metric for P over M is of the form:

$$P_M: D \rightarrow [0, 1] \text{ such that } \mathrm{P}_M(d) = \mathrm{P}_{M,D}(d)$$

We define the dependency-based sensitivity structural metrics bound to P as:

$$PMax_M = \max d \in D: \mathrm{P}_M(d)$$

4. The Meeting Scheduler Case Study

In the rest of the paper, we are going to present some examples of application of structural metrics for actor-dependency model properties in different contexts. We use the meeting scheduler system as example, and we study 2 different contexts of applications in the next following 2 sections: first, deciding whether a software meeting scheduler is adequate with respect to some non-functional requirements, expressed as model properties; once the convenience of a software system has been established, selecting the most convenient combination of software packages for implementing this system as COTS-based. This example is representative of the kind of uses that we envisage for structural metrics. The metrics defined will be of different kinds: actor- and dependency-based; global and sensitivity; using properties or not; with special cases or just taking into account sorts and types.

5. Studying the Adequacy of a Software System for Scheduling Meetings

In fig. 1, we have introduced as example an organizational environment for a meeting scheduler in the ACME organization. Let's assume now that ACME has found that the meeting scheduling process is currently not satisfactory enough: too often, people do not get aware of meetings or misunderstand their dates; from time to time, people seems to manage information, such as email addresses, with less privacy than required; the meeting date determination is not as agile as desired; etc. Therefore, the ACME's executive board has decided to analyse other possible strategies. In particular, it has decided to include the figure of a meeting scheduler such that the meeting initiator may delegate most of the duties. It becomes necessary to compare the behaviour of such an organizational

alternative to the existing one. In particular, we explore two possibilities about the meeting scheduler: to be a human role, carried out by some administrative figure, or to be a software system.

5.1. A new Organization Alternative for Scheduling Meetings

In fig. 2 we show the actor-dependency model that represents the new organizational system for scheduling meetings. There are some differences with respect to Yu's [2] (apart from the directory actor and its dependencies) because we have preferred to obtain a model as similar as possible to the previous one, to support easy comparison using the metrics.

We keep the 3 former actors and add a new one for the meeting scheduler. The dependencies from/to the *Directory* actor are similar, except for the change of the *Participant Address* dependency to *Participant Name* (i.e., the directory provides help to find participant names if necessary). The *Meeting Scheduler* takes the responsibility of coordinating with participants; it just needs the range of dates from the *Meeting Initiator* to start the process. We add a goal dependency from the *Meeting Initiator* to the *Meeting Scheduler*, and we keep the one from the *Meeting Initiator* to the *Meeting Participant*. Last, we consider that the *Meeting Scheduler* obtains the participants' names from the *Meeting Initiator* and then the corresponding email addresses from the *Directory*.

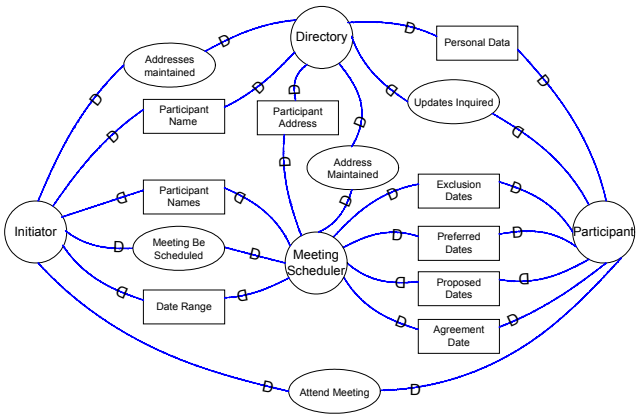

Figure 2. A *i** SD model for a meeting scheduler system with explicit meeting scheduler actor.

5.2. Non-Functional System Requirements

Next we enumerate the non-functional requirements that catch the problems that the previous organization suffered from. For each requirement, we describe how it can be related to the different parts of the actor-dependency model considering the sorts of actors and dependencies. Concerning the types, the statements given below drives us to consider just resource dependencies in the first three cases.

- The system shall respect privacy. Communication of information among actors is a risk concerning privacy. We consider that human actors introduce an hazard when communicating data. Furthermore, we consider more private software actors than human ones, since one of the problems of the former meeting scheduling process was lack of privacy caused by careless information management.
- The system shall be accurate. This is the most important requirement of the system. Of course, communication of information among software actors is totally accurate; on the contrary, accuracy gets damaged when people is involved. We also consider that, once transmitted, software keeps the information more accurately than human, who may introduce errors when e.g. registering the meeting in their agenda.
- The system shall be efficient. This means that the process itself is required to be agile, minimizing process of information. Needless to say, software actors and dependencies makes efficiency better.
- The system shall be fault tolerant. In our system, we try to avoid actors that are responsible of too many things to minimize collapses when the actor is unavailable. Responsibilities take the form of incoming dependencies, i.e. dependencies in which the actor plays the part of dependee.

5.3. Definition of the Structural Metrics

Given these descriptions, we define one model property bound to each requirement and for each property, we introduce a single structural metric: data privacy, data accuracy, process agility and responsibility dissolution, respectively. A more complete analysis could introduce other metrics bound to the properties, and then some technique for combining the value of all the metrics of one property into a single one should be applied, using some kind of weighting factor matrix as proposed by McCall in a context similar to ours [11].

Since the flow of data is a crucial factor in the first three cases, we define the metrics as global and dependency-based. Table 1 shows the corresponding dependency evaluation. We define $g_{D,A}(d)$ as a function h on the sort of the actor, $g_{D,A}((a, b, x)) = h_A(a) \times h_A(b)$ and show this factor at the table. We consider the four sorts of dependencies given by *i** (represented by their capital letter: G, SG, T, R), two sorts of actors, human

(H) and software (S) and three derived types of dependencies, human-human (H-H), software-software (S-S) and human-software (S-H). Dependency evaluation depends only on this information. The values assigned range in [0, 1]; we assign 1 to the best (accuracy) or worst (privacy and agility) possible case and then the rest of values try to measure the deviation that the sorts of actors and dependencies and the type of dependencies provoke in the corresponding evaluation. We may use weighting techniques such as the AHP [12] to compute the values in a more confident way.

	Dependency		Attribute		
	Sort	Type	DP	DA	PA
f_D	G	any	0	0	0
	SG	any	0	0	0
	T	any	0	0	0
	R	H-H	1	0,6	1
		H-S	0,9	0,8	0,7
		S-S	0,8	1	0,5
h_A	Sort		DP	DA	PA
	H		0,7	0,7	0,8
	S		1	1	1

Table 1. Dependency evaluation for the data privacy (DP), data accuracy (DA) and process agility (PA) metrics.

The definition of the three metrics on top of these dependency evaluations consists on giving values to the function $limit_P$. In the case of data privacy and process agility, we find convenient to use an absolute metric, i.e. we do not normalize the result, which means that the number of resource dependencies is more important than the percentage: every resource dependency endangers by itself the model property of interest. As a result, for these two metrics, we define $limit_P(D) = 1$. On the contrary, for data accuracy, we divide by the number of resource dependencies of the model, $limit_P(D) = \| \{d \in D: sort_D(d) = \text{resource}\} \|$, since the concrete number of resource dependencies is not really relevant.

Concerning the fourth metric, responsibility dissolution (RD) is better defined as a sensitivity actor-based metric. We do not assign different weights to actor sorts; we just count incoming dependencies and divide by the total number of dependencies, to make sure the [0, 1] range of values. The lesser the value is, the better the system behaves with respect this metric.

5.4. Some Alternatives for the Metrics

The proposal of metrics appearing in the previous section does not distinguish any particular actor or dependency. However, one could easily find arguments to prioritise some of the model elements according to the rationale of the properties addressed. Next we comment some examples:

- Data privacy. It is clear that the information about participants should be more protected than the information about meeting dates (in fact, it should be totally protected). So, we could assign a higher value to the evaluation of these dependencies.
- Data accuracy. We could assign a value to the agreement date resource higher than to the other date resources, since a misunderstanding concerning this resource makes the meeting fail.
- Process agility. Unlike data privacy, we may consider more important for agility the dependencies about dates than the others, since there may be some negotiation with the dates that does not exist with the participant addresses.
- Responsibility dissolution. We could consider that software actor failure is more severe because the data contained there in is not available until the system recovers, while the probability of this in human actors is smaller. This makes a great difference in case of needing some information urgently and timely.

5.5. Evaluation of the Alternatives

Table 2 provides the evaluation of the three considered organizational systems using the version of the metrics defined in section 5.3 (current system has been shown in fig. 1, proposed system in fig. 2). To make easier comparisons, we translate the results to the interval [0, 1] being 1 the best value and preserving the distances found in the measurement. Concerning the two proposed alternatives, software meeting scheduler is better than human meeting scheduler. When we compare this best new solution with the current system the situation is not so clear, but in fact we have mentioned in 5.2 that accuracy is the most important property and so the data accuracy results, in which the distance among the new solution and the current one is large, point out that the proposed system is worth.

Let's assume that these results, among others, makes the ACME company to acquire a software system for scheduling meetings. Once taken this decision, a further study could be carried out to define cost prediction metrics. For instance, we could define an equivalent to the notion of function point [13] to

make a prediction in terms of the functionalities that are to be implemented. We skip this analysis here. The next activity to be taken is thus procuring the system.

	Current system	Proposed system with human meeting scheduler	Proposed system with software meeting scheduler
DP	1	0,64	0,55
DA	0,42	0,63	1
PA	1	0,66	0,74
RD	0,64	1	1

Table 2. Evaluation of the metrics for the three alternative systems.

6. Selection of a COTS-based Solution for the Meeting Scheduler System

To carry out system procurement, we first enlarge our vision of the system. We take advantage of the current functionalities available in the market and propose a more advanced solution including anti-virus control. We also prefer to split the meeting scheduler into two actors, the scheduler itself and a message delivery service in charge of sending data to, and receiving data from, the human actors. We add also a system administrator for the meeting scheduler. The resulting *i** model is in fig. 3; we include just some *i**-like dependencies among the roles, enough for our illustrative purposes, and we also change some resources to tasks for the same reason. It can be argued that this new model should have been issued when carrying out the viability analysis of the previous section, and in fact this could be the case, but we tend to think that for the high-level organizational analysis it is better to abstract the system and focus on its basic facilities as we have done before.

At this point, we face the problem of deciding the best architecture of the system, including all the software components identified. In other words, we would like to obtain architectures as instances of the socio-technical *i** system model and compare them in the light of some architectural properties. This is the objective of the section.

6.1. Considering some COTS-based Alternatives

Following the current best practices, we choose a COTS-based solution for our system [14]. Many COTS components exist in the market playing the roles required in the model. Most meeting scheduler systems incorporate mail facilities that allow participants to communicate, whilst others offer features ranging from meeting management to the detection of conflicts that then necessitate resolution with conventional e-mail components. Furthermore, some existing e-mail components incorporate time-specific functions for managing calendars and agendas that may be used for scheduling meetings. Facilities for anti-virus protection and directory services might also be included; if not, independent components exist in the market fulfilling these functionalities. In other words, in most cases, a meeting scheduling solution is implemented as the composition of several COTS components that play the roles of the system presented in fig. 3. For this reason, a reliable COTS selection process should explore all the possible systems formed from combinations of existing COTS components.

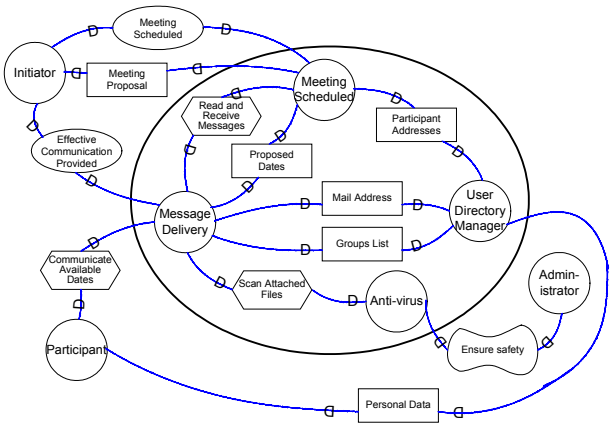

Figure 3. An actor-dependency model for the meeting scheduler socio-technical system example (excerpt)

In fig. 4 we give a partial view of three actor-dependency models for three feasible COTS-based system architectures coming from three different assignments of types of COTS components to model actors (for layout problems, we focus on the software actors and we just show the total number of dependencies among them). The architecture on the left is developed as the combination of 4 COTS components, each assigned to a single actor, whilst the one on the middle includes a meeting scheduler component that includes address lists and so the directory actor is duplicated. Finally, the one on the right provides a solution in which the meeting scheduler also deals with message delivery.

6.2. Evaluation of the COTS-based Architectures

Besides the usual compliance tests for the individual COTS components, the analysis of the software architectures that correspond to the considered combinations of COTS components may be a great help when comparing the explored alternatives.

Methodologies such as ATAM [15] perform these architectural analysis, but they tend to work on the physical layer of the architecture. In a previous paper [16], we shown a preliminary, informal work on the use of structural metrics that in fact can be seen as the motivation of the current, much more formal work.

As an example, we define next a structural metric, namely complexity of user interface, that measure the usability non-functional property [17].

Our first choice is an actor-dependency metric counting how many different COTS components interact with human actors, i.e. are dependers or dependees with some human actor (fig. 5, top and left). This is just a first approximation, easy to define and compute but not much accurate, giving better results to coarse-grained COTS-based solutions.

This first proposal may be refined by discarding those software actors whose link with human actors is based exclusively on soft goal dependencies, considering that this kind of dependencies will not affect the user interface. We use this idea to define a second kind of dependency-based metrics by taking into account goal, resource and task dependencies. Furthermore, we choose to give more weight to a goal dependency than the others, since a goal may involve (i.e., be decomposed into) some other dependencies. Concerning weights, we take the chance to prioritise types of users, for instance giving less importance to usability issues for the administrators. Fig 5, top and right, presents this option. Of course, it can be said that discarding soft goals is a very simplistic assumption, since there can be some related to usability that may have impact on the property under study; therefore, a more accurate definition should take into account the type of soft goal, using classifications such as some quality standard (e.g., [17]) and expressing them somehow in the model (for instance, using the concept of type proposed by the NFR framework [18]).

Last, we explore more precise metrics by assigning individual weights to dependencies that connect human and software actors (the others are given a zero value). These dependencies are: MS, Meeting Scheduled (goal); CAD, Communicate Available Dates (task); PD, Personal Data (resource); MP, Meeting Proposal (resource); ECP, Effective Communication Provided (goal); ES, Ensure Safety (soft-goal). We can use either quantitative or qualitative criteria for doing so. For instance, a quantitative criteria for prioritising resource dependencies is to consider the number of fields that the data screen requires to be filled, whilst task dependencies can be prioritised by the number of clicks the user is required to press in the worst case; for goals and soft goal dependencies we make some kind of estimation. In all these cases, we normalize again for obtaining values among 0 and 1, see fig. 5, bottom and left.

For qualitative analysis, we may apply criteria such as AHP [12] or laddering [19] to weight dependencies and/or actors. In fig. 5, bottom and right, we show an example of human-software dependency prioritisation using the AHP to define the weighting function. Needless to say, the convenience of such a detailed analysis must be considered.

To complete this and the previous metric we just need to define the values of $g_{A,D}(a)$ taking into account the actor assignment to concrete COTS components.

We do not include the results of the evaluation for both for space limitations and because we should consider the whole model and not the excerpt presented in fig. 3 to obtain valid data. We have done some experiments with this and other properties (such as data integrity, facility of integration, dependability, and others) with a larger model. In this particular example of complexity of user interface, we have observed that the first metric gives different results compared with the others, and this fact aligns with the small accuracy of its definition. The second metric gives the right tendency, but the differences are very small. The third and fourth metrics are similar, although the concrete values are different because of the difference of the dependency evaluation function. Of course, variations on the values used for this function may even change the ordering of the alternatives. We think that requirements priorisation existing approaches such as WinWin [20] may help in assigning the correct weights to the model elements.

6.3. Other Models to be Considered

In the previous set of metrics, we have studied the actor-dependency model resulting of instantiating the software roles of the meeting scheduler socio-technical system by other software roles corresponding to types of COTS components[2]. However other possible instantiations of the same origin actor-dependency model may be studied. For instance, we may argue that the complexity of user interface in fact depends on the variety of COTS component fabricants more than on the number of components. For instance, if both the meeting scheduler and the message delivery service are different products but from the same fabricant, they probably will use the same interface layout. Therefore, we could define an actor assignment using as assigned actors not the COTS components but their fabricants that appear in the model as attributes of the actors.

[2] In *i**, we would model the types of COTS components not as roles but as positions.

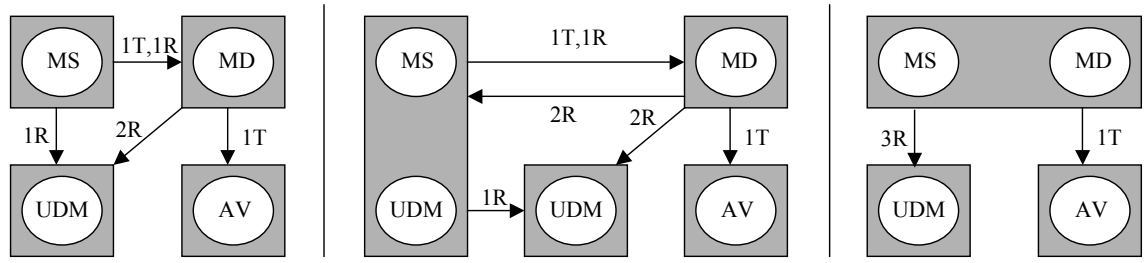

Fig. 4. Three different COTS-based architectures for the meeting scheduler system (MS, meeting scheduler; MD, message delivery; AV: anti-virus; UDM, user directory manager; G, goal; S: soft goal; R: resource; T: task).

$$f_A(a) = \begin{cases} 1, sort_A(a) = \{software\} \\ 0, \text{otherwise} \end{cases}$$

$$g_{A,D}(a) = \begin{cases} 1, \exists h \in A: sort_A(h) = \{human\}: \\ \quad ((a, h, x) \in D \vee (h, a, x) \in D) \\ 0, \text{otherwise} \end{cases}$$

$$f_D(d) = \begin{cases} 1, sort_D(d) = \{goal\} \\ 0.5, sort_D(d) = \{resource\} \vee sort_D(d) = \{task\} \\ 0, sort_D(d) = \{soft\ goal\} \end{cases}$$

$$g_D((a, b, x)) = \begin{cases} 0, sort_A(a) = \{software\} \wedge sort_A(b) = \{software\} \\ 0.5, a = Administrator \wedge sort_A(b) = \{software\} \\ 0.5, b = Administrator \wedge sort_A(a) = \{software\} \\ 1, \text{otherwise} \end{cases}$$

	Interface interaction	f_D(d)
CAD	8	0,8
MP	4	0,4
MS	7	0,7
PD	10	1
ECP	5	0,5
ES	2	0,2

	PCA	IUD	LF	AUP	PD	AD	$f_D(d)$
CAD	1	5	1	1	3	7	0,26
MP	1/5	1	1/3	1/5	1	5	0,08
MS	1	3	1	3	7	5	0,32
PD	1	5	1/3	1	5	5	0,23
ECP	1/3	1	1/7	1/5	1	5	0,08
ES	1/7	1/5	1/5	1/5	1/5	1	0,03

Fig. 5. Four different proposals of metrics for the *complexity of user interface* property.

7. Conclusions and Future Work

In this paper we have presented a framework for the definition of structural metrics for actor-dependency models. The object of measure are the properties of the system model, which usually represent non-functional or organizational requirements. The framework consists of three categories of metrics with two general forms each, one for actor-dependency metrics and another for dependency-based. We have illustrated our proposal in a particular case, a meeting scheduler system, using structural metrics as assessment for deciding the convenience of software support, then selecting a particular combination of COTS components for the system. As a result, and this is the main contribution of the paper, we may say that actor-dependency models can be analysed in a systematic way with respect to a given set of model properties: it is enough with choosing and tuning the appropriate type (or types) of metrics for each property.

The most relevant characteristics of our approach:

- Expressiveness. It takes into account roles, sorts and attributes of actors and dependencies; distinguishes global, local and sensitivity model analysis; allows to focus on actors or dependencies taking into account which concept is predominant in the particular metric.
- Accuracy. We have provided a formal definition of actor-dependency models that is used as a baseline upon which we have build our framework.
- Cost-sensitivity. Metrics can be defined more or less accurately depending on the effort put in their definition, up to the level of defining the weight of every model element individually.
- Skill support. Since the process to define metrics is the same over and over, namely tailoring a metric general form to the specific needs of a model property, accumulated knowledge may provide skills for augmenting productivity. Tool support helps in developing such skills, both in the form of spreadsheets or ad-hoc software systems which we are already developing.

- Reusability. Metrics can be used in different models of the same kind of domain.
- Generality. It has not been tightened to any particular goal-oriented language or formalism. The underlying concepts have been defined in an abstract way allowing therefore customisation.

To the best of our knowledge, there is not much related work in the field. Yu proposed in his definition of *i** [2] concepts such as opportunity and vulnerability that could be modelled within our framework. The most remarkable proposal in this area is part of the AGORA method [21] that provides techniques for estimating the quality of requirements specifications in a goal-oriented setting. In fact, AGORA puts more emphasis in the analysis of the AND/OR graph resulting from decomposition than in the kind of actor-dependency models that we have analysed in this paper. AGORA is expressive and accurate enough, although the method does not address explicitly to the other 3 issues mentioned for our framework (cost-sensitivity, reusability and generality). The basic difference is that AGORA does not provide any kind of general forms of the metrics, making then harder and less systematic to define new metrics or to reuse (parts of) them.

We have identified several ways to proceed ahead in this line of research. For making our proposal useful, we remark the following:

- Construction of a catalogue of reusable metrics. Model properties addressing non-functional aspects such as security, efficiency and so on are likely to appear over and over in system analysis. Thus, we aim at providing a complete and versatile catalogue of metrics for such properties.
- Integration of the framework with other proposals. In particular, we are especially interested in using this framework in the analysis of system architectures [15, 22]. We think that metrics on goal-oriented models may provide a first-cut criteria for classifying candidate architectures.

7. References

[1] A. Dardenne, A. van Lamsweerde, S. Fickas, "Goal-directed Requirements Acquisition", *Science of Computer Programming*, 20, 1993, pp. 3-50.

[2] E. Yu, *Modelling Strategic Relationships for Process Reengineering*, PhD. thesis, University of Toronto, 1995.

[3] J. Castro, M. Kolp, J. Mylopoulos, "Towards Requirements-Driven Information System Engineering: The Tropos Project", *Information Systems*, 27, 2002.

[4] E. Yu, "Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering", *Proceedings of the 3rd IEEE International Symposium in Requirements Engineering (ISRE'97)*, 1997.

[5] V. Santander, J. Castro, "Deriving Use Cases from Organizational Modeling", *Proceedings of the IEEE 10th Joint International Requirements Engineering Conference (RE'02)*, Essen (Germany), Sept. 2002, pp. 32-42.

[6] M. Lorenz, J. Kidd. *Object-oriented software metrics: a practical guide*. Prentice-Hall, 1994.

[7] S.R. Chidamber, C.F. Kemerer. "A Metrics Suite for Object-Oriented Design". IEEE TSE 20(6), 1994.

[9] A. van Lamsweerde, R. Darimont, P. Massonet, "Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learned", *Proceedings of the IEEE 2nd International Symposium on Requirements Engineering (ISRE'95)*, 1995, pp. 194-203.

[10] E. Yu. "Understanding 'why' in software process modeling, analysis and design". In *Proc. of the 16th Int. Conf. on Software Engineering*, ICSE'94, pages 159-168, Sorrento, Italy, May 1994

[11] J.P. Cavano, J.A. McCall, "A Framework for the Measurement of Software Quality", *Proceedings of the ACM Software Quality Assurance Workshop*, 1978.

[12] T.L. Saaty, *The Analytic Hierarchy Process*, McGraw-Hill, 1990.

[13] D. Garmus, D. Herron. *Measuring The Software Process: A Practical Guide to Functional Measurements*. Prentice-Hall, 1995.

[14] B.C. Meyers, P. Oberndorf, *Managing Software Acquisition*, Addison-Wesley, 2001.

[15] L. Baas, P. Clements, R. Kazman. *Software Architecture in Practice, 2nd edition*. Addison-Wesley, 2003.

[16] X. Franch, N.A.M. Maiden, "Modeling Component Dependencies to Inform their Selection", *Proceedings of the 2nd International Conference on COTS-Based Software Systems (ICCBSS'03)*, LNCS 2580, Springer-Verlag, 2003.

[17] *ISO/IEC Standard 9126-1 Software Engineering – Product Quality – Part 1: Quality Model*, 2001.

[18] L. Chung, B. Nixon, E. Yu, J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, 2000.

[19] T.J. Reynolds, J. Gutman, "Laddering Theory, Method, Analysis and Interpretation", *Journal of Advertising Research*, vol. 28, 1988, pp. 11-31.

[20] B. W. Boehm, P. Grünbacher, R. O. Briggs, "EasyWinWin: A Groupware-Supported Methodology for Requirements Negotiation", Proceedings of the 23rd ICSE 2001, 12-19 May 2001.

[21] H. Kaiya, H. Horai, M. Saeki, "AGORA: Attributed Goal-Oriented Requirements Analysis Method", *Proceedings of the IEEE 10th RE*, Sept. 2002, pp. 13-22.

[22] Grünbacher, P., Egyed, A., Medvidovic, N. "Reconciling Software Requirements and Architectures - The CBSP Approach,". *Proceedings of the 5th IEEE International Symposium on Requirements Engineering* (RE), Toronto, Canada, August 2001, pp. 202-211.