2004

Hierarchic decomposition in agent oriented conceptual modelling

R. B. Brown
University of Wollongong, bobbrown@uow.edu.au

Aditya K. Ghose
University of Wollongong, aditya@uow.edu.au

Hierarchic decomposition in agent oriented conceptual modelling

Abstract

Software development processes requires a thorough understanding of stakeholder objectives and requirements. Product-centrism is an insufficient stance from which to achieve greater efficiencies and reduce reengineering. Stakeholder requirement elicitation is thus worthy of formalization. A suite of tools, notably the i* model, provides a framework for early-phase requirements capture. These tools currently are at best only semiautomated and essentially consist of a notational glossary and sets of mark-up symbols. Increasing formalization may lead to greater automation of the process in the future, but currently there is a degree of flexibility that presents pitfalls for the unwary practitioner. A notion of contextual consistency would enhance the applicability such toolkits. Requirements generated from stakeholder objectives may suffer scoping errors, complicated by the complexity of practical examples. Hierarchical situations of contextual confusion are explored. A formalisation is offered of the constraints that circumscribe the set of valid decompositions.

Disciplines

Physical Sciences and Mathematics

Publication Details

This article was originally published as: Brown, RBK, Ghose, A, Hierarchic decomposition in agent oriented conceptual modelling, Proceedings Fourth International Conference on Quality Software (QSIC 2004), 8-9 September 2004, 240-247. Copyright IEEE 2004.

Hierarchic Decomposition in Agent Oriented Conceptual Modelling

Robert B.K. Brown Aditya Ghose
School of Information Technology and Computer Science, University of Wollongong
c/o SITACS Bldg3, University of Wollongong, NSW 2522 Australia
{bobbrown,aditya}@uow.edu.au

Abstract

Software development processes requires a thorough understanding of stakeholder objectives and requirements. Product-centrism is an insufficient stance from which to achieve greater efficiencies and reduce re-engineering. Stakeholder requirement elicitation is thus worthy of formalization. A suite of tools, notably the i model, provides a framework for early-phase requirements capture. These tools currently are at best only semi-automated and essentially consist of a notational glossary and sets of mark-up symbols. Increasing formalization may lead to greater automation of the process in the future, but currently there is a degree of flexibility that presents pitfalls for the unwary practitioner. A notion of contextual consistency would enhance the applicability such toolkits. Requirements generated from stakeholder objectives may suffer scoping errors, complicated by the complexity of practical examples. Hierarchical situations of contextual confusion are explored. A formalisation is offered of the constraints that circumscribe the set of valid decompositions.*

1. Introduction

Software engineering (SE) centres on two key concepts: 'engineering discipline' and 'all aspects of software production' [1]. Such 'traditional' software developments are product-centric. Some traditional software 'myths' disregard stakeholder objectives and the work then required after initial presentation to adjust the product to suit clients' needs [2].

The SE community is recognising that inadequate requirements lead to increased likelihood of failure, especially on 'softer' socio-political grounds [3]. This highlights the need to elicit stakeholder requirements and target their softer, qualitative objectives.

Maciaszek draws the production-centric distinction between 'business rules' – a functional requirement describing an 'always on' (invariant) aspect of the system, and 'constraint statements' which define restrictions on system behaviour or the production process [4]. Whilst this distinction may not strictly address stakeholder preference, it allows acceptance or rejection on these grounds.

Stakeholders generate their own notations and terminologies, complicating the business of capturing such details [5]. This difficulty informs approaches to requirement elicitation that are sensitive to consistency and viewpoint. It is necessary therefore to use a systematic approach when capturing requirements. A significant risk of failure exists in marginalizing the stakeholder's softer objectives, despite their inherent messiness.

One approach to identifying stakeholder preferences adopts a goal-orientation and asks 'what does the stakeholder want to achieve'. Goal formulations express intended system properties [6]. Goal-centrism offers stability as top-level goals are often invariant under decomposition, and facilitate back-tracking when re-design issues arise.

Shifting to process orientation necessitates consideration of the software in its environment. It becomes necessary to identify active elements that have choice [6]. Such active elements (actors) are the loci for the formulation of goals and preferences. Process-centrism requires examination of actor internal states. This could be facilitated by use of a tool that can standardize the capture of intentions, motivations and rationales.

The remainder of this paper constitutes a review of one such toolset and describes an extension to it.

Proceedings of the Fourth International Conference on Quality Software (QSIC'04)
0-7695-2207-6/04 $ 20.00 IEEE

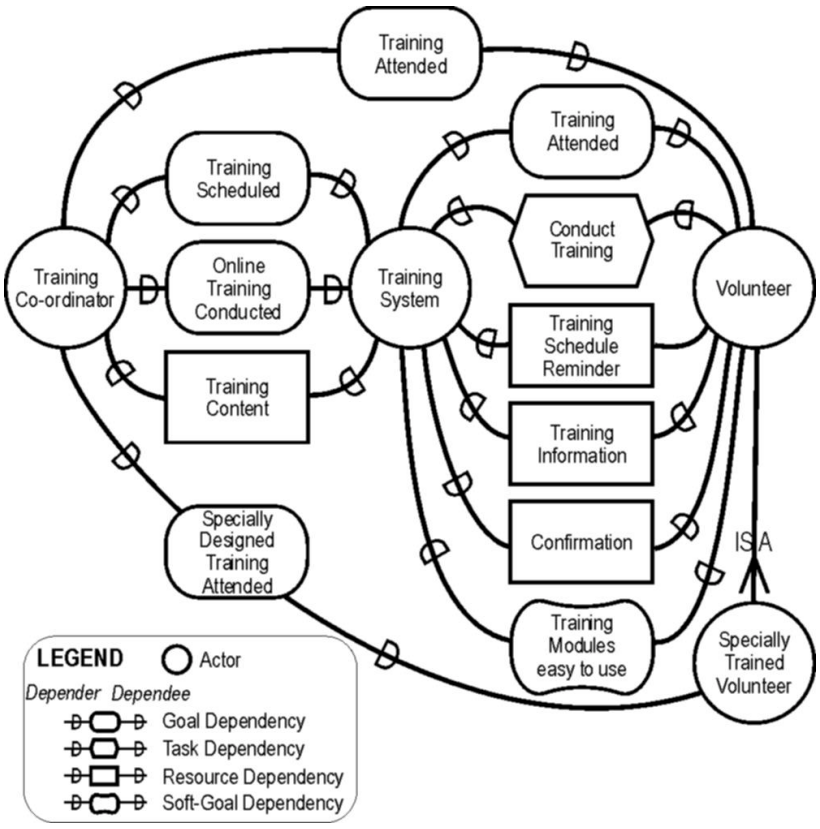

Figure 1: SD model of the activity: "provide training to volunteers"

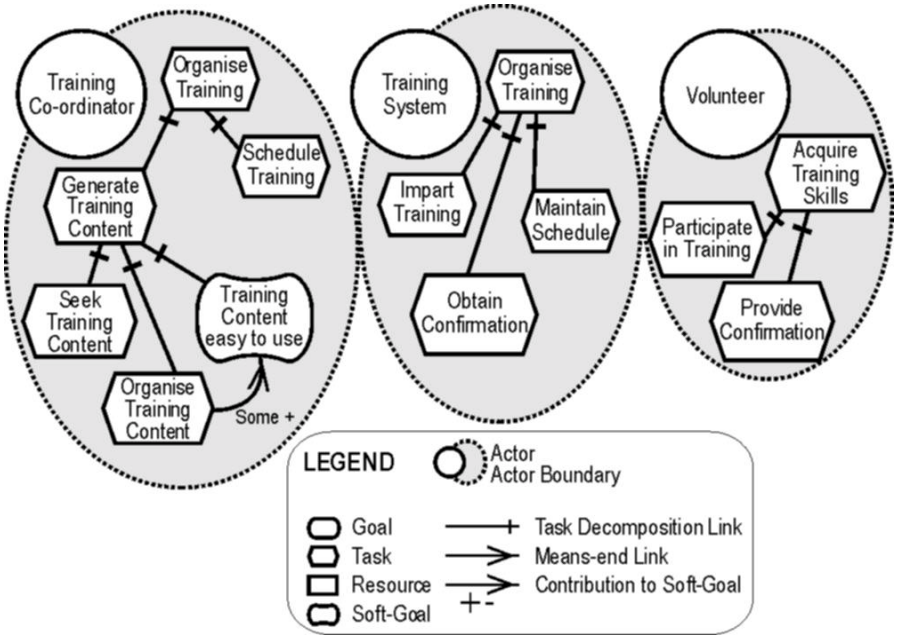

Figure 2: Some SR models of actors in the activity: "provide training to volunteers"

2. Intentionality

Eric Yu's model, i* ('eye-star'), meaning "distributed intensionality" [*sic.*], though initially conceived as a business-process engineering tool, frames processes as social activities between actors who depend upon each other for goals to be obtained and tasked performed [7]. It links organizational design decisions to strategic business reasoning, which approach can be applied to the formulation of requirements for a software project.

i* incorporates two main diagrammatic tools, the Strategic Dependency model (SD) and the Strategic Rationale model (SR) [8].

3. SD and SR Models

In examining some activity stakeholders deem necessary, SD's deal with five entity types: the Actor entity represents a human, synthetic or organizational active agent capable of decision. 'Actor' is loosely equivalent to the concept 'role', which invites confusion. The 'Task' entity, which need not represent a complete system specification, is an artefact of a *dependency relationship,* definable between actors [8]. 'Resources' are entities without outstanding open issues, and may be called upon by actors. A Goal is an assertional statement between actors in which the *dependee* is free to choose actions that will resolve the goal. The 'Softgoal' entity whilst also an assertional statement, is qualitative, with no clear notion of satisfaction. Softgoals may, at best, be "sufficiently satisfied" (or 'satisficed'). Softgoals behave like preference constraints to be optimised [9]. Softgoals represent typically qualitative non-functional requirement, such as 'the software shall be learnable". Figure 1 is an example of an SD model, examining the activity, "provide training to volunteers".

Whilst entities are described in dependency relations within the SD, actors' internal intentions and rationale remain unexamined. For a more complete model of the internal workings of an actor, Yu proposed the SR model, drawing upon the earlier work of Chung in representing non-functional requirements [10].

The SR model may be envisaged as the *inside* of an actor. It is drawn as a dashed ovoid, extending from the actor entity node. Figure 2 serves to illustrate the nature of an SR, representing the internal structure of some actors from the action "providing training to volunteers".

Within this SR are defined links not unlike those in the SD, however these linkages are more specific. Tasks are decomposed via Task-Decomposition links indicating an ordinal ranking. Goals may be linked to other entities with Means-End links indicating tasks or resources required for satisfaction.

Softgoals have Means-Ends links that are qualified with an indicator of the *degree* to which the linked entity contributes towards satisficement. The contributions include *make, break, help, hurt, positive, negative, and, or, unknown* and *equal* [11]. In Figure 2, a task is seen to provide a 'somewhat positive' contribution towards the Softgoal "training content easy to use", within the actor "Training Coordinator".

SR diagrams may exist within SD diagrams for the same activity. When opened out, dependency linkages previously connected to the actor may then link to an element within that actor's SR.

4. Consistency in Intensionality

The scope of the activity is initially described by a client request. Populating the entities of the activity requires investigative sessions with stakeholders to identify high-level goals and dependencies. Refinement occurs as the analyst deconstructs goals and identifies resources and tasks.

There is a long history of requirement engineering employing techniques of abstraction and refinement. The i* model serves as a convenient toolkit for abstracting detail gathered by eliciting intensions and dependencies from stakeholders. Hierarchic decomposition becomes necessary as the abstraction is refined, and we propose just such an extension to i*.

In practical situations however, we can expect to encounter complex situations. We should thus expect to be working with large and complex models. The value of this early phase intensionality capture and analysis in software development requires that these complex models can be checked for consistency, and further, that we have a system by which the software development can be verified and validated against the original expressions of intent as elicited from stakeholders.

5. Contextual Inconsistency

We anticipate the use of a *co-evolutionary* framework, whereby diagrammatic or plain-language analysis of qualitative stakeholder preferences (such as i*) parallels the construction of formalized structures that preface pseudo-code. Horizontal checks between the two processes ensure consistency. This requires some notion of consistency for the diagrammatic structure.

Whilst an SR contains the internal intentionality of an actor, it is possible a high-level actor's SR may need

to contain sub-actors. It is also possible that an actor as originally described by the stakeholder could fulfil, for the purposes of the activity in question, two or more 'roles'.

Whilst an i* model can cope with such problems by splitting agents ('actor' is often synonymous with 'role'), the problem may not be identified and resolved in time or within budget. The very flexibility of diagrammatic systems can allow errors of granularity to persist.

6. Hierarchic Inconsistency

The activity may involve dependencies active on both the high-level 'parent' actor and the lower-level 'sub' actor. For all of these entities to be modelled on the one conceptual 'plane' may give rise to difficulties.

Let us imagine a high level plane contains a diagram designated "SD". We may then imagine SR diagrams for each of the actors existing on the same plane. Let us label these SR diagrams as $SR_1, SR_2, \ldots, SR_n$ where n is the number of actors in SD.

Let us now imagine that SD has been examined and some detail elicited at a lower hierarchic level (SD′). SD′ may yield its own SR diagrams labelled $SR'_1, SR'_2, \ldots, SR'_m$.

SD and SD′ should be consistent because the actors of SD′ have a part-whole relationship with parent actors in SD. We examine rules for this below. Relationships between SD and the SR diagrams should be obvious, as are relationships between SD′ and its constituent SR′ diagrams. Less obvious is the meaning of consistency between the SR diagrams, and SD′.

Consider the following example:

Activity: "University sells online degree courses through a partner offshore campus". The SD contains actor *a* University (Uni) and *b* Partner Campus (Partner) among others. The actor *a* contains a number of sub-actors a'_n, including a'_1 Faculty of Informatics and a'_2 IT Services (ITS).

For administrative, accounting and security reasons, a'_2 (ITS) requires the use of common application suites, and common user platforms. For curricular and pedagogical reasons, a'_1 (Informatics) prefers to use its own connections, serving its own applications.

Actor b'_1, (Partner Campus Teaching Staff), for reason of its slow remote connection, may require a hand-crafted a'_1-style solution with 'stripped-down' applications permitting quicker service for students. Sub actor b'_2, (Partner Campus IT), may simultaneously support the standardised a'_2-style solution for maximum network security.

If a'_1 and a'_2 are somehow left *within* the *a* actor, then clearly there will arise inconsistent dependencies. Without deconstructing the parent *a*, there is little hope of identifying or resolving the issues.

Currently published versions of i* have not addressed this issue, and have not defined rules for enforcing hierarchical consistency. Indeed, the notion of "parent" and "sub" actor entities is not defined at all. Such situations require an entire re-conceptualisation of the activity.

Linking a hierarchic set of SD's can allow full explication of the issues and dependencies. Constructing such a set however, requires definitions of hierarchic consistency.

7. Guidelines for Detecting Inconsistency

It is empirically true that SR diagrams obey certain purely structural rules in i* space.

1. SR diagrams map to specific actors, and do not intersect with each other when drawn into the SD diagram that contains all actors involved in a given action.
2. SR diagrams wholly contain means-end and task-decomposition links.
3. SR diagrams do not wholly contain dependency-arrow links.
4. SR boundaries consist of the locus of points which perpendicularly bisect all of the dependency linkages which have a terminus within them, yet simultaneously do not cut any means-end or task-decomposition linkages.

Any breach in these observed trends is a malformation of the model and a strong indicator of a misconceptualisation of the activity. This may indicate the need to construct a layered set of SD's.

8. Formalizing Constraints on Decompositions

We shall now focus on formalizing constraints that circumscribe the set of valid decompositions of SD models. We shall express our constraints in terms of *FormalTROPOS* specifications underlying SD models. Our constraints will be expressed in terms of adjacent layers (a *higher layer* and a *lower layer*) in a potentially multi-layered decomposition. We assume the ability to trace every actor in a lower-level SD model to a *parent actor* in the immediate higher layer. We represent the constraints in terms of two kinds of

relationships between conditions in higher- and lower-level SD models: *consistency* and *entailment*.

We shall consider dependencies first. Three kinds of formal conditions may be associated with each dependency: *creation conditions* (for a dependency d, these are denoted by $C_{create}(d)$), *invariant conditions* (for a dependency d, these are denoted by $C_{invariant}(d)$) and *fulfilment conditions* (denoted by $C_{fulfil}(d)$) [12]. Let actor a in a higher-level SD model be decomposed into n actors $a_1',\ldots,a_n'$ in the immediate lower-level SD model. This structure is illustrated in Figure 3.

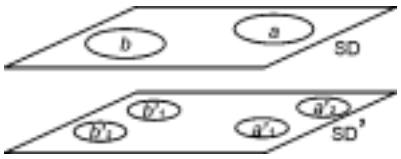

Figure 3: An SD hierarchy showing parent and sub actors

Let $\{d_1,\ldots,d_k\}$ be the set of dependencies in which actor a is either a depender or a dependee. Let $\{d_1',\ldots,d_l'\}$ be the set of all dependencies in which any of the actors in the lower-level SD model are either a depender or a dependee. We list the constraints below and illustrate them with examples drawn from the previous sample activity "University sells online degree courses through a partner offshore campus":

In the following, we will use the notion of a maximal consistent subset (MCS) of a set of sentences in an underlying formal language. For our purposes, this language will be that used in *FormalTROPOS*, i.e., linear-time typed first-order temporal logic. However, this could vary, the only requirement being that the language should come with a well-defined notion of consistency. We will use *MCS(X)* to denote the set of maximal consistent subsets of a set of sentences X.

For any $m \in MCS(X)$, $m \not\vdash \bot$

and there does not exist any m'

such that $m \subset m' \subseteq X$ where $m' \not\vdash \bot$

- *Constraints on creation conditions:*

  If $\bigcup_i C_{create}(d_i) \vdash \bot$,

  $\bigcup_i C_{create}(d_i) \cup \bigcup_i C_{create}(d_i') \not\vdash \bot$

Thus, the set of all creation conditions for all dependencies involving an agent a in a higher-level SD model, taken together, do not contradict the set of creation conditions for all dependencies involving actors obtained by decomposing actor a.

This consistency constraint is weak. University *(a)* initiates a dependency on the network resource to achieve successful sale of course material. Neither a'_1 nor a'_2 may have contradictory agendas, such as: a'_2 tries to limit delivery of course materials to marginalise a'_1's claim to a slice of next years operating budget.

It is important to note that the constraint discussed above applies only in instances where the collection of creation conditions of dependencies in the higher layer is consistent. We note that this is often not the case, such as with dependencies that are temporally ordered and which effect changes in the context being modelled. A dependency, for instance, might be created when a condition $\neg p$ holds, and might have the condition p as a fulfilment condition. This condition p in turn may lead to the creation of another dependency. Given this, we need to articulate the following additional (and somewhat weaker) constraint:

If $\bigcup_i C_{create}(d_i) \vdash \bot$, then for every

$m \in MCS(\bigcup_i C_{create}(d_i))$, there exists an

$m' \in MCS(\bigcup_i C_{create}(d_i'))$,

such that $m \cup m' \not\vdash \bot$

In other words, for every maximal consistent subset of the set of creation conditions for dependencies in the higher layer SD diagram, there exists a maximal consistent subset of the set of creation conditions for dependencies in the lower layer SD diagram that it is consistent with. We note that the set of maximal consistent subsets of the set of creation conditions could be a singleton, in which case the current constraint reduces to the previous one. The intuitive motivation for these constraints is simply the observation that; when an actor in a higher layer is decomposed into lower layer actors the dependencies that the higher layer actor might be involved in are also correspondingly decomposed. We have imposed the constraint that every lower layer actor can be traced back to a unique higher layer actor (in some sense the "parent" actor). We believe that a similar constraint cannot be imposed on dependencies, i.e., a given lower layer dependency may incorporate aspects of multiple higher layer dependencies. The only constraint that can be imposed is a consistency constraint. It should not be the case that the creation conditions of all dependencies in the lower layer that could be created simultaneously (i.e., whose creation conditions are consistent) contradict those of a higher layer dependency, since this would indicate that the lower layer model violates the semantics of the higher layer model in some sense.

- *Constraints on invariant conditions:*

  If $\bigcup_i C_{invariant}(d_i) \not\vdash \bot$,

  $\bigcup_i C_{invariant}(d_i) \cup \bigcup_i C_{invariant}(d_i') \not\vdash \bot$

Thus, the set of all invariant conditions for all dependencies involving an agent a in a higher-level SD model, taken together, do not contradict the set of

invariant conditions for all dependencies involving actors obtained by decomposing actor a.

This consistency constraint is also weak. Contrary reasons for maintaining a dependency should not arise, such as: Partner *(b)* maintains receipt of course materials in a form best suited to students network capacity but Partner Teaching Staff *(b′1)* wish to restrict student access to foreign materials to further the development of local content.

As with creation conditions, we recognize that the set of invariant conditions for a given higher layer may be contradictory, possibly because of the temporal ordering of the dependencies. In the event that this is the case, the following weaker constraint applies:

If $\bigcup_i C_{invariant}(d_i) \models \perp$, then for every

$m \in MCS(\bigcup_i C_{invariant}(d_i))$, there exists an

$m' \in MCS(\bigcup_i C_{invariant}(d_i'))$, such that $m \quad m' \not\models \perp$

The motivations for this weaker constraint are similar to those discussed for the constraints on creation conditions.

In our example activity "University sells online degree courses through a partner offshore campus", consider that *creation* and *invariant* conditions may vary over time and/or context. Thus, at one time the high-level (SD) actor *b* (Partner) may have the Goal Dependency upon actor *a* (Uni) "students have access to online teaching materials". The *creation* condition will arise immediately before the commencement of course delivery to the students at the partner campus. *Invariant* conditions hold throughout the delivery of the material to the students.

At a later time, created under the condition that delivery of the material is complete, in order to safeguard intellectual property, the high level (SD) actor *a* (Uni) has the Goal Dependency upon actor *b* (Partner) that, "student denied access to online teaching materials". At least two MCS's can be expected in SD, one containing each of these contradictory dependencies.

Under decomposition, the lower level SD′ is expected to contain related dependencies that reflect those in the higher level. Sub-actor a'_1 (Informatics Faculty) will have a Task Dependency upon actor a'_2 (IT Services) to "enable student access privileges" created just before course delivery starts, and invariant during delivery. The fulfilment of this dependency itself forms a creation condition, when all course elements have been satisfactorily delivered to Partner students. Under this condition, sub-actor a'_1 (Informatics Faculty) will have a Task Dependency upon actor a'_2 (IT Services) to "restrict student access privileges".

We can see that at least one MCS will exist in SD′ for each of inconsistent dependencies. A well-formed model will have at least one MCS in SD′ that is not inconsistent with each MCS found in SD.

- *Constraints on fulfilment conditions:*

  If $\bigcup_i C_{fulfil}(d_i) \not\models \perp$,

  $\bigcup_i C_{fulfil}(d_i') \models C_{fulfil}(d_i)$

In other words, the combination of the fulfilment conditions of the dependencies in the lower-level model entail that the fulfilment conditions of the higher-level dependencies have been satisfied.

This entailment constraint is strong. It requires that fulfilment of SD′ goals *entails* fulfilment of SD goals. In the case where entailment does not follow, then the decomposed SD′ is either incomplete or flawed. This strongly suggests a misconception of stakeholder goals and rationales.

If the collection of fulfilment conditions in the higher layer model is inconsistent, the following weaker constraint applies:

If $\bigcup_i C_{fulfil}(d_i) \models \perp$, then for every

$m \in MCS(\bigcup_i C_{fulfil}(d_i))$, there exists an

$m' \in MCS(\bigcup_i C_{fulfil}(d_i'))$, such that $m' \models m$

These conditions hold where the union of each of the three classes of dependencies in the higher level (SD) are consistent.

We can posit the case however, where inconsistencies exist between dependency conditions of a given hierarchic level of an SD. As temporality is not explicit in the i* notation, an SD may contain dependencies which are inconsistent over time. For example, a Goal *k* may need to be satisfied at one time, but another Goal $\neg k$ at another time.

We must therefore extend our constraints to cover such instances. We initially address our *creation* and *invariant conditions*, which impose a weak notion of consistency.

Under decomposition from SD to SD′, the number of the terms used from the language may be expanded, but cannot be contracted. The higher level SD is, in effect, a more abstract description of higher level intensions and dependencies, whilst the lower level SD′ is a more refined description, containing more specific intensions and dependencies.

Without inconsistency, the set of terms in an SD is itself a consistent set. If a given layer of the SD hierarchy contains inconsistencies, then we may write out a finite number of sets of terms each of which is an MCS.

We recognise the extended conditions thus:

$\text{number}(MCS_{SD}) \leq \text{number}(MCD_{SD'})$

The refined lower layer, SD′, must contain *at least* the same number of MCS's as the higher abstract layer, SD. SD′ may contain more, as it may have invoked more terms from the language, under refinement.

For each $MCS_{SD'}$ $\exists$number(MCS_{SD}) ≥ 1

s.t. these MCS_{SD} $MCS_{SD'} \nvDash \perp$

For each MCS in the lower, more refined, layer, there must exist at least one MCS in the more abstract higher layer with which it is consistent.

For the stronger entailment constraint on the *fulfilment* condition, we must similarly consider the ideal case.

$\bigcup_i C_{fulfil}(d_i') \nvDash \perp$

and $\bigcup_i C_{fulfil}(d_i) \nvDash \perp$

In this case, the union of *fulfilment conditions* in the SD is itself consistent, AND the union of *fulfilment conditions* in the SD′ is itself consistent.

If this is not so, we must address two more cases; one: where the higher layer has inconsistencies, and two: where inconsistencies in the lower layer arise through decomposition.

For each MCS_{SD}

number($MCS_{SD'}$) $\geq 1 \vDash MCS_{SD}$

Each MCS of a higher layer SD which contains inconsistencies, is entailed by *at least* one MCS of the lower layer SD′.

For each $C_{fulfil}(d_i)$

number($MCS_{SD'}$) $\geq 1 \vDash C_{fulfil}(d_i)$

When decomposition of an internally consistent SD yields a lower SD′ which contains inconsistencies, then for each fulfilment condition of SD, there must exist at least one MCS of terms in SD′ which entails it.

From our example activity we illustrate the first of these two cases. Actor *a* (Uni) has the Soft-Goal Dependency upon *c* (Partner Campus Students) to "maximise number of students recruited from offshore to Uni Campus", whilst the *b* (Partner) has the competing Soft-Goal Dependency upon *c* (Partner Campus Students) "maximise the number of students retained at Partner Campus". These are *created* when the delivery contract is signed and remain *invariant* as long as the contract remains in effect. As soft-goals, they are *fulfilled* when they are *satisficed* ie: when student numbers are sufficiently maximised.

In SD′, actor a'_1 (Informatics Faculty) has the Soft-Goal Dependency upon a'_4 (Uni Recruitment) to "maximise recruitment of students from Partner Campus to Uni campus" whilst $b'3$ (Partner Campus Management) has the Soft-Goal Dependency upon $b'1$ (Partner Campus Teaching Staff) to "maximise retention of Partner students at Partner Campus". The *creation*, *invariant* and *fulfilment* conditions are essentially the same as those of the related dependencies in the higher level SD.

From our previous constraints, there should be at least one MCS in SD for each of the contradictory high level dependencies, and there will be at least one not-inconsistent MCS in SD′ for each. Our stronger constraint in the case of *fulfilment* conditions requires that there be at least one MCS in SD′ which, when fulfilled, entails fulfilment of an MCS from SD. Thus when a'_1 has maximised its recruitment, then actor *a* will have fulfilled its related goal, and the same can be said of actors b'_3 and *b*.

Failure to meet these constraints implies the model is not well formed. Either the high level SD has been misconceptualized and/or the decomposition is incomplete or flawed.

From our example activity we illustrate the latter of these cases. Consider that SD contains no inconsistent dependencies. Actor *a* (Uni) has the Task Dependency upon actor *b* (Partner) to "sign delivery contract with Partner campus" amongst others, whilst actor *b* (Partner) has the Goal Dependency upon actor *a* (Uni) "local staff trained to deliver course material" amongst others. Both actors *a* and *b* have the Goal Dependency upon actor *c* (Partner Campus Students) to have "received complete transmission of course materials".

Under decomposition, when the intensions of sub-actors are considered, it becomes apparent that an inconsistency has arisen. Actor a'_1 (Uni Informatics Faculty) wishes to retain ownership of all intellectual property (IP) associated with the course material; thus a'_1 has the Task Dependency upon actor a'_3 (Uni Legal Unit) to "include IP protection clause into delivery contract". Actor b'_3 (Partner Campus Management) wish to be able to deploy aspects of the course material in the future so they may attain independent university status; thus b'_3 has the Task Dependency upon b'_1 (Partner Campus Teaching Staff) "local teachers learn and retain knowledge of all course material".

Observe that *fulfilment* of the Task Dependency "include IP protection clause into delivery contract" entails fulfilment of the higher level Task Dependency "sign up Partner campus", and that *fulfilment* of the Task Dependency "local teachers learn and retain knowledge of all course material" entails fulfilment of the higher level Goal Dependency "local staff trained to deliver course material".

9. Conclusions and Future Work

Under process-orientation, Goals identification and the requirement formulation through goal analysis have matured into useful suites of notational and analytical tools.

Re-engineering is identified as a drain of resources in the software life cycle. The inability to analyse and reason with stakeholder qualitative preference and rationale is a major source of software failure.

The i* framework presents a notation for representing process-oriented goal-centric dependencies as elicited from stakeholders. The model provides a structural basis for retaining some notion of the intentionality and rationale that informs the actors in the activity.

The method grew out of a business-process modelling project, and when merged with earlier work on representing Softgoals in requirements engineering, became a somewhat mature software process-engineering tool worthy of consideration and exploration. The toolkit however necessarily retains a considerable degree of flexibility, and requires skilful use by the analyst.

Efforts are underway to formalise the toolkit, with a view to automating the production of requirements from stakeholder utterances. Until that extension is realised, the very flexibility which is a strength of the method provides scope for hierarchic inconsistency to enter the model, informed by initial stakeholder claims in the absence of domain expertise on the part of the analyst.

The i* toolkit currently lacks the rules and procedures to identify and resolve hierarchic inconsistency. Some simple rules of structural consistency are observed, and a formalisation of constraints that circumscribe the set of valid decompositions if offered, which may point the way towards a semi-automated hierarchic consistency checking extension to i*.

Dependencies may not remain invariant under situational context. Currently, i* can express roles, but not readily identify or resolve role-related inconsistency. Future work will consider such horizontal or role-taking inconsistencies.

Resolving both consistency dimensions within diagrammatic methodologies will lead to tools for crosschecking between the diagrammatic and formal strands of a *co-evolutionary* SE framework.

Future work will propose formal constraints that hold over the relationships that exist between elements of SD and SR diagrams, both within and between hierarchically decomposed layers.

10. References

[1] Sommerville, I., *Software Engineering (6th Ed)*, Edinburgh Gate, Pearson Education, 2001, pp.7.

[2] Pressman, R.S., Software engineering – a practitioner's approach (4th Ed), New York, McGraw-Hill, 1997, pp.17-18.

[3] Goguen, J.A. and C. Linde, "Techniques for requirements elicitation", Proceedings of the IEEE international symposium on requirements engineering. 4-6 Jan. 1993, pp. 152-164.

[4] Maciaszek L.A., Requirements analysis and system design – developing information systems with UML, Sydney, Addison Wesley, 2001, pp.16.

[5] Sommerville. I., P. Sawyer, and S. Viller, Viewpoints for requirements elicitation: a practical approach, Proceedings of the 3rd IEEE international conference on requirements engineering (ICRE'98), Colorado Springs, USA, 6-10 Apr. 1998, pp.74 – 81.

[6] Lamsweerde, A.van., "Goal-oriented requirements Engineering: a guided tour", Proceedings of the 5th IEEE international conference on requirements engineering (ICRE'01), Toronto, Canada. 27 –31 Aug. 2001, pp. 249 – 262.

[7] Yu, E.S.K., J. Mylopoulos and Y. Lesperance, "AI models for business process reengineering", IEEE Expert, 11(4) , Aug. 1996, pp.16-23.

[8] Yu, E.S.K., Strategic Modelling for Enterprise Integration, Proceedings of the 14th World Congress International Federation of Automatic Control, Beijing, China, 1999.

[9] Krishna, A., A.K. Ghose, S.A.Vilkomir, "Co-evolution of complementary formal and informal requirements", To appear in the Proceedings of IWPSE 2004 - International Workshop on Principles of Software Evolution (held in conjunction with RE 04 - 12th IEEE International Requirements Engineering Conference), Kyoto, Japan, Sept., 2004.

[10] Chung, K.L., Representing and using non-functional requirements, Ph.D. thesis, Department of Computer Science, University of Toronto, 1993

[11] Mylopoulos, J., K.L. Chung and B. Nixon, "Representing and using non-functional requirements: a process-oriented approach", IEEE Transactions on Software Engineering, 18(6), Jun. 1992, pp.483 - 497

[12] Fuxman, A., R. Kazhamiakin, M. Pistore and M. Roveri, Formal Tropos: language and semantics, Trento, 2003.