

APPLYING BEHAVIOR ENGINEERING TO PROCESS MODELING

David Tuffley, Software Quality Institute, Griffith University
Terry Rout, Software Quality Institute, Griffith University
Nathan, Brisbane, Qld. 4111, AUSTRALIA
d.tuffley@griffith.edu.au | t.rout@griffith.edu.au

***Abstract:** The natural language used by people in everyday life to express themselves is often prone to ambiguity. Examples abound of misunderstandings occurring due to a statement having two or more possible interpretations. In the software engineering domain, clarity of expression when specifying the requirements of software systems is one situation where absence of ambiguity is important. Dromey's (2006) Behavior Engineering is a formal method that reduces or eliminates ambiguity in software requirements. This paper seeks an answer to the question: can Dromey's (2006) Behavior Engineering reduce or eliminate ambiguity when applied to the development of a Process Reference Model?*

INTRODUCTION

Behavior Engineering has proven successful at reducing or eliminating the ambiguity associated with software requirements (Dromey, 2006). But statements of software requirements are not the only kind of artefact developed in the software engineering domain that need to be clear and unambiguous. Process Reference Models (PRM) is another category of software development artefact that might also benefit from being clear and unambiguous. A Process Reference Model is a set of descriptions of process entities defined in a form suited to the assessment and measurement of process capability. PRMs have a formal mode of expression as prescribed by ISO/IEC 15504-2:2003. PRMs are the foundation for an agreed terminology for process assessment (Rout, 2003).

The benefits of a method for achieving greater clarity are twofold: (a) PRM developers would gain from improving the efficiency of process model development, and (b) users of process models would benefit by achieving a clearer understanding of the underlying intention of a process which then serves as a consensus starting point for determining how a process might be applied in their own case. This paper therefore examines the ability of Behavior Engineering to disambiguate a particular PRM currently being developed.

This paper illustrates how Dromey's Behavior Engineering method (2006) can be used to disambiguate process models, making the resulting model clearer and easier to understand. It is suggested that this method has broader applicability in the Software and Systems Engineering domains. The paper examines in detail how Behavior Engineering has been applied in practice to a specific Process Reference Model developed by the authors. Before and after views are given of several process outcomes that had already passed through three previous reviews to remove ambiguity. The Behavior Engineering analysis results in evident improvements to clarity.

In a more general sense, it is suggested that this method may be helpful to the modelling of processes at both project and organisational levels, including project-level processes, high-level policy documents, and project agreements.

WHAT IS BEHAVIOR ENGINEERING?

Overview

Essentially, Behavior Engineering is a method for assembling individual pieces to form an integrated component architecture. Each requirement is translated into its corresponding 'behavior tree' which describes unambiguously the precise behaviors of this particular requirement (Glass, 2004). The 'tree' is built up from (a) components, (b) the states the components become, (c) the events and decisions/constraints associated with the components, and (d) the causal, logical and temporal dependencies associated with the component (Glass, 2004).

When each component is modelled in this way, and then integrated into a larger whole, a clear pattern of intersections becomes evident. The individual components fit together like a jigsaw puzzle to form a coherent component architecture in which the integrated behavior of the components is evident. One component establishes a precondition for another component to perform its function and so on. This allows a software system to be constructed *out of* its requirements, rather than merely satisfying its requirements (Glass, 2004).

Duplications and redundancies are identified and removed, for example, when the same requirement is expressed twice using different language in different places. Another benefit is that requirements traceability is managed with greater efficiency by creating traceable linkages between requirements as they move towards implementation.

Historical context

The practices now described as *behavior engineering* evolved from earlier work in which an approach for clarifying and integrating requirements for complex systems was developed (Dromey, 2001). This remains a significant application of the approach (Dromey, 2006); however, as the technique evolved, it became apparent that it could be applied to more general descriptions of systems behavior (Milosevic and Dromey, 2002). To date, some preliminary exploration of applying the technique to the analysis and validation of process models has been undertaken.

The OOSPICE Project (Stallinger et al, 2002) had the overall aim of improving time-to-market, productivity, quality and re-use in software development by focussing on the processes and technology of component-based software development (CBD). OOSPICE combined four major concepts of software engineering: CBD, object-oriented development, process assessment and software process improvement. Its objectives were the definition of a (a) unified CBD process metamodel, (b) a CBD assessment methodology, (c) resulting in component-provider capability profiles, and (d) a new CBD methodology and extensions to the ISO/IEC 15504 *Information Technology: Process Assessment. Joint Technical Committee IT-015, Software and Systems Engineering* (2005).

A key part of the OOSPICE was the definition of a model of coherent processes addressing the issues of component-based development; the process model was strongly aligned to ISO/IEC 12207 *Standard for Information Technology-Software Life Cycle Processes* (1998), but incorporated significant additional processes that specifically addressed CBD issues. The process model was developed following the approach of ISO/IEC 12207, with a series of structured activities and defined tasks. Additional detail on input and output work products was also specified for each activity.

The process model was examined using the behavior tree method in order to assess its consistency and completeness. The behavior tree analysis was highly successful; a total of 73 task level problems, 8 process level problems and numerous task integration problems were identified. In addition, examples were found where fragments of tasks were identified which subsequently have no integration point with the larger process tree – a weakness caused by unspecified or inconsistent task inputs or outputs.

An indicative example of the behavior tree method applied to a single process is shown below (explanation of notation given later):

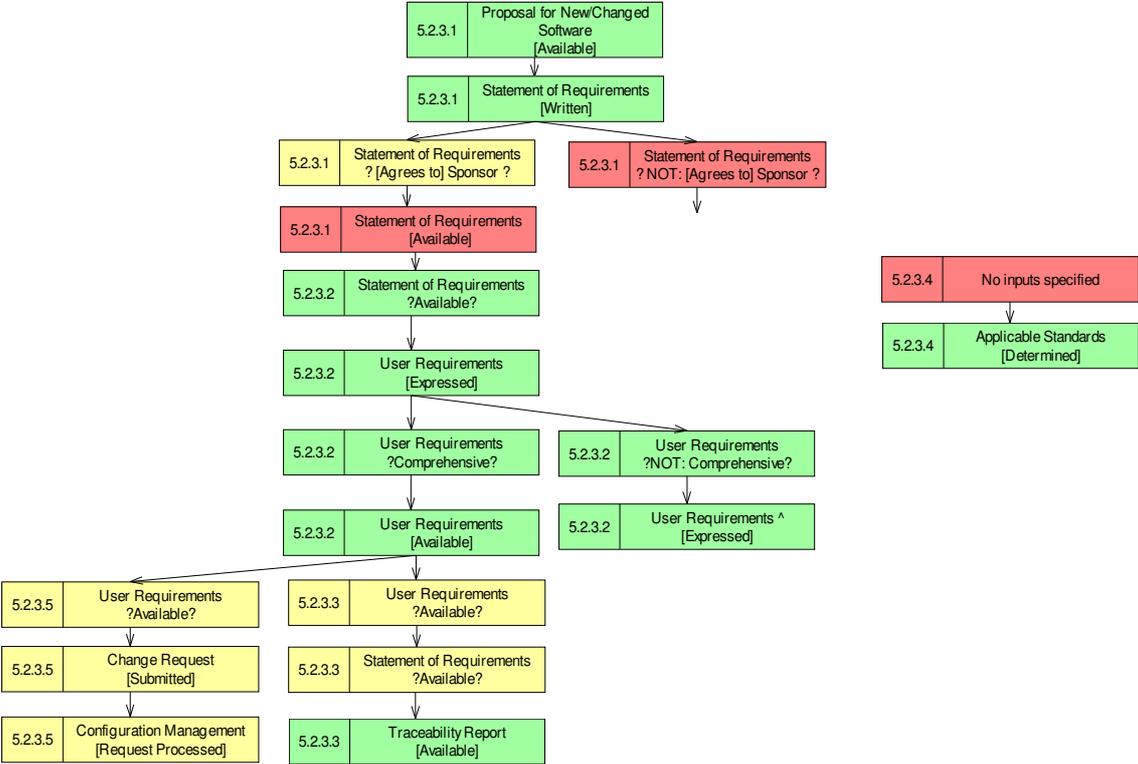


Figure 1 – Behavior Tree Analysis – OOSPICE Process Model

Figure 1 shows the integrated tree resulting from analysis of a single process. It clearly shows missing (dark shaded 5.2.3.1, 5.2,3,4) and "implied" but unstated elements (light shaded, 5.2.3.1, 5.2.3.3, 5.2.3.5), and also a failure in integration, resulting in lack of overall consistency (Ransom-Smith, McClung and Rout, 2002). The medium-shaded boxes were unchanged.

Encouraged by success on OOSPICE, the technique was subsequently applied to the review of the Capability Maturity Model Integration (V 1.2) (Chrissis, Konrad and Shrum, 2003). This work was undertaken in the context of a review of drafts of the SEI's Capability Maturity Model Integration (CMMI) V1.2, and the results formed the basis of requests for change submitted to the SEI; given resource constraints, it was not possible to apply the technique to support the complete review, but where problems were seen to exist, an analysis was conducted.

Figure 2 is indicative of how the technique helped to clarify an ambiguity in the specification of the Requirements Development Process Area in CMMI:

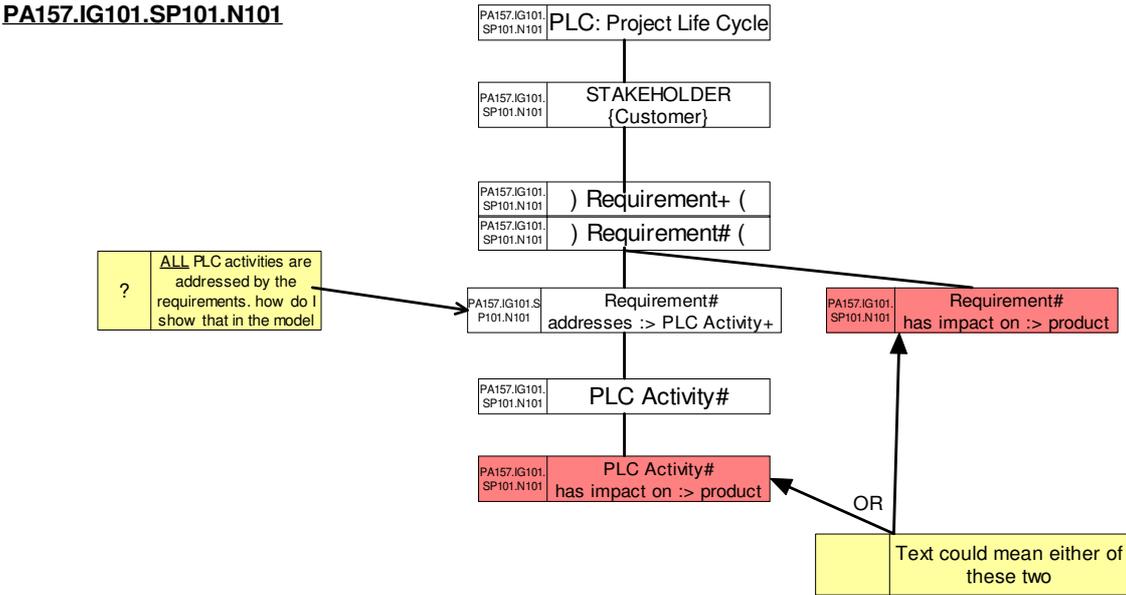


Figure 2 – Behavior Tree Analysis – Requirements Development Process Area

Given the potential identified in these two applications of the approach, it seemed logical to apply the Behavior Tree approach to the larger task of verifying a complete model. The subject of the current study is a specification for a set of organizational behaviors, specified in terms of purpose and outcomes of implementation, which would support and reinforce effective leadership in organizations, and particularly in integrated and virtual project teams.

Applying Behavior Engineering to a complete model

From the discussion above, it might reasonably be hypothesised that given the parallels between process model and software system requirements (sets of required behaviors and attributes expressed in natural language) that Behavior Engineering may prove useful in verifying a process reference model.

LEADERSHIP PROCESS REFERENCE MODEL PROJECT OVERVIEW

The leadership of integrated virtual teams is a topic in the software engineering domain that has received little attention until a project to develop such a process reference model was undertaken by the Software Quality Institute.

The topic is an important one, considering the increasing trend in a globalised environment for complex projects to be undertaken by virtual teams. The challenges of bringing any

complex project to a successful conclusion are multiplied by the coordination issues inherent in virtual environments.

The Leadership Process Reference Model (PRM) is being developed using a Design Research (DR) approach (Hevner, 2004). DR is well-adapted to the software engineering domain, and IT development generally, being used to good effect by MIT's Media Lab, Carnegie-Mellon's Software Engineering Institute, Xerox's PARC and Brunel's Organization and System Design Centre (Vaishnavi and Kuechler,2004/5).

In this project, DR is applied in the following way, consistent with Hevner's guidelines:

- A designed artefact is produced from a perceived need, based on a comprehensive literature review.
- A series of review cycles follow in which the artefact is evaluated for efficacy by a range of stakeholders and knowledgeable persons and progressively improved. In this project, five reviews are performed.
- The first and second reviews involve interviews (four interviews per round) with suitably qualified practitioner project managers. These validate the content of the PRM.
- The third review applies ISO/IEC TR 24774 *Software and systems engineering -- Life cycle management -- Guidelines for process description* (2007) to achieve consistency in form and terminology of PRMs in the Software Engineering domain.
- The fourth review applies Dromey's Behavior Engineering to the draft PRM.
- The fifth review is by an Expert Panel comprised of recognized experts in the field of PRM-building.

APPLYING BEHAVIOR ENGINEERING TO VERIFY PRM

In this project, Behavior Tree (a subset of Behavior Engineering) verification is applied as the fourth (of five) reviews. Behavior Tree analysis *could* have been applied at any stage. The circumstances of this particular project determined that the Behavior Tree analysis verification was performed towards the end, not the first or last review stage.

Being the second last review, it might be construed that the number and extent of changes that resulted from applying BE is an indication of its efficacy as a model verification tool. The previous three reviews notwithstanding, Behavior Tree analysis resulted in a significant number of changes. Indeed, most of the process outcomes needed to be reworded for clarity. Unnecessary qualifiers were removed, conjoined outcomes were split into two, each concerned with a single clear point.

Behavior Engineering is comprised of a series of related activities, performed in a broad sequence, beginning with the Behavior Tree and followed by the Composition. With the space available, this paper concerns itself with the Behavior Tree component of the broader Behavior Engineering process.

BEHAVIOR TREES – FIVE W’S (AND ONE H)

The Behavior Tree approach is based on the systematic application, with associated formal notation, of the principle of comprehensive factual description of an event known as the *Five W’s (and one H)* whose origins extend back to classical antiquity. In the 1st Century BC, Hermagoras of Temnos quoted the 'elements of circumstance' as the loci of an issue (Wooten, 1945):

Quis, quid, quando, ubi, cur, quem ad modum, quibus adminiculis
(Who, what, when, where, why, in what way, by what means)

In the modern world, this dictum has evolved into *who, what, when, where, why and how*. This principle is widely recognised and practiced in diverse domains such as journalism and police work, indeed almost anywhere that comprehensive and unambiguous description of events or attributes is needed.

Translated to the Software Engineering domain, *who, what, when, where, why and how* becomes Behavior Tree Notation. This is a branched structure showing component-states. The table below shows the application of the Behavior Tree aspect of BE in which each distinct component is described in terms of *who, what, when, where, why and how*, or the subset of these six descriptors that is applicable to this particular component.

Behavior Trees are therefore defined as *a formal, tree-like graphical device that represents behavior of individuals or networks of entities which realize or change states, make decisions, respond-to/cause events, and interact by exchanging information and/or passing control* (Dromey, 2002). Naming conventions, elements and syntax are illustrated below:

Variable	Description
N, N_t	Behavior Tree Nodes
T, T_t	Behavior Trees
C, C_t	Components
$C\#$	A Component Instance
s	A State of a Component
e	An Event
a	An Attribute of a Component
b	A Branching Condition of a Component

Table 1: Variable naming conventions (Dromey, 2007b)

Label	Name	Description
A	Component Name	Specifies a component
B	Behavior	Specifies the behaviour associated with the component
C	Operator(s)	Indicates behaviour of this node is dependent on another node in the tree
D	Label	An optional label for disambiguation (in case a node appears elsewhere with the same component and behaviour)
E	Behavior Type	Delimiters on the behaviour indicate the type of behaviour involved
F	Traceability Link	A reference to the requirements document
G	Traceability Status	Indicates how the node relates to the link
H	Tag	The box on the left-hand side of the node (by default, contains traceability information, but may be used differently, or omitted, in different contexts)
I	Behavior Tree Node	

Table 2: Elements of a Behavior Tree node (Dromey, 2007b)

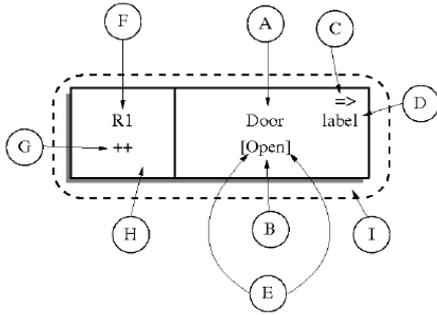


Figure 3: Behavior Tree Node concrete syntax example (Dromey, 2007b)

FUNCTIONAL REQUIREMENT TO BEHAVIOR TREE – INFORMAL TO FORMAL

Functional requirement. *When a car arrives, if the gate is open, the car proceeds, otherwise if the gate is closed, when the driver presses the button, it causes the gate to open.*

Behavior Tree. Translating the above statement into Behavior tree is illustrated below:

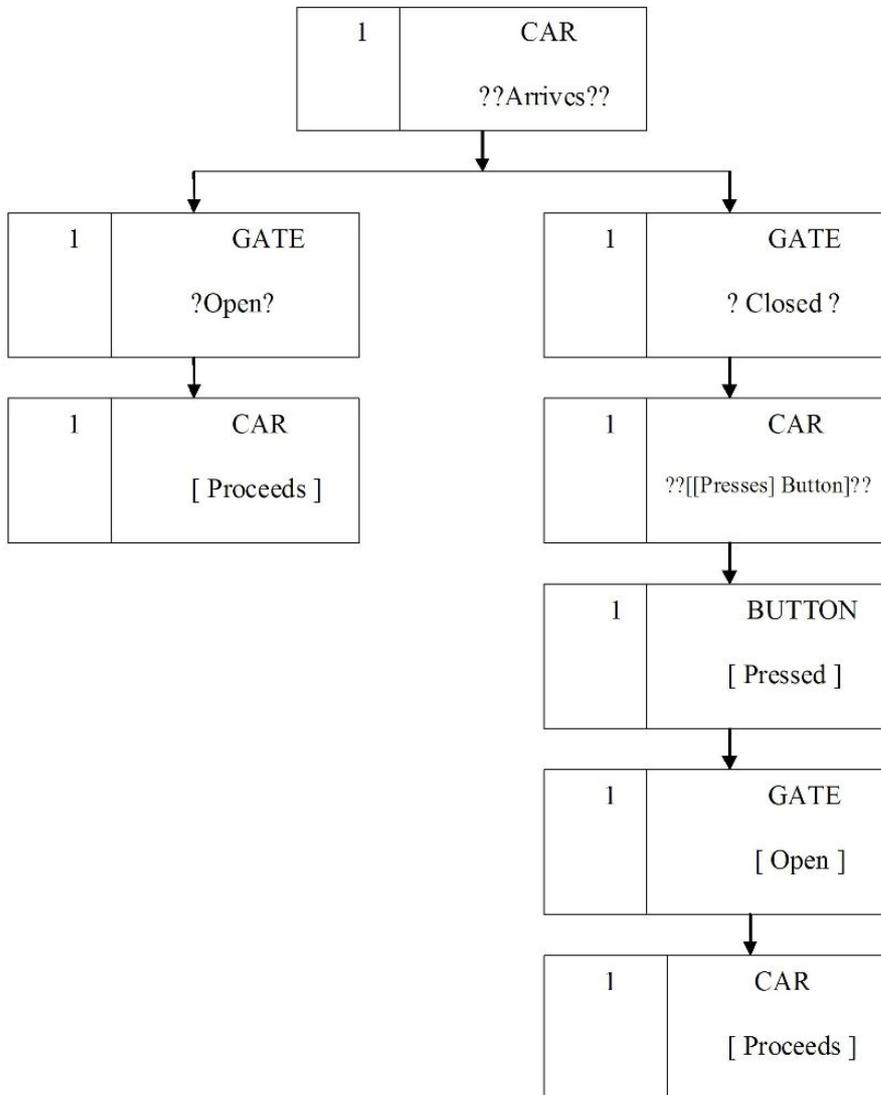


Figure 4: Functional requirement to Behavior Tree notation (Dromey, 2007a)

REMOVING AMBIGUITY

Statement: *The man saw the woman on the hill with a telescope.* This statement can be interpreted at least three different ways, a situation not uncommon with natural language. The developer must determine which interpretation is valid.

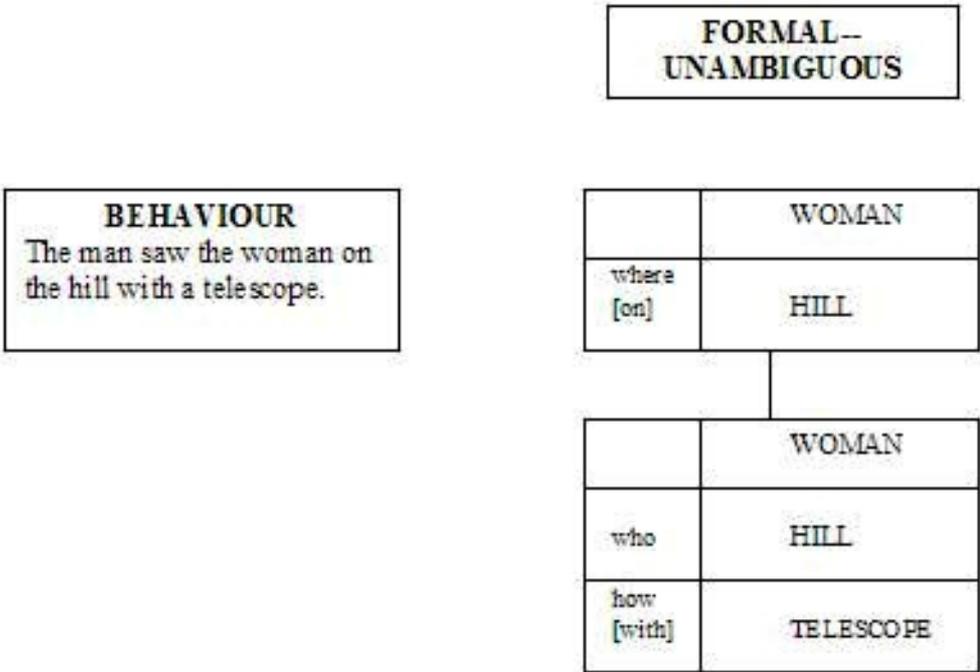


Figure 5: Resolving ambiguity using Behavior Tree notation (Dromey, 2007a)

In this example, the application of BT notation clarifies the statement by establishing the precondition (that the woman is located on the hill) from which the primary behavior can then be distinguished (that the man saw the woman by using the telescope).

APPLYING BEHAVIOR TREE NOTATION TO A PROCESS MODEL

The left-hand column of the table below shows the outcomes of the V0.3 PRM before applying the Behavior Tree notation. The Behavior Tree component column is what results from applying the *who, what, when, where, how and who (or subset)* using formal notation, from which a clear, simple restatement of the outcome can be derived, as shown in the third column. Note that the material removed from the outcome is not discarded, but relocated to the Informative Material section (not shown) where it serves a useful purpose for persons seeking a fuller understanding of the outcome. Refer to the *Rationale for Change* for discussion on specific improvements made by applying Behavior Tree notation.

In general terms, the improvements derived from the application of BT is greater clarity and economy of words (eg. in first example below 17 words in V0.3 becomes 8 in V0.4 by rephrasing ‘what is to be accomplished’ to simply ‘goal(s)’ and removing the qualifier ‘ideally seen as an accomplished fact’ to the informative section. BT highlighted where and how these economies of expression could be made by applying the process illustrated in Figure 5 to remove ambiguity; in other words the more informal language of V0.3 was rendered into formal language in V0.4 PRM.

An advantage of BT notation here is that it provides a rigorous, consistently applied editorial logic for people without qualifications and/or much experience as editors. An experienced editor *may* achieve the same results without BT notation, anyone else would arguably benefit from its application.

V0.3 PRM	Behavior Tree Component	V0.4 PRM	Rationale for change								
FIRST EXAMPLE											
Leader creates a shared vision of what is to be accomplished, ideally seen as an accomplished fact.	<table border="1"> <tr> <td>1.1.1</td> <td>LEADER (creates)</td> </tr> <tr> <td>what</td> <td>SHARED VISION/</td> </tr> <tr> <td>(of)</td> <td>GOAL(S)</td> </tr> </table>	1.1.1	LEADER (creates)	what	SHARED VISION/	(of)	GOAL(S)	Leader creates a shared vision of the goal(s).	<p><i>Goal(s) not ‘what is to be accomplished’</i></p> <p><i>Remove qualification (ideally seen as an accomplished fact) to Informative Material</i></p>		
1.1.1	LEADER (creates)										
what	SHARED VISION/										
(of)	GOAL(S)										
Leader clearly communicates the shared vision with team, ideally seen as an accomplished fact.	<table border="1"> <tr> <td>1.1.2</td> <td>LEADER (communicates)</td> </tr> <tr> <td>what</td> <td>SHARED VISION/</td> </tr> <tr> <td>(of)</td> <td>GOAL(S)</td> </tr> </table>	1.1.2	LEADER (communicates)	what	SHARED VISION/	(of)	GOAL(S)	Leader communicates the shared vision of the goal(s) with the team.	<p><i>Goal(s) included</i></p> <p><i>Remove qualification altogether - redundant</i></p>		
1.1.2	LEADER (communicates)										
what	SHARED VISION/										
(of)	GOAL(S)										
Leader facilitates strong commitment in team to achieving the shared vision, encouraging resilience in the face of goal frustrating events.	<table border="1"> <tr> <td>1.1.3</td> <td>LEADER (gets)</td> </tr> <tr> <td>what</td> <td>COMMITMENT /</td> </tr> <tr> <td>(to)</td> <td>SHARED VISION/</td> </tr> <tr> <td>(of)</td> <td>GOAL(S)</td> </tr> </table>	1.1.3	LEADER (gets)	what	COMMITMENT /	(to)	SHARED VISION/	(of)	GOAL(S)	Leader gets commitment from team to achieving the goal(s).	<p><i>Create a new outcome about resilience (it should be a stand-alone outcome rather than a qualification of the commitment to goals outcome.</i></p>
1.1.3	LEADER (gets)										
what	COMMITMENT /										
(to)	SHARED VISION/										
(of)	GOAL(S)										
New outcome in V0.4	<table border="1"> <tr> <td>1.1.4</td> <td>LEADER (encourages)</td> </tr> <tr> <td>what</td> <td>RESILIENCE /</td> </tr> <tr> <td>(in)</td> <td>TEAM</td> </tr> <tr> <td>when</td> <td>GOAL-FRUSTRATING EVENTS</td> </tr> </table>	1.1.4	LEADER (encourages)	what	RESILIENCE /	(in)	TEAM	when	GOAL-FRUSTRATING EVENTS	Leader encourages resilience in team when goal-frustrating events occur.	<p><i>New outcome focussing on the important issue of resilience in the face of goal-frustrating events</i></p>
1.1.4	LEADER (encourages)										
what	RESILIENCE /										
(in)	TEAM										
when	GOAL-FRUSTRATING EVENTS										
Leader develops a concrete and achievable set of goals that support achievement of the shared vision.	<table border="1"> <tr> <td>1.1.5</td> <td>LEADER (develops)</td> </tr> <tr> <td>what</td> <td>OBJECTIVE(S) /</td> </tr> <tr> <td>(to)</td> <td>ACHIEVE</td> </tr> <tr> <td>what</td> <td>GOAL(S)</td> </tr> </table>	1.1.5	LEADER (develops)	what	OBJECTIVE(S) /	(to)	ACHIEVE	what	GOAL(S)	Leader develops practical objective(s) to achieve the goal(s).	<p><i>Practical objectives support the achievement of the goal(s)</i></p> <p><i>Change ‘shared vision’ to ‘goals’ since the objectives derive directly from the goals.</i></p>
1.1.5	LEADER (develops)										
what	OBJECTIVE(S) /										
(to)	ACHIEVE										
what	GOAL(S)										
SECOND EXAMPLE											
Leader consistently displays <i>integrity</i> , characterised by	<table border="1"> <tr> <td>1.2.1</td> <td>LEADER (behaves)</td> </tr> </table>	1.2.1	LEADER (behaves)	Leader behaves with integrity	<p><i>Remove qualifiers to the informative section</i></p>						
1.2.1	LEADER (behaves)										

trustworthiness, and adherence to principle	what	INTEGRITY
---	------	-----------

Leader consistently displays <i>competence</i> , characterised by technical, interpersonal, conceptual and reasoning skills	1.2.2	LEADER (behaves)	Leader behaves competently	<i>Remove qualifiers to the informative section</i>
	what	COMPETENCE		

THIRD EXAMPLE

Leader provides richly-textured communications media for team members to use on-demand.	3.4.1	LEADER (provides)	Leader provides team-members with on-demand synchronous high-resolution communications media	<i>Rename 'richly-textured' to 'hi-res' (a more common term)</i> <i>Add 'synchronous' as appropriate</i> <i>Reorder the sentence to be subject-verb-object.</i>
	who	TEAM-MEMBERS		
	when	ON-DEMAND		
	what	HIGH-RES ICT /		
	/ (that is)	SYNCHRONOUS		

FOURTH EXAMPLE

Leader allocates project requirements before team members are recruited to verify the integrated team structure is appropriate to goals.	2.3.1	LEADER (verifies)	Leader verifies team structure before recruiting team-members by allocating project requirements	<i>Restructure sentence to place emphasis on correct aspects (this outcome is primarily about verifying the team structure')</i>
	what	TEAM-STRUCRURE/		
	/ when (before)	RECRUITING TEAM-MEMBERS		
	how	ALLOCATING REQUIREMENTS		

FIFTH EXAMPLE

Leader develops higher capability self-management functions early in the project lifecycle where <i>complex</i> tasks are performed <i>asynchronously</i> in virtual environments (i.e. where temporal displacement is high).	3.5.2	LEADER (develops)	Leader develops high-capability self-managing performance functions where complex tasks are performed asynchronously	<i>Reword to simplify.</i> <i>Take 'early in project' and put in Informative section.</i>
	what	PERFORMANCE-FUNCTIONS		
	/ (that are)	SELF-MANAGING /		
	/ (and)	HIGH-CAPABILITY		
	when	COMPLEX TASKS /		
	/(are perf)	ASYNCHRONOUSLY		

Table 3: Applying behavior tree notation to a process model

The Behavior Tree notation analysis was performed by the first named author after receiving around 60 minutes of training from Professor Dromey. The data shown in Table 3 is a

representative subset of the review done on the V0.3 PRM. As an indication of the defect density, this version of the model contained 24 processes with 63 outcomes collectively. Almost all outcomes were changed in some way as a result of the analysis. The kind of defects found and fixed is represented in the table above.

Defects are identified when the notation is applied, beginning with the main entity (leader in most cases), a verb that describes what the entity does (eg. develops, or verifies, or provides etc), and followed by the specific *what*, or *who* or *when* etc as makes sense for each outcome in order to build up a complete unit of sense. This process goes beyond simple editing however. When applied rigorously to the process model, a high-degree of consistency and clarity of expression is achieved. Even with competent editors, other process models (eg. OOSPICE and CMMI as discussed earlier) do not achieve this level of consistency and clarity.

The analysis of the 24 processes and 63 outcomes took around six hours to perform, including the documenting of the analysis using the kind of table seen above (with PRM before and after, notation and rationale for change).

CONCLUSION

The approach to specifying processes in terms of their purpose and outcomes was developed in the course of evolution of ISO/IEC 15504 (Rout, 2003) and is arguably a key innovation in the approach to process definition and assessment embodied in the Standard. By viewing a process (or collection of processes) in this way, it becomes clear that the outcomes represent the results of desired organizational behavior that, if institutionalised, will result in consistently achieving the prescribed purpose. The approach redirects the analysis of process performance from a focus on conformance to prescribed activities and tasks, to a focus on demonstration of preferred organizational behavior through achievement of outcomes.

Given this, it is logical to see that the application of the Behavior Tree approach to the analysis of such process models will be effective. The earlier studies reported here were of a much smaller scale than the current study, which embraces the full scope of a comprehensive model of organizational behavior. The aim in applying the approach was to provide a more formalised verification of the integrity, consistency and completeness of the model than conventional approaches – based generally on expert review – could achieve.

It may therefore be seen from Table 3 above that applying Behavior Tree notation to the draft outcomes of a process reference model produced significant improvement to the clarity of the outcomes by simplifying the language, reducing ambiguity and splitting outcomes into two where two ideas were embodied in the original.

It is suggested, based on the evidence outlined above, the Behavior Engineering is a useful tool for model-builders in the domain of model-based process improvement. It reinforces the claims that the technique is a superior tool for the verification of complex descriptions of system behavior.

REFERENCES

- Chrissis, M.B., Konrad, M., & Shrum, S., (2003). *CMMI Guidelines for Process Integration and Product Improvement*. Addison-Wesley, Boston.
- Dromey, R.G. (2001) *Genetic Software Engineering - Simplifying Design Using Requirements Integration*, IEEE Working Conference on Complex and Dynamic Systems Architecture, Brisbane, Dec 2001.
- Dromey, R.G. (2002). *From Requirements To Design – Without Miracles*, Whitepaper published by the Software Quality Institute. Available: <http://www.sqi.gu.edu.au/docs/sqi/gse/Dromey-ICSE-2003.pdf> (accessed 13 April 2009)
- Dromey, R.G. (2006). *Climbing Over the 'No Silver Bullet' Brick Wall*, IEEE Software, Vol. 23, No. 2, pp.118-120.
- Dromey, R.G. (2007a). *Principles for Engineering Large-Scale Software-Intensive Systems* Available: <http://www.behaviorengineering.org/docs/Eng-LargeScale-Systems.pdf> (accessed 14 April 2009) pg 39.
- Dromey, R.G. (2007b). *Behavior Tree Notation* Available: <http://www.behaviorengineering.org/docs/Behavior-Tree-Notation-1.0.pdf> (accessed 10 June 2009) pg 2-3.
- Glass, R.L. (2004). *Is this a revolutionary idea or not?*, Communications of the ACM, Vol 47, No 11, pp. 23-25.
- Hevner, A., March, S., Park, J. and Ram, S. (2004). *Design Science in Information Systems Research*. MIS Quarterly 28(1): pp 75-105.
- ISO/EIA 12207 (1998) *Standard for Information Technology-Software Life Cycle Processes*. This Standard was published in August 1998.
- ISO/IEC 15504 (2003) *Information Technology: Process Assessment. Joint Technical Committee IT-015, Software and Systems Engineering. Part 2 Performing an Assessment*. This Standard was published on 2 June 2005.
- ISO/IEC TR 24774 (2007). *Software and systems engineering -- Life cycle management -- Guidelines for process description*. This Standard was published in 2007.
- Milosevic, Z., Dromey, R.G. (2002) *On Expressing and Monitoring Behavior in Contracts*, EDOC-2002, Proceedings, 6th International Enterprise Distributed Object Computing Conference, Lausanne, Switzerland, Sept, pp. 3-14.
- M. Ransom-Smith, K. McClung and T. Rout, (2002) *Analysis of D5.1 – initial CBD process model using the Behavior Tree method*. Software Quality Institute report for the OOSPICE Project, December 4.
- Rout, T.P. (2003) *ISO/IEC 15504 - Evolution to an International Standard*, Softw. Process Improve. Pract; 8: 27–40.

Stallinger, F., Dorling, A., Rout, T., Henderson-Sellers, B., Lefever, B., (2002) *Software Process Improvement for Component-Based Software Engineering: An Introduction to the OOSPICE Project*, EUROMICRO 2002, Dortmund, Germany, April.

Vaishnavi, V. and Kuechler, W. (2004/5). *Design Research in Information Systems* January 20, 2004, last updated January 18, 2006. URL:
<http://www.isworld.org/Researchdesign/drisISworld.htm> Authors e-mail: vvaishna@gsu.edu
kuechler@unr.edu

Wooten, C.W. (2001) *The orator in action and theory in Greece and Rome*. Brill (Leiden, Boston).