

# Using Composition Trees to Model and Compare Software Process

Lian Wen<sup>1</sup>, David Tuffley<sup>1</sup>, Terry Rout<sup>1</sup>,

<sup>1</sup>Software Quality Institute, Griffith University, Brisbane,  
Queensland, Australia  
{l.wen, d.tuffley, t.rout}@griffith.edu.au

**Abstract.** Software processes described by natural languages are frequently ambiguous and it is usually difficult to compare the similarity and difference between one process defined in one standard and its counterpart defined in another standard. This paper proposes Composition Tree (CT) as a graphic language to model software process based on its purpose and expected outcomes. CT is a formal graphic notation originally designed for modeling component based software system. This paper demonstrates that CT can be a powerful notation to give a clear and unambiguous description of a software process as well. This paper also investigates an algorithm which can compare two CT-modeled processes and provide an intuitive view called a Comparison Composition Tree (CCT) to highlight the differences and similarities between the two processes.

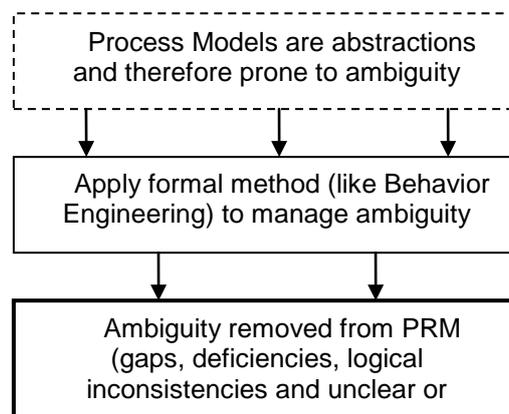
**Keywords:** Software Process, Behavior Engineering, Composition Tree, Process Reference Model

## 1 Introduction

Process models are abstract representations of a process architecture, design or definition [6]. They are abstractions, not direct representations of reality. The language that people use when developing these abstractions, these process models, is prone to ambiguity due to the fallible way in which people use language. George Box famously observed that all models are wrong but some are useful [1]. Models are simplifications of reality and in the process of simplifying, essential information might be left out resulting in ambiguity. Even with ambiguity, models can be useful, but this ambiguity points to a need for a way to reduce or eliminate it. Such a way might be found in a formal method such as Behavior Engineering.

In seeking a solution to the problem of ambiguity in process modeling, one sees a similar problem with the requirement specifications for software systems. Ambiguous language, incomplete descriptions, repetition and redundancies in the way specifications are expressed inevitably leads to sub-optimal project outcomes (systems that do not meet the user's needs). Behavior Engineering [3] successfully addresses the problems faced by software developers seeking to translate a set of user requirements into a complete and consistent requirements specification.

Behavior Engineering uses a formally-grounded graphical notation with the capability to represent a wide range of system behaviors in unambiguous terms. Its strength is its ability to accommodate complexity and detail, ease of use, and in particular for this project its ability to expose defects.



**Fig. 1.** Use formal method to remove ambiguity from abstract model

Previous research indicates that BE notations can be useful verification tools for process modeling [13]. This paper refines this concept by proposing a detailed scheme to model a software process based on its purpose and process outcomes in a Composition Tree (CT) [5], which is one of the key parts of

the BE notations. The graphic version of a process model is more intuitive, less ambiguous and easier to verify than the original natural language described process.

With the quick development and diversity of software standards for different domains [12], systematic methods of comparison of software processes are crucial for process analysis, understanding and evolution [10]. However, it is difficult to consistently, systemically and automatically compare two processes if they are described in natural languages.

This paper proposes a formal method which can compare two processes when they are modeled in CTs. This method is based on a precisely defined tree merging algorithm [14]; therefore it can be automated.

The proposed comparison method generates a Comparison Composition Tree (CCT) that explicitly shows the difference and similarity of the two compared processes. In particular, the CCT highlights the difference in a way that is easy to read and understand, so it can be very useful for people to study the evolution of processes.

The paper is organized in the following way: Section 2 and Section 3 provide necessary background information about Process Models and Composition Trees respectively; Section 4 introduces the method to use a CT to model a process; Section 5 describes the algorithm to compare two CTs; Section 6 demonstrates this comparison method through a case study that compares processes for configuration management in two different standards; finally, a brief conclusion is given in Section 7.

## 2 Software Process Models

Feiler and Humphrey [6] define a process model as an *abstract representation of a process architecture, design or definition*. Process models in this broad sense can be seen as process elements at an architectural, design and definitions level. The abstraction inherent in process models serves to capture and represent the essential nature of processes. Any representation of the process can be said to be a process model. Process models can be analyzed, validated, and if enactable can simulate the modeled process [6].

Scacchi [11] distinguishes software process models from software lifecycle models. The former are descriptive or prescriptive characterizations of how software is developed, whereas the latter *represent a networked sequence of activities, objects, transformations, and events that embody strategies for accomplishing software evolution* [11]. This definition is not inconsistent with that of Feiler and Humphrey [6] discussed above. Process models are useful for developing more precise and formalized descriptions of software life cycle activities, using a rich notation, syntax, or semantics, often suitable for computational processing [11]. This idea lends support for the use of Behavior Engineering [4] and its notation that might be accurately described as *rich notation, syntax, or semantics* to develop Process Reference Models.

ISO/IEC 24774:2007 - *Software and systems engineering -- Life cycle management -- Guidelines for process description* [9] outlines a standard format for any process reference model, including those intended for process implementation and process assessment. This general purpose standard outlines the elements used to describe a process; title, purpose statement, outcomes, activities and tasks.

- The **title** conveys the scope of the process as a whole, expressed as a short noun phrase that summarize the scope of the process, identify the principal concern of the process, and distinguishes it from other processes within the scope of a process model.
- The **purpose** describes the goal of performing the process. It is expressed as a high level goal for performing the process, preferably stated in a single sentence. The implementation of the process should provide measurable, tangible benefits to the stakeholders through the expected outcomes
- The **outcomes** express the observable results expected from the successful performance of the process. Outcomes are expressed in terms of a positive, observable objective or benefit. The list of outcomes associated with a process shall be prefaced by the text, 'As a result of successful implementation of this process:' The outcomes should be no longer than two lines of text, about twenty words. The number of outcomes for a process should fall within the range 3 to 7. Outcomes should express a single result. The use of the word 'and' or 'and/or' to conjoin clauses should be avoided. Outcomes should be written so that it should not require the implementation of a process at any capability level higher than 1 to achieve all of the outcomes, considered as a group.
- The **activities** are a list of actions that may be used to achieve the outcomes. Each activity may be further elaborated as a grouping of related lower level actions;
- The **tasks** are specific actions that may be performed to achieve an activity. Multiple related tasks are often grouped within an activity.

ISO/IEC 24774:2007 [9] makes it clear that the outcomes should not go beyond what is stated in the purpose. There should be no capability level issues expressed in the outcomes.

Secondly the outcomes must address all of the issues that are apparent in the purpose statement. Nothing should be missed. The outcomes must therefore be necessary and sufficient to satisfy the purpose.

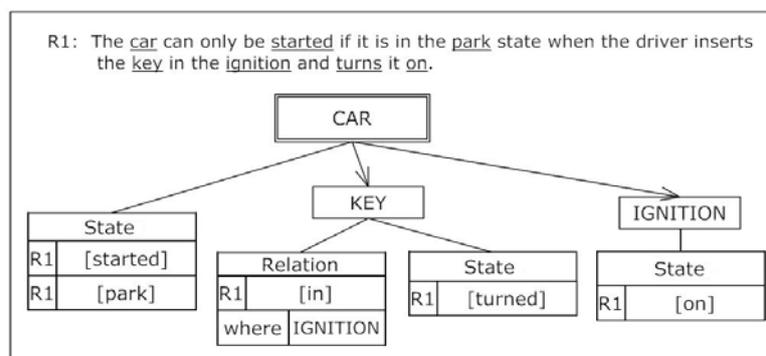
### 3 Composition Trees

A Composition Tree is originally used to describe the composition of a component based software intensive system [5]. It provides useful summary information including states, attributes and relationships about the system and other entities under the system.

Similar to the way of constructing a Behaviour Tree from the functional requirements [4], A Composition Tree can also be constructed through translating the individual functional requirements one by one. In this section, a small sized case study is used to explain both the notations of Composition Trees and also the procedure to build a Composition Tree from the functional requirements. In order to help readers to capture the concepts quickly, the introduction is more intuitive but less formal. A formal and more complete description of Composition Trees can be found at the Behaviour Engineering website [1].

**Case study:** a CAR system:

- R1: The car can only be started if it is in the park state when the driver inserts the key in the ignition and turns it on.
- R2: A dashboard light remains on if the driver's seatbelt is not fastened when the driver is seated and the ignition is on.
- R3: If the handbrake is on when the ignition is on, the brake-light turns on.
- R4: The security alarm is on when the car is locked, and if anyone tries to break in by breaking a window or forcing a door the alarm will sound.
- R5: When the driver, on approaching the car, presses the key-button it unlocks the door and turns the security alarm off.
- R6: When the car is unlocked the driver may get in and put the car into the park state.



**Fig. 2.** The Composition Tree (CT) generated from translating R1

Fig. 2 is the Composition Tree (CT) created from Requirement 1 (R1). This diagram shows the following information:

- There are two components KEY and IGNITION<sup>1</sup> under the system component CAR, which is drawn in doubled line.
- The CAR system has two different states “started” or “park”.
- The IGNITION has one state “on” and the KEY has one state “turned”.
- The KEY has one relation which is in the IGNITION.
- The requirement tag R1 helps people to trace the information in the diagram back to the original requirement.

<sup>1</sup> Some researchers may also model DRIVER as one component in the CAR system. However, this paper considers that the driver is something external to the system.

Please note that all the compositional information shown in Fig. 2 is faithfully translated from the original requirement including most of the terminology. However the information presented in the graph is more precise and less ambiguous.

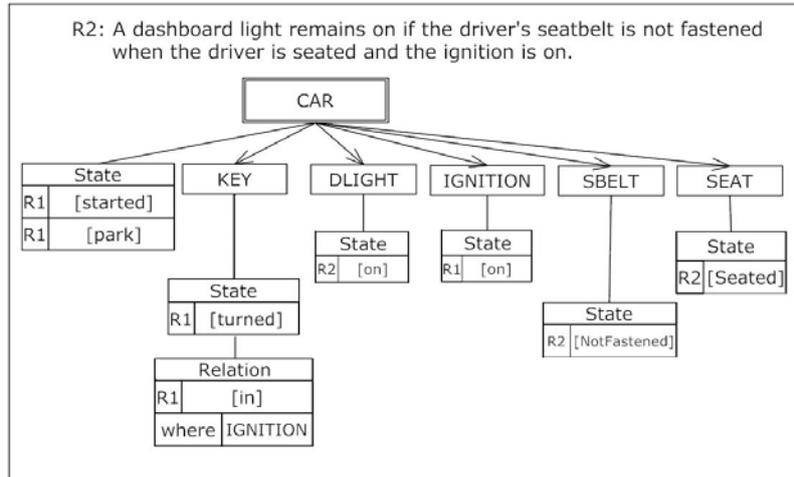


Fig. 3. Integrate the composition information from R2 into the Composition Tree.

Fig. 3 integrates the compositional information in R2 into the CT. The new tree may also be called an Integrated Composition Tree (ICT). However, in order to avoid unnecessary confusion, we will only use Composition Tree (CT) in this paper. The new CT shows the following addition information:

- There are three more components DLIGHT (dashboard light), SBELT (seatbelt) and SEAT<sup>2</sup>.
- The DLIGHT has state “on”, the SBELT has state “notfastened” and the SEAT has state “seated”.

A composition tree only captures the static compositional information of individual component. The dynamic, logical and cause and effect relationship between components is captured in Behavior Trees [4].

Based on the same process, an entire CT can be generated through integrating all the requirements as in Fig. 4.

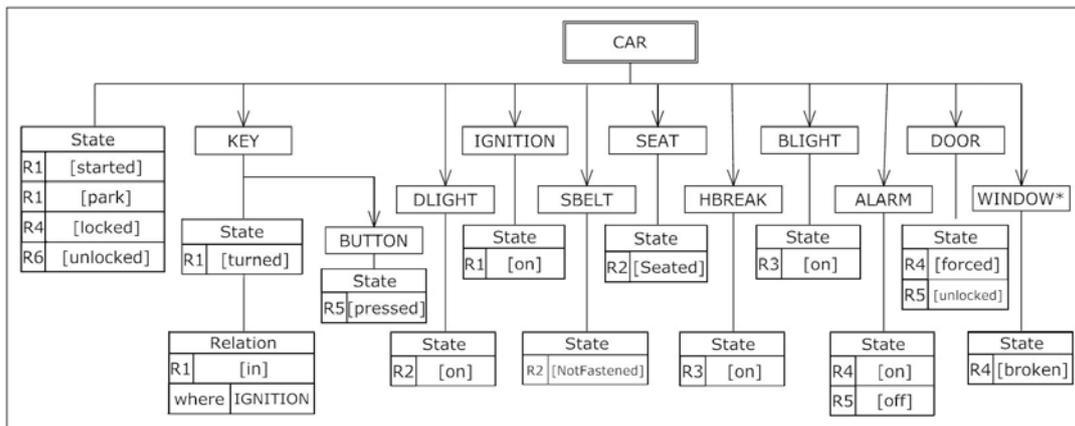


Fig. 4. The complete Integrated Composition Tree of the CAR system

Fig. 4 shows the complete list of the components in the CAR system, and the expected states and relations for each component. All the information is directly translated from the user requirements. The advantages of the CT over the natural language described functional requirement are:

- All information is integrated together so it is easy to identify the requirement defects. For example, the CT shows that many components have only one state. It's obvious that the original requirements are not completed because each component should have at least two different states.

<sup>2</sup> The SEAT component is not explicitly mentioned in R2, but based on the common understanding of a car, it is a reasonable interpretation of R2.

- A CT arranges the information about one component in one place. It will be easier for people to design and implement the components than the original requirements with the information of one component may be scattered all around the requirements.
- The more specific graphic notation is less ambiguous than the more flexible natural language.
- A CT removes the entire alias so it will use a consistent vocabulary for the system.

There are more discussions about the advantage of using CT in Software and System Engineering in elsewhere [5], which will not be repeated here. The focus of this paper is to investigate using CTs for Process Modeling.

#### 4 Using the Composition Tree approach to Model Process

According to ISO/IEC 24774:2007 [9], the standard elements to describe a process include the title, the purpose, outcomes, activities and tasks. Apart from the title, which is only the name of a process, purpose and outcomes are more static elements, so they may be more suitable to be modeled by composition trees.

This paper proposes a way to use a Composition Tree (CT) to model a software process based on its purpose and outcomes, which are usually documented in natural languages.

The way to construct a CT from the process purpose and outcomes includes the following steps:

1. Read through the purpose and outcomes, and make a complete and consistent list of nouns and acronyms, which are usually components or attributes of components.
2. Starting from the process purpose state, identify the components and their state and draw the initial CT.
3. Read each outcome one by one, to identify the components, states, relationship and attributes and then integrate the information in the CT.

This paper uses the Configuration Management process defined in ISO/IEC 12207:2000 [8] to explain the process.

**Process Name:** Software configuration management.

**Process Purpose:** *The purpose of the Configuration management process is to establish and maintain the integrity of the work products/items of a process or project and make them available to concerned parties.*

**Process Outcomes:**

1. *a configuration management strategy is developed;*
2. *items generated by the process or project are identified, defined and baselined;*
3. *modifications and releases of the items are controlled;*
4. *modifications and releases are made available to affected parties;*
5. *the status of the items and modifications are recorded and reported;*
6. *the completeness and consistency of the items is ensured; and*
7. *storage, handling and delivery of the items are controlled.*

In order to model this process, the first step is to identify and list all the components:

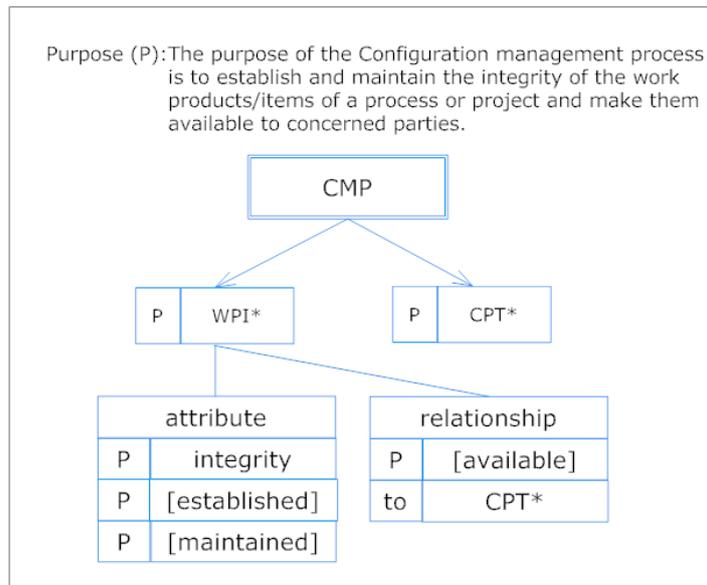
**CMP:** Software Configuration Management Process

**WPI:** Work product or item

**CPT:** Concerned Party

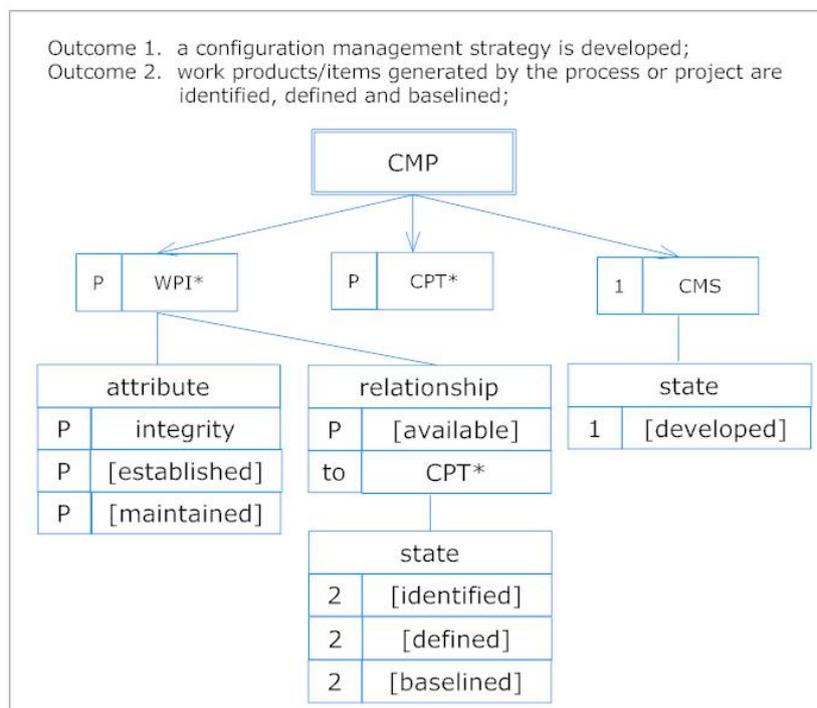
**CMS:** Configuration Management Strategy

The second step is to translate the process purpose into a composition tree as below: Fig. 5 shows that there are two different types of components under CMP, WPI and CPT; the “\*” sign indicates that the component may have more than one instance. The WPI has an attribute called integrity and the integrity needs to be established and maintained. There is also a relationship for WPI; the relationship is that WPIs should be available to CPTs. The tag “P” in each box means this piece of compositional information is translated from the purpose of the process.



**Fig. 5.** The CT constructed from the purpose of the Configuration Management process

The next step is to translate the outcomes one by one and integrate the compositional information into the CT. Fig. 6 shows the CT after outcome 1 and outcome 2 have been integrated. The compositions of a component include attributes, relationship and states; the three different compositions can be drawn directly under the component or one under another as in Fig.6 (the states of WPI are drawn under the relationship of WPI). There is no semantic difference, the variation is only for an easy to read layout.



**Fig. 6.** The CT from the Process Purpose, Outcome 1 and Outcome 2

Eventually, when all the outcomes are translated and integrated into the CT, it will be a complete CT showing the purpose and the expected outcomes as in Fig. 7.

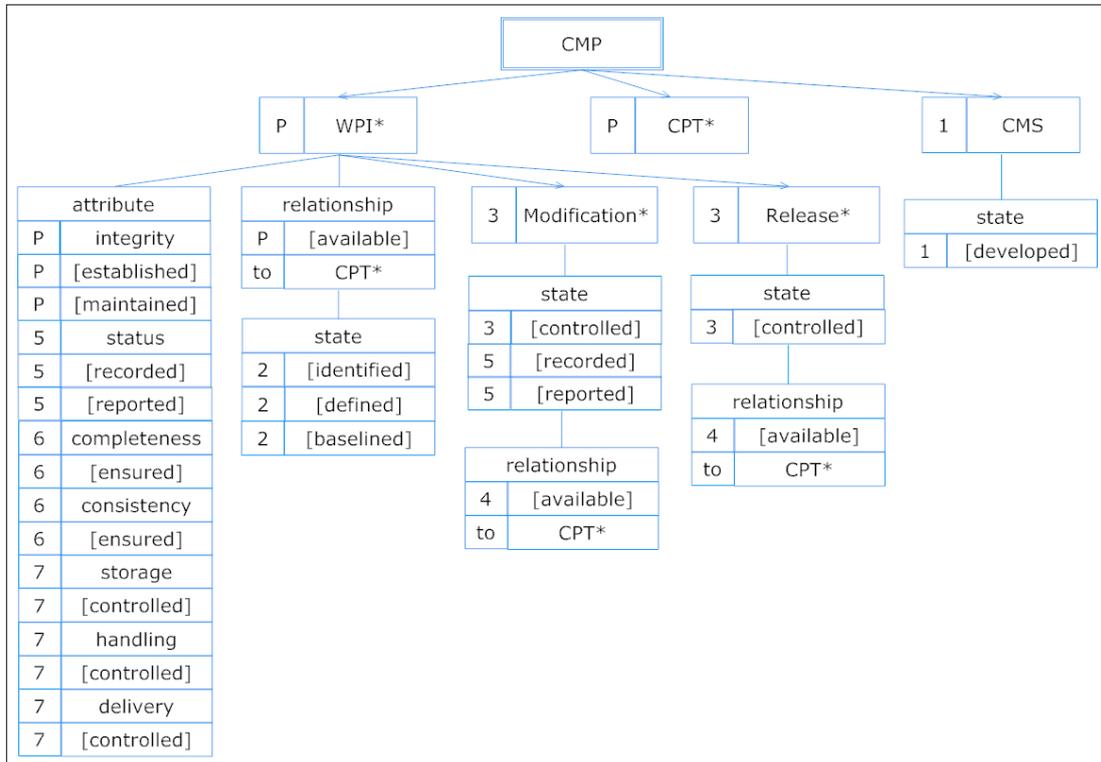


Fig. 7. The complete CT of the Configuration Management Process defined in 12207

Fig. 7 shows the Configuration Management Process in a CT. It has a few advantages over the initial natural language description:

1. All information is integrated in one graph, so the relationships between different parts become visible.
2. The information of each component is arranged in one place so it will be easier to retrieve. For example, from Fig. 7, it is easy to find that WPI has the following attributes: integrity, status, completeness, consistency, storage, handling and delivery.

Generally, the graphical version of the process may have less ambiguity, may be easier to understand and easier for people to identify process defects.

## 5 The Algorithm to Compare Two CTs

The previous section has explained how to use a CT to model a software process.

This section will introduce an algorithm which is to compare two CTs of two similar processes and identify the differences.

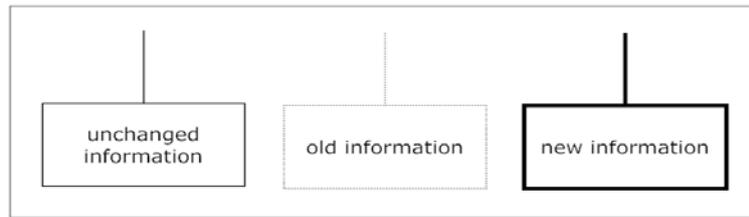
The comparison implements a label matching tree merging algorithm which has been used for comparing different versions of Behavior Trees [14]. However, as the notation of a composition tree is different from a behavior tree, there are some differences in the merging trees.

A critical task in tree merging algorithm is to identify the matching nodes. For CTs, the way to identify the same nodes is based on the name of component, state, etc. Therefore, before applying the merging algorithm, the first step is to identify the same component and/or same state which may be called by different names in the two compared trees and to establish a mapping between them. For example in Fig. 7, the component of work product and work item is called WPI but the same component may be called WP in another CT. In this situation, a mapping table is required.

The second step is to compare and merge the two trees. To simplify the discussion, we may call the first tree as the old tree and then the second tree as the new tree. In this way, the comparison procedure will create a merged tree that is called a Comparison Composition Tree (CCT). A CCT shows all the information of both trees and also highlights the difference in an easy to read way. To achieve this purpose, a display style convention is used in this paper as in Fig. 8.

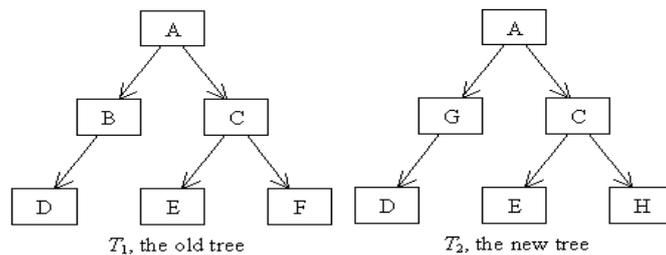
Under this display style convention, in a CCT, a piece of information which exists in both the old tree and the new tree is called unchanged and will be drawn in normal style; a piece of information if

only exists in the old tree will be called old and will be drawn in dotted lines; a piece of information if only exists in the new tree will be called new and will be drawn in bolded lines.



**Fig. 8.** The display style convention for a CCT

Now we will use a simple abstract example to explain the tree merging algorithm. Suppose that  $T_1$  and  $T_2$ , shown in Fig. 9, are the old CT and the new CT respectively.

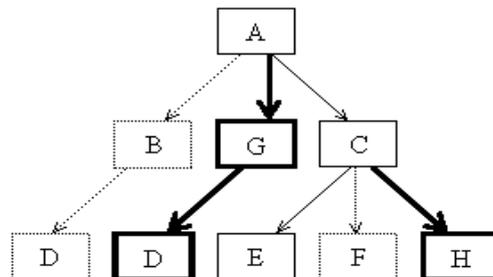


**Fig. 9.** The old CT  $T_1$  and the new CT  $T_2$

To compare  $T_1$  and  $T_2$  and generate the CCT, we use the following algorithm:

1. Start the comparison from the root nodes (in this example, node A). Because the root node exists in both trees, it is created in the CCT as an unchanged node.
2. Find the compared node's child-node set in both trees. (In this example, the child-node set in the old tree is {B, C} and the child-node set in the new tree is {G, C}.)
3. If a node exists in the old tree's child-node set but not in the new tree's child-node set, this node will be marked as an old node in the CCT. (In this example, B is such a node)
4. In the old tree, the sub trees under the old node will be generated in the CCT as old. (In this example, the node D under node B in  $T_1$  is such a case)
5. If a node exists in the new tree's child-node set but not in the old tree's child node set, this node will be created in the CCT as a new node. (In the example, G is such a node)
6. In the new tree, the sub trees under the new node will be generated in the CCT as new. (In this example, the node D under node G in  $T_2$  is such a case)
7. If a node exists in the child-node sets of both trees, it will be generated in the CCT as an unchanged node. (In the example, the node C is such a case)
8. An unchanged node will be a new comparison node and the algorithm will go back recursively to step 2.

The CCT  $T_c$  produced from  $T_1$  and  $T_2$  is shown in Fig. 10, following the style convention used in Fig. 8.



**Fig. 10.** The Comparison Composition Tree  $T_c$

## 6 Case Study

The previous section uses two abstract trees to explain the tree merging algorithm. This section will apply this algorithm on a case study based on real processes.

Section 4 has used a CT to model the Configuration Management process defined in ISO/IEC 12207 [8] (see Fig. 7). A similar Configuration Management process has also been defined in ISO/IEC 15288: 2002 [7] for system engineering. The next step is to model the Configuration Management process from 15288 and compare it with that from 12207.

The purpose and outcomes of the Configuration Management process in 15288 are quoted below:

**Purpose (P):** *The purpose of the Configuration Management Process is to establish and maintain the integrity of all identified outputs of a project or process and make them available to concerned parties.*

### Outcomes:

- A configuration management strategy is defined.
- Items requiring configuration management are defined.
- Configuration baselines are established.
- Changes to items under configuration management are controlled.
- The configuration of released items is controlled.
- The status of items under configuration management is made available throughout the life cycle.

Based on the purpose and outcome, following components are identified:

**CMP:** Configuration Management Process

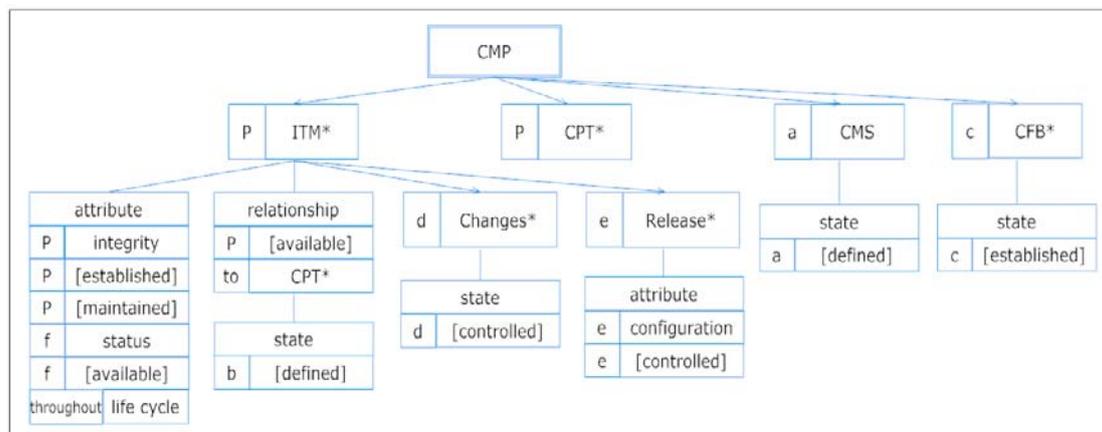
**ITM:** identified output / item

**CPT:** Concerned Party

**CMS:** Configuration Management Strategy

**CFB:** Configuration Baseline

The CT of the Configuration Management process defined in 15228 is drawn in Fig. 11.



**Fig. 11.** The CT of the Configuration Management process defined in 15228

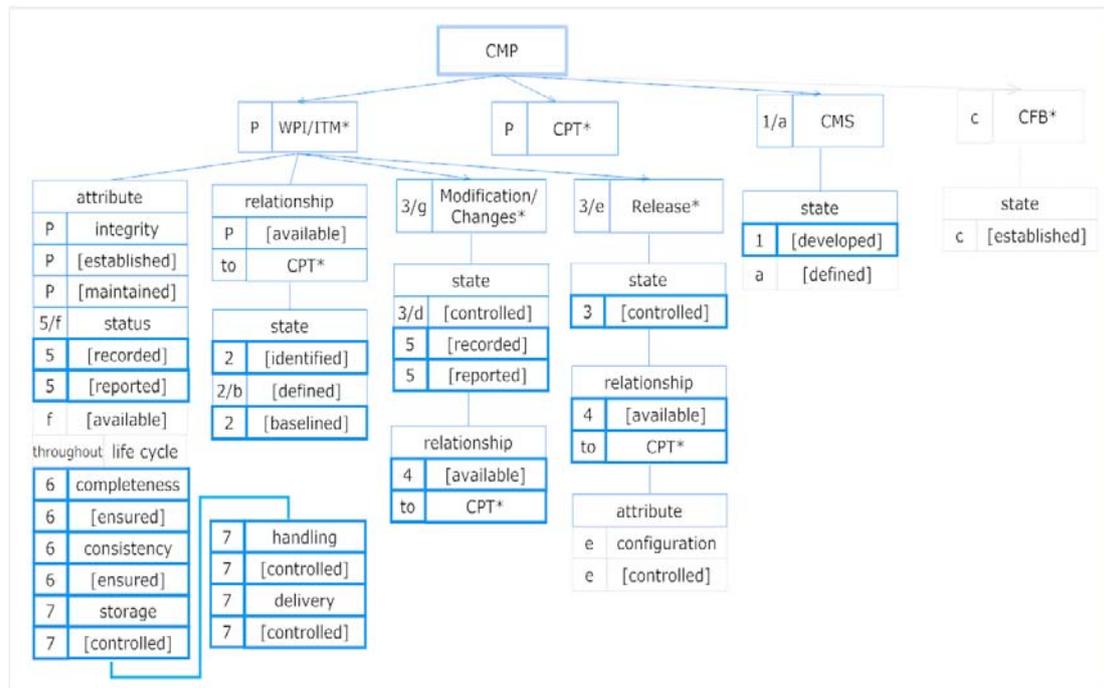
Considering that the CT in Fig. 11 is the old tree and the CT in Fig. 7 is the new tree, the comparing process will merge them together to generate a CCT. The first step is to create a list of mapping terms used in the two CTs. The list is shown in Table 1. After the mapping table has been established, the two CTs can be compared and merged into one CCT as in Fig. 12.

**Table 1.** The mapping terms in the two processes

#	15228	12207	Comments
1	ITM	WPI	Work product is a common term in software process, while item is a more general name.
2	Change	Modification	They are sub component order ITM/WPI

Fig. 12 shows the similarities and differences of the Configuration Management Process defined in both 12207 and 15228. There are a number of interesting things can be found in this CCT:

- There is component called Configuration Baseline (CFB) in 15228 but no such component mentioned in 12207. However, 12207 has mentioned that the WPI should be baselined.
- 12207 requires a configuration management strategy (CMS) to be developed, but 15228 asks a CMS to be defined. Can we assume that “developed” and “defined” mean the same thing regarding to a CMS?
- Both 12207 and 15228 have some requirements on the release of work product or items. However, 12207 asks the releases to be controlled and to be available to the concerned parties, while 15228 asks only the configuration of releases to be controlled.
- For the modifications (or changes) of a WPI/ITM, both standards ask them to be controlled, but 12207 also ask them to be recorded, reported and available to the concerned parties.
- Both standards ask WPI/ITMs to be available to the CPTs and defined, but 12207 has mentioned that the WPI/ITMs should also be identified and baselined.
- Both standards ask the integrity of WPI/ITM\* to be established and maintained.
- 12207 asks the status of WPI/ITMs should be recorded and reported, but 15228 only ask it to be available (throughout life cycle).
- 12207 has some requirements regarding to the completeness, consistency, storage, handling and delivery of WPIs but there is no corresponding requirements explicitly mentioned in 15228.



**Fig. 12.** The CCT between the Configuration Management Process of 12207 and 15228

Fig. 12 demonstrates how compositions can be used to highlight the similarity and difference between two processes. The advantage of a CCT is that it shows the similarity and difference between in a clear, complete and unambiguous way. This information can be helpful for people who are developing new process models. It also offers a tool of considerable strength to support the harmonization of process models drawn from different domains, but where the essential purpose of the process is the same.

## 7 Conclusions

The paper has proposed the formal graphic notation known as a Composition Tree (CT) as a viable way to verify processes and process model[5]. The graphic notation is intuitive and unambiguous and makes it easier to define semantics.

The paper extends the use of CTs by applying a tree merging algorithm [14] that compares the CTs of two similar processes to generate a Comparison Composition Tree (CCT). This displays the difference and similarity of two compared processes in a clear and user friendly way.

The results to date are promising. Even still at a preliminary stage, further researches including formal semantics of the process CT, automation tools and large case studies are on the way, the proposed method can be useful for people to study software processes as well as to design new processes. At this stage, application of the approach has been limited to the comparison of processes described using the same basic modeling approach (Process Reference Models). It has previously been demonstrated [13] that the approach can be employed to analyse other types of process model; it should therefore, be possible to generate CCT representations to compare models of the same process generated using different modeling approaches – for example, to compare a Process Reference Model with an Implementation Model and a Process Assessment Model addressing the same process, thus enabling validation across multiple models. This capability offers considerable benefits for developers of process models, whether such models are for intended for standardization, instruction or improvement purposes. Future research seems definitely warranted, and should focus on exploring the full capabilities of CT in relation to these issues.

## 8 References

1. Behavior Engineering Web Site, <http://www.behaviorengineering.org/>
2. Box, G.E.P., (1979). Robustness in the strategy of scientific model building, in *Robustness in Statistics*, R.L. Launer and G.N. Wilkinson, Editors. Academic Press: New York.
3. Dromey, R.G. (2006). Climbing Over the 'No Silver Bullet' Brick Wall, *IEEE Software*, Vol. 23, No. 2, pp.118-120.
4. Dromey, R.G., "Formalizing the Transition from Requirements to Design", in *Mathematical Frameworks for Component Software, Models for Analysis and Synthesis*, Chapter 6, World Scientific, 2006, Ed. Liu, Z., and He, J., ISBN 981-270-017-X, pp. 173-206
5. Dromey, R. G., *System Composition: Constructive Support for the Analysis and Design of Large Systems*, SETE-2005, Systems Engineering/Test and Evaluation Conference , Brisbane, Australia, 2005
6. Feiler P.H., Humphrey, W.S. (1992). *Software Process Development and Enactment*, Software Engineering Institute, Pittsburgh, CMU/SEY-92-TR-04 p 11
7. ISO/IEC 15288:2002, *Information technology - System engineering – System life cycle process*
8. ISO/IEC 12207:2008, – *Information technology – Software engineering – Software life cycle processes*
9. ISO/IEC TR 24774 (2007). *Software and systems engineering -- Life cycle management -- Guidelines for process description*.
10. Podorozhny, R. M., Perry, D.E. and Osterweil, L. J., *Artifact-based functional comparison of software processes*, 4th International Workshop on Software Process Simulation and Modeling, May 2003, pp V.29.1-10
11. Scacchi, W. (2001). *Process Models in Software Engineering*. Encyclopedia of Software Engineering, 2nd Edition, J.J. Marciniak (ed.) John Wiley and Sons, Inc, New York.
12. Sheard, S. A., *The frameworks quagmire, a brief look*, Proceedings of the 7th Annual International INCOSE, Symposium (INCOSE'97), 1997
13. Tuffley, D., Rout, T., *Behavior Engineering as Process Model Verification Tool*, The proceedings of the 10th International SPICE conference, 2010
14. Wen, L., Dromey, R.G., "From Requirements Change to Design Change: A Formal Path", Proceedings of the 2nd IEEE International Conference on Software Engineering and Formal Methods, pp. 104-113, 2004.