# Transforming Software Package Classification Hierarchies into Goal-Based Taxonomies*

Claudia Ayala[1] and Xavier Franch

Universitat Politècnica de Catalunya
08034 Barcelona, Catalunya, Spain
{cayala, franch}@lsi.upc.edu

**Abstract.** Software package selection is an activity that plays an increasingly crucial role in the delivery of software systems. One of its main open issues is how to structure the knowledge about the software marketplace and in particular how to know which types of packages are available and which are their objectives. Profit and non-profit organizations of any kind use to arrange these types into categories in a hierarchical form. However, the rationale behind the proposals found is often confusing and therefore their usefulness is hampered. In this paper we propose the use of taxonomies for structuring this knowledge. Our taxonomies are goal-driven, which means that we provide a rationale for the decisions taken. The leaves of the taxonomies are the types of packages available in the market, whilst the intermediate nodes are categories that group them when closer relationships are found. The proposed taxonomies are not defined from the scratch but applying the appropriate transformation rules to some departing classification available. We define the syntactic form of the rules and also their applicability conditions as properties on the involved goals. We apply them to a particular case, a taxonomy for business applications.

## 1 Introduction

The amount of software packages [1, 2] available on the market is growing more and more. This tendency is due both to the increasing adoption of component-based software technologies by the community, and to the continuous creation of new communication and marketing channels that bridge the gap between providers and consumers of those products. Therefore, there is an increasing need for organizing the available types of software packages to achieve more efficient and reliable selection processes.

As a response to this need, profit and non-profit companies, organizations and teams have arranged this knowledge by defining categories of services, products and knowledge, usually structured in a hierarchical form [3-14]. Regardless of their comprehension, completeness, and scope, all of these proposals share a common characteristic that may be considered as a drawback: they present a hierarchy of items without a clear rationale behind. The categorization relies on experience, knowledge, and observation but they rarely use knowledge engineering and requirements engineering techniques to classify the enclosed items. Sometimes, the meaning of a

* This work has been partially supported by the Spanish MEC project TIN2004-07461-C02-01.
[1] C. Ayala's work has been partially supported by a FI grant (Catalan government).

K.V. Andersen, J. Debenham, and R. Wagner (Eds.): DEXA 2005, LNCS 3588, pp. 665 – 675, 2005.

particular category is not clear without further examining the items, especially if it is absolutely unknown to the user. As a result, the understanding, use, evolution, extension, and customization of the categorization proposal may be difficult.

The purpose of this paper is to present an approach for solving this drawback. Our proposal is based on the use of goals to provide semantics to the nodes of existing ad-hoc classification hierarchies. Then, we may define the process of taxonomy construction as the repeated application of transformation rules over the nodes of the source hierarchy. These transformations must satisfy some completeness and correctness conditions with respect to the involved goals which will be presented here. For illustration purposes, we apply these rules to a case study.

## 2 A Process for the Construction of Software Package Taxonomies

The transformation of an unstructured classification into a goal-based taxonomy can be roughly divided into two steps. Firstly, we apply techniques for discovering goals in the departing classification hierarchy; examples are the GBRAM method [15], built on top of the Inquiry Cycle approach [16], and decision trees building algorithms such as C4.5 [17] and CART [18] for expressing goals as a combination of classifiers values. Secondly, we rearrange the hierarchy to make it correct and complete by applying repeatedly transformation rules.

In the rest of the paper we focus on the second step of the process. In other words, we consider that the goal discovery process will be covered by existing techniques as those mentioned above. For defining formally the transformation rules, we first state the notions of taxonomy and goal and we define what a correct and complete taxonomy is. These two notions will induce naturally a process to apply the transformation rules in a comfortable order that guarantees termination while being expressive enough.

A *goal-oriented taxonomy* T is a tree over a domain. As such, we need predicates and functions shown in table 1.

**Table 1.** Predicates and functions over taxonomies

| | |
|---|---|
| *Belongs*(T, A): the element A belongs to T | *Parent*(T, A, B): A is parent of B in T |
| *Leaf*(T, A): A is a leaf in T | *Root*(T, A): A is the root of T |
| *Children*(T, A): returns the set of children of A in T | |
| *Successors*(T, A): returns the set of successors of A in T | *Goal*(A): goal of node A |
| *Ancestors*(T, A): returns the set of ancestors of A in T | *Name*(A): name of node A |

*Goals* are defined over a set $X = \{x_k\}_n$ of independent variables that characterize the taxonomy. *Goal satisfaction* is defined by means of assignment to the variables, therefore for each assignment *ass* = ($x_1 \leftarrow v_1$, …, $x_n \leftarrow v_n$), the expression $sat_{ass}(G)$ yields true if the goal G evaluates to true for this assignment, otherwise false. Throughout the rest of the paper we use the predicates on goals with the meaning and abbreviations showed in table 2.

**Table 2.** Predicates, semantics and abbreviations used over goals

| *Predicate or function* | *Semantics* | *Abbrev* |
|---|---|---|
| *impliesGoal*(G, H): the goal G implies the goal H | $\forall$ass: $sat_{ass}(G) \Rightarrow sat_{ass}(H)$ | G⇒H |
| *soft-impliesGoal*(G, H): the goal G implies the goal H for some assignment whilst the reverse is not true | $\exists$ass: $sat_{ass}(G) \Rightarrow sat_{ass}(H) \wedge \neg\exists$ass: $sat_{ass}(H) \Rightarrow sat_{ass}(G)$ | G±⇒H |
| *disjointGoals*(G, H): goals G and H are mutually exclusive | $\forall$ass: $\neg\, sat_{ass}(G) \wedge sat_{ass}(H)$ | ¬G∧H |
| *equivGoals*(G, H): goals G and H are equivalent | $\forall$ass: $sat_{ass}(G) = sat_{ass}(H)$ | G≡H |
| *emptyGoal*(G): goal G is never satisfied | $\forall$ass: $\neg\, sat_{ass}(G)$ | G=∅ |
| *F* = *diffGoals*(G, H): obtain the difference of goals G and H | $\forall$ass: $sat_{ass}(G) \wedge \neg\, sat_{ass}(H) \Leftrightarrow sat_{ass}(F)$ | G–H |
| *F* = *unionGoals*(G, H): obtain the union of goals G and H | $\forall$ass: $sat_{ass}(G) \vee sat_{ass}(H) \Leftrightarrow sat_{ass}(F)$ | G∪H |
| *F* = *intersectGoals*(G, H): obtain the intersection of goals G and H | $\forall$ass: $sat_{ass}(G) \wedge sat_{ass}(H) \Leftrightarrow sat_{ass}(F)$ | G∩H |
| *UnionGoalsExt*($\{G_k\}_n$), *intersectGoalsExt*($\{G_k\}_n$) are extensions to a set of goals (used quantified) | | |

A goal-oriented taxonomy T is said to be correct and complete if it satisfies the conditions below. C1 ensures that decomposition of software package types is well-formed, C2 that the taxonomy provides a unique way for classifying software packages and C3 that software packages can always be classified using the taxonomy:

**C1.** *Parent-child correctness*. The goal of each node is implied by its parent goal:

$$\forall X: \text{belongs}(T, X): [\forall Y: Y \in \text{children}(T, X): \text{Goal}(X) \Rightarrow \text{Goal}(Y)]$$

**C2.** *Siblings correctness*. The goals of siblings are disjoint:

$$\forall X, Y: X \neq Y \wedge \text{belongs}(T, X) \wedge \text{belongs}(T, Y) \wedge (\exists A: \text{belongs}(T, A): \text{parent}(T, A, X) \wedge \text{parent}(T, A, Y)): \neg\, \text{Goal}(X) \wedge \text{Goal}(Y)$$

**C3.** *Completeness*. The goals of siblings cover altogether the goal of their parent:

$$\forall X: \text{belongs}(T, X) \wedge \neg\, \text{leaf}(T, X): [\text{Goal}(X) \equiv \cup Y: Y \in \text{children}(T, X): \text{Goal}(Y))]$$

Given these correctness and completeness notions, we can define a process for rearranging a goal-oriented taxonomy: first we remove conflicts among parents and children to ensure C1, next we detect and solve conflicts among siblings to ensure C2 and afterwards we complete the taxonomy identifying new nodes that fulfill the parent goal to ensure C3. We add a fourth step, once we have a correct and complete goal-oriented taxonomy, to tailor the result to the particular (and subjective) taste of the designer with respect to level of detail. Transformation rules are used in each step to progress towards the result.

## 3 A Set of Transformation Rules for Manipulating Taxonomies

In this section we present the transformation rules that are used when restructuring taxonomies. Taxonomy manipulation is goal-based taking into account the properties of correctness and completeness of taxonomies stated in section 2. There are 8 transformation rules that are applied into the 4 identified steps; some rules apply to more than one step. Some of the rules may take slightly different forms, and we have

therefore some different variations of the basic idea which are given different names. The rules are specified by pre and postconditions using the predicates and functions on taxonomies and goals introduced in section 2; in postconditions, the expression *x@pre* stands for the value of *x* before applying the rule. Due to lack of space, the predicates on the form of the tree are not written explicitly, they can be easily inferred from Fig. 1, which shows graphically the transformation rules. In each step we assume as invariants the conditions C*i* (see section 2) that have been ensured in the previous step that must be considered implicitly as part of the preconditions and postconditions of the rules that apply in this step. Last, we get rid of renamings, we assume that every rule has the right to change the name of the involved nodes for legibility.

### 3.1 Transformation Rules in Step 1

When a node B is found such that its goal is not implied by its parent's goal, we have basically two options: to move B to another part of the taxonomy T or else to remove it. We introduce therefore two rules, *Reallocation* and *Removal*.

**R1. Reallocation(T, A, B, X)** Changes the current position of B (and its successors), because its goal violates C1with respect to its parent's goal A. The new parent position will be X.
- *Precondition*: $\neg$ Goal(A) $\Rightarrow$ Goal(B)

There are several variations of reallocation, each refining slightly this initial contract:

– B's goal is implied by some other node X's goal in the taxonomy T, which is the destination of the reallocation. We call this variant *Strict Reallocation* and it is characterized by the fact that its precondition is further enlarged:

**R1.1. Strict Reallocation(T, A, B, X)**.
- *Precondition*: $\neg$ Goal(A) $\Rightarrow$ Goal(B) $\wedge$ Goal(X) $\Rightarrow$ Goal(B)

– B's goal is not implied by any other node's goal in the taxonomy T, but there is a relationship with node X's goal, which is the destination of the reallocation. We represent this relationship by the concept of soft implication ($\pm\Rightarrow$, see section 2). In this case, we have three more subcases:

a) X's goal is left as it is, which means that we need further rules to be applied on X and B to reach a correct state. In other words, this reallocation is just a first move in a multi-rule movement. We call this *Unfinished Reallocation*.

**R1.2. Unfinished Reallocation(T, A, B, X)**.
- *Precondition*: $\neg$ Goal(A) $\Rightarrow$ Goal(B) $\wedge$ Goal(X) $\pm\Rightarrow$ Goal(B)

b) X's goal is enlarged, to capture the part of B's goal that is not implied. Of course, this increment of satisfaction must be propagated to all X's ancestors. This variant is called *Enlarged Reallocation*.

**R1.3. Enlarged Reallocation(T, A, B, X)**.
- *Precondition*: $\neg$ Goal(A) $\Rightarrow$ Goal(B) $\wedge$ Goal(X) $\pm\Rightarrow$ Goal(B)
- *Postcondition*: Goal(X) = Goal@pre(X) $\cup$ Goal(B) $\wedge$
  $\forall Y \in$ ancestors(T, X): Goal(Y) = Goal@pre(Y) $\cup$ Goal(B)

c) B's goal is narrowed, to discard the part of the goal that is not implied by X's goal. Of course, this narrowing must be propagated to all B's successors. Thus, we obtain the *Narrowed Reallocation*.

**R1.4. Narrowed Reallocation(T, A, B, X)**.
- *Precondition*: $\neg$ Goal(A) $\Rightarrow$ Goal(B) $\wedge$ Goal(X) $\pm\Longrightarrow$ Goal(B)
- *Postcondition*: Goal(B) = Goal@pre(B) $\cap$ Goal(X) $\wedge$
  $\forall Y \in$ successors(T, X): Goal(Y) = Goal@pre(Y) $\cap$ Goal(B)

In the case of removal, in this step we consider a special case of the general *Removal* rule, called *Hard Removal* (for distinguishing it from the type of removal introduced later) that is applied when the node is not related neither to its parent goal, nor to other nodes, nor the overall taxonomy goal.

**R2. Removal (T, A, B)**. Deletes the B node (and its successors) from the taxonomy T.

**R2.1. Hard Removal (T, A, B)**
- *Precondition*: $\neg$ Goal(A) $\Rightarrow$ Goal(B)

### 3.2 Transformation Rules in Step 2

When some siblings $\{C_k\}_n$ that are children of A are found such that their goals are not disjoint, we have basically three options:

– To remove repeatedly until no overlapping exists, applying the *Soft Removal* rule such that an element B of these $\{C_k\}_n$ (and its successors) is removed each time.

**R2.2. Soft Removal(T, A, B)**.
- *Precondition*: $\exists$C: parent (T,A,C): Goal(B) $\wedge$ Goal(C)

– To merge them to hide the overlapping part applying a particular case of the Merge rule, called *Inclusive Merge*.

**R3. Merge (T, A, $\{C_k\}_n$, B)**. Merges the nodes $\{C_k\}_n$ into one called B. The new node's goal is the union of siblings' goals. The children of the merged nodes become children of the new goal. It is important to stand out that these new children are in fact siblings and so new conflicts may arise, to be dealt with later in step 2.
- *Postcondition*: Goal(B) = $\cup$ k: 1 ≤ k ≤ n: Goal($C_k$)

**R3.1 Inclusive Merge(T, A, $\{C_k\}_n$, B)**
- *Precondition*: $\cap$ k: 1 ≤ k ≤ n: Goal($C_k$) ≠ $\varnothing$

– To keep them extracting the common part applying the *Split* rule. This rule has a slightly difficult contract. It takes the common part of the set of siblings $\{C_k\}_n$ and makes a new node B with it. In the precondition, we must make sure that C1 is not violated after applying the rule. This could happen if for some child of some of the involved nodes, its goal is not implied neither by its current parent after the operation, nor by the new node; in other words, the goal of the child could be partially covered by the old and new nodes.

**R4. Split (T, A, $\{C_k\}_n$, B)**.
- *Precondition*: $\cap$ k: 1 ≤ k ≤ n: Goal($C_k$) ≠ $\varnothing$ $\wedge$
  $\forall X \in$ children(T, $C_k$): Goal(X) $\Rightarrow$ ($\cap$ k: 1 ≤ k ≤ n: Goal($C_k$)) $\vee$
  ($\cap$ k: 1 ≤ k ≤ n: Goal($C_k$)) $\Rightarrow$ Goal(X)

- *Postcondition*: $\text{Goal}(B) = \cap\, k: 1 \le k \le n: \text{Goal}(C_k) \wedge$
  $\forall k: 1 \le k \le n: \text{Goal}(C_k) = \text{Goal}(C_k)@\text{pre} - \text{Goal}(B) \wedge$
  $\forall k: 1 \le k \le n: \forall X \in \text{children}@\text{pre}(T, C_k):$
  $(\text{Goal}(B) \Rightarrow \text{Goal}(X)) \Rightarrow \text{parent}(T, B, X) \wedge$
  $(\text{Goal}(C_k) \Rightarrow \text{Goal}(X)) \Rightarrow \text{parent}(T, C_k, X)$

### 3.3 Transformation Rules in Step 3

Step 3 is oriented to reach the state C3. When the decomposition of a node is such that its children do not cover its goal, the only behavior that applies is to create new nodes covering the part of goal left. Thus, we only can to apply the *Identification* rule adding as preconditions the incompleteness of parent's goal and that the new node's goal will not violate the states C1 and C2 ensured in the two previous steps.

**R5. Identification(T, A, B)**. Inserts a new node as child of an existing one whose goal is not fully covered by its children's goals. The new node's goal must not violate neither C1 nor C2

- *Precondition*: $\text{Goal}(A) \Rightarrow \text{Goal}(B) \wedge$
  $(\forall X: X \in \text{children}(A): \neg\text{Goal}(X) \wedge \text{Goal}(B)) \wedge$
  $\neg\, (\text{Goal}(A) \equiv \cup\, X: X \in \text{children}(A): \text{Goal}(X))$

### 3.4 Transformation Rules in Step 4

Step 4 just restructures the taxonomy once it has been proven correct with the aim of leveraging the incoming taxonomy. In a few words, the rationale for changing the form of a correct taxonomy is:

– a leaf A is too abstract (i.e., its attained goal is too coarse-grained, mixing different concepts) and should be decomposed into $\{C_k\}_n$ nodes to add detail. Thus, *Division* rule should be applied:

**R6. Division(T, A, $\{C_k\}_n$)**. Breaks a node A into several descendants. Relationships among the new nodes' goals and the divided node's goal shall ensure that C1, C2 and C3 are preserved.

- *Precondition*: $\forall k: 1 \le k \le n: \text{Goal}(A) \Rightarrow \text{Goal}(C_k) \wedge$
  $\forall j, k: 1 \le j < k \le n: \neg(\text{Goal}(C_j) \wedge \text{Goal}(C_k)) \wedge$
  $\text{Goal}(A) \equiv \cup\, k: 1 \le k \le n: \text{Goal}(C_k)$

– the children of a node A do not really add value to the taxonomy and must be either removed applying *Pruning* rule or merged (the variant used in this step is called *Non-inclusive Merge* a weaker version of the merge of step 2):

**R7. Pruning(T, A)**. Eliminates the children (and all its successors) of an intermediate node A.

**R3.2 Non-inclusive Merge (T, A, $\{C_k\}_n$)**

- *Postcondition*: $\text{Goal}(B) = \cup\, k: 1 \le k \le n: \text{Goal}(C_k)$

– the conceptual gap among a node A and some of its children is too wide, resulting in a too flat hierarchy, and an intermediate node with a new goal must be introduced using *Abstraction rule*:

**R8. Abstraction(T, A, $\{C_k\}_n$, B)**. Creates a new node B as parent of a set of existing ones $\{C_k\}_n$, which becomes child of their parent A. The new node's goal is the union of the goals of the nodes in the set. The children of the merged nodes become children of the new goal.

- *Postcondition*: Goal(B) = $\cup$ k: 1 ≤ k ≤ n: Goal($C_k$)

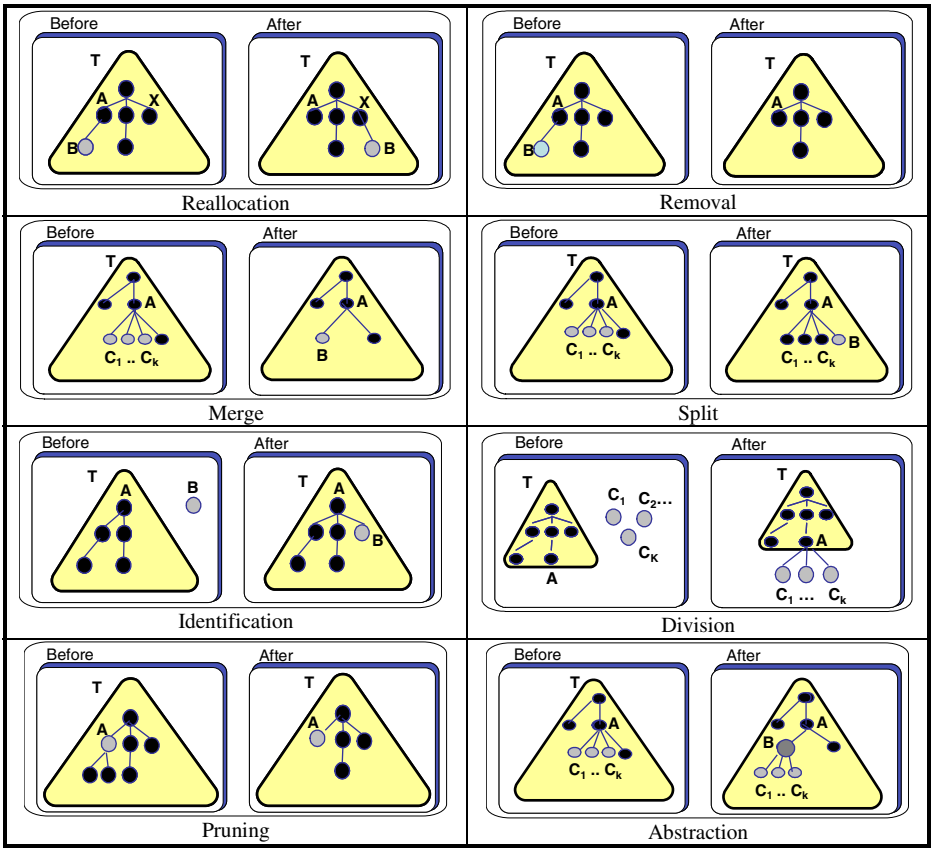


**Fig. 1.** Set of Transformation Rules

### 3.5 Final Remarks

As a summary, table 3 presents the transformation rules according to the step in which they can be used. It is not difficult to demonstrate both the correctness of the rules and the termination of each step. For correctness, it can be shown that, at each step, the rules preserve the conditions that apply. Our style in writing the rules (preconditions, postconditions, and invariants) makes these proofs easier. For termination, we can define metrics whose value is not incremented by the application of the rules. For step 1, the metrics counts the number of pairs parent-child that violate the goal implication rule. Each rule applied decrements by 1 or does not modify the value. For step 2, it is not so easy because individual merges can generate more conflicts among children

than the one that is being solved; therefore, we need a more elaborated metrics in which not just the number but also the position at the tree (more precisely, the level) is taken into account. For step 3, and identification either decrements by 1 or does not modify the metrics' value. Step 4 has not an invariant to be ensured, so we do not need termination condition, the step may finish at any moment.

**Table 3.** Summary of transformation rules and their applicability to the 4 steps

| *Step* | *Reallocation* | *Removal* | *Split* | *Merge* | *Identification* | *Division* | *Pruning* | *Abstraction* |
|---|---|---|---|---|---|---|---|---|
| **1** | R1.1, R1.2, R1.3, R1.4 | R2.1 | | | | | | |
| **2** | | R2.2 | R3 | R4.1 | | | | |
| **3** | | | | | R5 | | | |
| **4** | | | | R4.2 | | R6 | R7 | R8 |

## 4 An Example: A Taxonomy for Business Applications

In this section, we apply our goal-based transformation rules for restructuring the classification of Business Applications (BA) –i.e. products that are used in the daily functioning of all types of organizations worldwide– proposed by Gartner [3]. We only expose the process of rearranging the hierarchy, assuming that the process of discovering goals and binding them to the departing classification has been performed before. We focus on a particular part of the original Gartner BA classification, the subtree bound to Supply Chain Management (see Fig. 2). Due to lack of space, we leave one of its nodes (Supply Chain Planning) out of our study. We complement the figure with the goals of the nodes that we are going to manage in our process (see table 4).

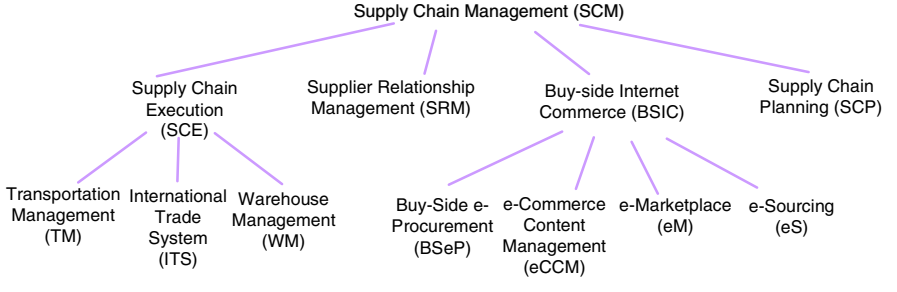


**Fig. 2.** An excerpt of the BA Gartner classification

Taking into account the form of the tree and the attached goals, table 5 summarizes the goal-based reasoning and transformation rules applied in each step of our example, whilst Fig. 3 shows the taxonomy after each of the 4 steps. The relationships among goals that violate some condition are stated in the second column. The other two columns state the rule that is applied at every moment. The relationships among goals require knowledge on the domain, enough as to discern for instance that in the

first step we discover that the goal of SCE is not oriented to automate business processes, such that we can infer that G(SCE) does not imply G(ITS).

**Table 4.** Goals boud to some nodes of the BA taxonomy

| *Node Names* | *Goal Attained* |
|---|---|
| Supply Chain Management (SCM) | To encompass the process of creating and fulfilling demands of the market for goods and services, mainly in 2 categories: execution and planning |
| Supply Chain Execution (SCE) | To manage relationships with the supplier (sourcing and procurement), and manufacture and logistics aspects |
| Transportation Management (TM) | To manage all freight deliver activities across the enterprise |
| International Trade Systems (ITS) | To automate the import/export business process |
| Warehouse Management (WM) | To manage the operation of a warehouse or distribution center |
| Supplier Relationship Management (SRM) | To manage enterprise interactions with the organizations that supply goods and services that are used. |
| Buy-Side Internet Commerce (BSIC) | To manage the internet by-side commerce |
| eCommerce Content Management (eCCM) | To address how to take unstructured product content and create and manage structured data on internet |

**Table 5.** Transforming the original Gartner classification into a goal-based taxonomy

| *Step* | *Goal Reasoning* | *Rule Applied* | |
|---|---|---|---|
| 1 | $\neg G(BSIC) \Rightarrow G(eCCM)$ | 2.1 | Hard Removal(BA, BSIC, eCCM) |
| | $\neg G(BSIC) \Rightarrow G(eM)$ | 2.1 | Hard Removal(BA, BSIC, eM) |
| | $\neg G(BSIC) \Rightarrow G(BSeP) \wedge G(SRM) \Rightarrow G(BSeP)$ | 1.1 | Strict Reallocation(BA, BSIC, BSeP, SRM) |
| | $\neg G(BSIC) \Rightarrow G(eS) \wedge G(SRM) \Rightarrow G(eS)$ | 1.1 | Strict Reallocation(BA, BSIC, eS, SRM) |
| | $\neg G(SCM) \Rightarrow G(BSIC)$ | 2.1 | Hard Removal(BA, SCM, BSIC) |
| | $\neg G(SCE) \Rightarrow G(ITS)$ | 2.1 | Hard Removal(BA, SCE, ITS) |
| 2 | $G(SCE) \cap G(SRM) \neq \varnothing$ | 3.1 | Inclusive Merge(BA, SCM, {SCE, SRM}) |
| 3 | $\neg$ (G(SCE) ≡ G(BSeP)∪G(eS)∪G(WM)∪G(TM)) Left goals: Manufacturing Management (MM), Not-eProcurement (eP), Not-eSource (NeS) | 5<br>5<br>5 | Identification(BA, SCE, MM)<br>Identification(BA, SCE, NeP)<br>Identification(BA, SCE, NeS) |
| 4 | Logistics Management (LM)<br>Sourcing (S)<br>Procurement (P)<br>Supplier Interactions(SI) | 8<br>8<br>8<br>8 | Abstraction(BA, SCE, {WM, DM}), creates LM<br>Abstraction(BA, SCE, {eS, NeS}), creates S<br>Abstraction(BA, SCE, {eP, NeP}), creates P<br>Abstraction(BA, SCE, {S, P}), creates SI |

We have rearranged the whole Gartner BA taxonomy. It originally had 96 nodes, 78 were leafs (i.e., types of software packages) and just 18 intermediate nodes. The longest path from the root to a leaf was of length 6 (just one, and two of length 5), whilst the maximum width was 10 siblings (average width 4,33). Our process resulted in a taxonomy of 188 nodes, 120 of them leafs. The tree changed its form, with a

longest path of length 8 (in fact, 16 paths of this length and 21 of length 7) and 7 siblings at most (average width 3,18). As a result, not only we provided rationale for the taxonomy, but also we let it well-suited for our main purpose, to use it for software package selection, with better defined ways to reach a leaf (i.e., a type of software package) from the root having less alternatives to consider at each step.

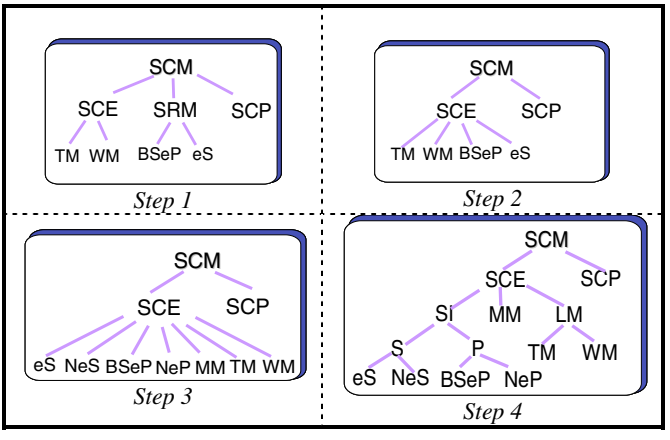


**Fig. 3.** The 4-Step Taxonomy construction process applied to an excerpt of the BA case

## 5 Conclusions

In this paper we have presented a proposal for building taxonomies that arrange the existing myriad of software packages into categories. The proposal is build upon the notion of goal.

We have identified 8 transformation rules specified by contracts that are used along a 4-phase process to manipulate an existing departing taxonomy to obtain the target one. As a result, and this is the main contribution of our work, we obtain a high-quality taxonomy in which the rationale for the classification is very clear and correctness and completeness are ensure by construction. As far as we know, this rationale distinguishes our approach of other taxonomy proposals as the ones mentioned in the introduction.

This work is a continuation of [19], in which we just presented a taxonomy in the form of decision tree for business application. In this paper we have improved our proposal by making it domain-independent, by defining precisely the rules and the process to use them, and by using the more general notion of goal instead of classifier (as done in decision trees).

We have demonstrated the feasibility of our approach in one of the most populated and critical fields of the software package marketplace (business applications). In particular, we have solved some problems observed in the departing hierarchy: categories whose reason-to-be was not clear, categories that were not defined precisely (as a consequence, their granularity was not always adequate); categories that overlapped; the criteria for decomposing categories was never declared and often

was not evident, making hard the use of the hierarchy; levels of abstraction were different at different parts of the hierarchy; and so on.

Different actors may benefit from our approach:

- Software consultant companies offering assessment for business automation may structure their services better.
- Medium- and large-size companies with their own IT department may be more confident on their own selection.
- Software engineers which usually carry out package selection may structure better their knowledge and may aim at a better return of investment.

## References


1. D. Carney, F. Long. "What Do You Mean by COTS? Finally a Useful Answer". *IEEE Software*, 17 (2): 83-86, March/April 2000.
2. B. Craig Meyers, P. Oberndorf. *Managing Software Acquisition*. SEI Series in Software Engineering, 2002.
3. Gartner Inc. web page. www.gartner.com, last accessed February 2005.
4. Forrester Research Inc. web page. www.forrester.com, last accessed February 2005.
5. ComponentSource web page. www.componentsource.com, last accessed February 2005.
6. Genium Software Development web page. http://www.genium.dk/index.xml, last accessed February 2005.
7. INCOSE web page. www.incose.org, last accessed February 2005.
8. ITPapers web page. www.itpapers.com, last accessed February 2005.
9. eCOTS web page. www.ecots.org, last accessed February 2005.
10. IT product guide web page. http://productguide.itmanagersjournal.com, last accessed February 2005.
11. CBSE web page. www.cbsenet.org/pls/CBSEnet/ecolnet.home, last accessed Feb. 2005.
12. E. Arranga. “Cobol Tools: Overview and Taxonomy”. *IEEE Software*, 17(2): 59-61, 2000.
13. R. Glass and I. Vessey, "Contemporary Application-Domain Taxonomies". *IEEE Software*, 12(4): 63-76, 1995.
14. SWEBOK web page. www.swebok.org, last accessed February 2005.
15. A.I. Antón. “Goal-Based Requirements Analysis”. In *Proceedings 2nd IEEE International Conference on Requirements Engineering* (ICRE), Colorado Springs (USA), 1996.
16. C. Potts, K. Takahashi, A.I. Antón. “Inquiry-Based Requirements Analysis”. *IEEE Software*, 11(2), March 1994.
17. J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kauffman, 1993.
18. L. Breiman, J.H. Friedman, R.A. Olshen, C. J. Stone. *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group, 1984.
19. J.P. Carvallo, X. Franch, C. Quer, M. Torchiano. “Characterization of a Taxonomy for Business Applications and the Relationships Among Them”. In *Proceedings of 3rd International Conference on COTS-Based Software Systems* (ICCBSS), LNCS 2959, 2004.