A Tool to Visualize Behavior and Design Evolution

Lian Wen
Griffith University
170 Kessels Rd, Qld, Australia
l.wen@griffith.edu.au

Diana Kirk
Griffith University
170 Kessels Rd, Qld, Australia
d.kirk@griffith.edu.au

R. Geoff Dromey
Griffith University
170 Kessels Rd, Qld, Australia
g.dromey@griffith.edu.au

ABSTRACT
Large software systems are usually developed through a long time of evolution. The capability to retrieve and visualize the evolution history of the architecture and individual components, and also trace them back to the evolution of functional requirements will significantly help people to understand the system and reduce the cost for the maintenance. This paper demonstrates the functions of capturing, visualizing and propagating the software evolution of a behavior-oriented collaborative environment called "Integrare".

Keywords
Behavior tree, software evolution, behavior-oriented design

1. INTRODUCTION
"Integrare" is a collaborative environment based on a behavior-oriented design approach [1]. The environment integrates a number of software tools that provide functions cover a wide range of design phases . The functions of version control and software evolution management have been implemented recently.

The underlining approach of Integrare is to use behavior trees (BT), which are formal and easy to understand tree-structured graphs, to represent functional requirements. For a software system, each requirement can be translated into one behavior tree and these behavior trees can be integrated into one single behavior tree called a design behavior tree (DBT). From a DBT, a component-based design can be retrieved. During the evolution of a software system, each version of the system can be represented by a separate DBT. We can compare those different DBTs through a merging algorithm which generates an evolutionary design behavior tree (EvDBT). The EvDBT contains the information in all the compared DBTs and visualizes the evolution information in a clear way. From the EvDBT, different evolutionary design documents can then be retrieved to reveal the evolution of different design aspects such as the component architecture and the behavior of individual component. The advantage of this approach is that an EvDBT and all the other evolutionary design documents can be generated automatically by "Integrare" and also the evolution of designs can be traced back to the evolution of the functional requirements.

A small example of a DBT is shown in Figure 1. The DBT represents a car-light system with the following three requirements:

R1. When a car approaches the light, the driver checks the light.
R2. If the light is red, the driver brakes and the car will stop.
R3. If the light is green, the driver proceeds.

The three requirements are translated into three behavior trees and integrated into the DBT shown in the figure.

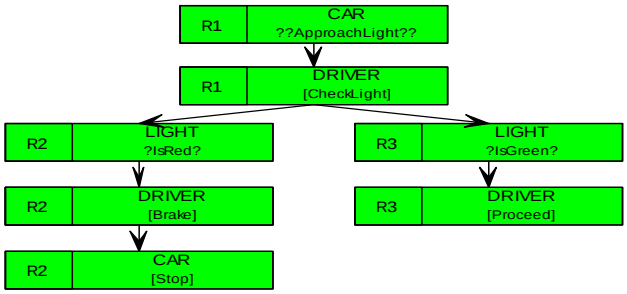

Figure 1. The design behavior tree for car-light system.

The meaning of the DBT in Figure 1 is mostly self-explained. There are three components "CAR", "DRIVER", and "LIGHT"; R1, R2 and R3 are requirement tags used to trace the behavior back to the requirements. The double question mark "??" indicates the behavior is an event, the single question mark "?" means a condition and the brackets "[…]" means a state realization. From the DBT, different views of the system can be retrieved automatically. Figure 2 presents the component integration network (CIN), which shows the component architecture, and the component DRIVER's component behavior tree (CBT), which is similar to a state diagram in a tree form.

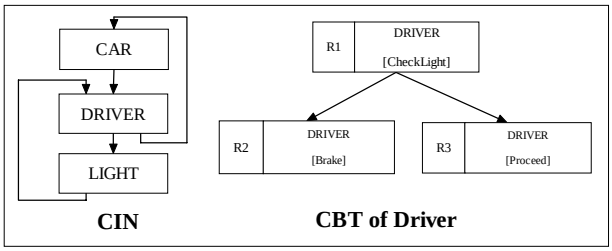

Figure 2. The CIN and the CBT of DRIVER

2. ALGORITHM
The previous section has briefly introduced the key concepts of the behavior oriented design approach. This section will introduce the tree comparison algorithm which can compare different DBTs to generate an evolutionary behavior tree. This algorithm is an extension of an earlier version [2], which can compare only two DBTs. The extended version can compare multiple DBTs at one time and has been implemented in Integrare.

If the behavior of a software system is changed, the changes will be reflected in a new DBT. Through comparing the new DBT and the old versions of DBTs, an evolutionary behavior tree is generated. From the evolutionary behavior tree, evolutionary design documents can be produced that show the change impact on the component architecture as well as on the behavior and interface of individual components.

The tree comparison algorithm uses a recursive merging process to generate the new tree. In this paper, due to the limitation of page space, we will only use a simple example to illustrate the algorithm without providing a formal description.

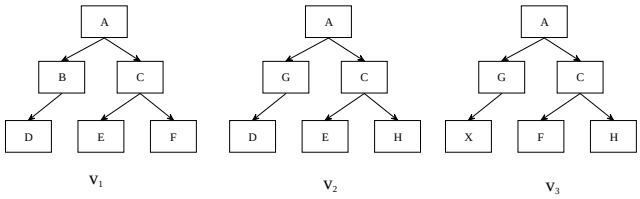

Figure 3. The behavior tree of three versions

Suppose that the three behavior trees shown in Figure 3 are three versions of a software system, and then after comparing them, an evolutionary behavior tree is generated and shown in Figure 4.

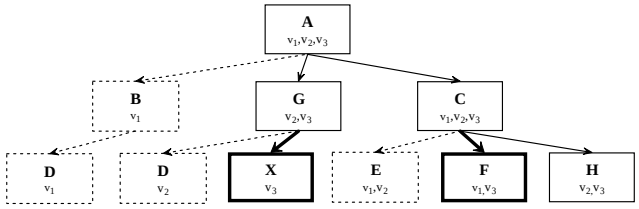

Figure 4. The evolutionary tree merged from v_1, v_2 and v_3

From Figure 4, we can see that each node is attached with a set of version tags. For example, the root node A is attached with v_1,v_2,v_3; it means that it exists in all the three versions, i.e. there is no evolution on this node, and it is displayed as the normal style. However, node B, which is attached with tag v_1, exists in the first version but has been removed in the two other versions, so it is displayed in the style of dotted lines. Similarly, node X is only attached with tag v_3, so it is a new node added in the latest version and is displayed in the bolded lines.

Once an evolutionary behavior tree is generated, other evolutionary design documents can be retrieved through the process similar to that from a normal DBT. However, the set of version tag associated with each node will also be transferred into the targeted design diagrams and some details can be seen in the following case study.

3. A CASE STUDY

For the car-light system discussed in the first section, suppose that a new version has been introduced. In the new version, the requirement R1 and R3 have not been changed, but the requirement R2 is changed to: "If the light is red, the driver *brakes hard* and the car will stop", and a new requirement R4 is introduced as: "If the light is amber, the driver *brakes light* and the car will be slow down". The DBT of the new version is shown in Figure 5. The evolutionary DBT generated from comparing DBT in Figure 1 and Figure 5 is shown in Figure 6. Figure 7 is the evolutionary component behavior tree of component DRIVER. In this diagram, behavior [Brake] has been removed and two new behaviors [BrakeHard] and [BrakeLight] are added to the component. One interesting thing is that each node in the evolutionary design documents carries both the version tags (visualized through different display styles) and the requirement tags. Therefore, an evolutionary diagram not only visualizes the evolution, but it also provides information to trace back to the change of the requirements. Due to the limitation of the length of the paper, other evolutionary design documents will not be presented.

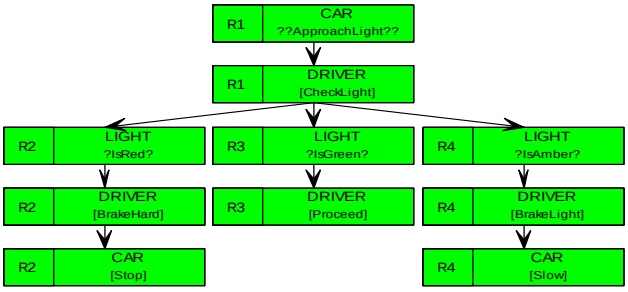

Figure 5. the DBT of the new version of the car-light system

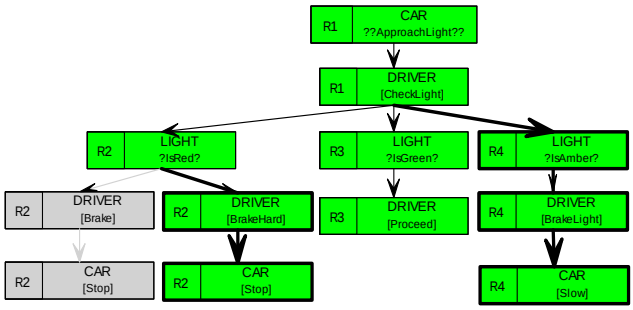

Figure 6. The EvDBT of the car-light system

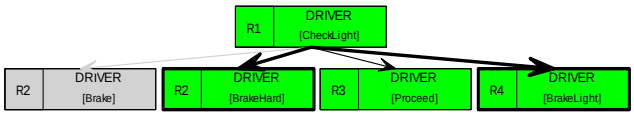

Figure 7. The evolutionary component behavior tree

This paper has briefly demonstrated the evolutionary management capability of Integrare based on a behavior-oriented design approach. This work helps to identify evolution, visualize evolution and trace evolution.

4. REFERENCES

[1] Dromey, R.G., "*From Requirements to Design: Formalising the Key Steps*", IEEE International Conference on Software Engineering and Formal Methods, 2003, pp. 2-11.

[2] Wen, L., Dromey, R.G., "*From Requirements Change to Design Change: A Formal Path*", Proceedings of the Second IEEE International Conference on Software Engineering and Formal Methods, 2004, pp. 104-113.

[3] Wen, L., Colvin, R., Lin, K, Seagrott, J, Yatapanage, N, Dromey, G, "*'Integrare', a Collaborative Environment for Behavior-Oriented Design*", Submit to CDVE2007

Screenshots of Integrare

In the following two pages, we will use a small case study: Microwave oven to demonstrate the capability of managing and visualizing software evolution of Integrare. The original requirements of the Microwave oven are:

- **R1**. There is a single control button available for the user of the oven. If the oven is idle with the door is closed and you push the button, the oven will start cooking (that is, energize the power-tube for one minute).
- **R2**. If the button is pushed while the oven is cooking it will cause the oven to cook for an extra minute.
- **R3**. Pushing the button when the door is open has no effect (because it is disabled).
- **R4**. Whenever the oven is cooking or the door is open the light in the oven will be on.
- **R5**. Opening the door stops the cooking.
- **R6**. Closing the door turns off the light. This is the normal idle state, prior to cooking when the user has placed food in the oven.
- **R7**. If the oven times-out the light and the power-tube are turned off and then a beeper emits a sound to indicate that the cooking is finished

We take the original requirement as version 1 and the DBT of version 1 is shown in Figure 8:

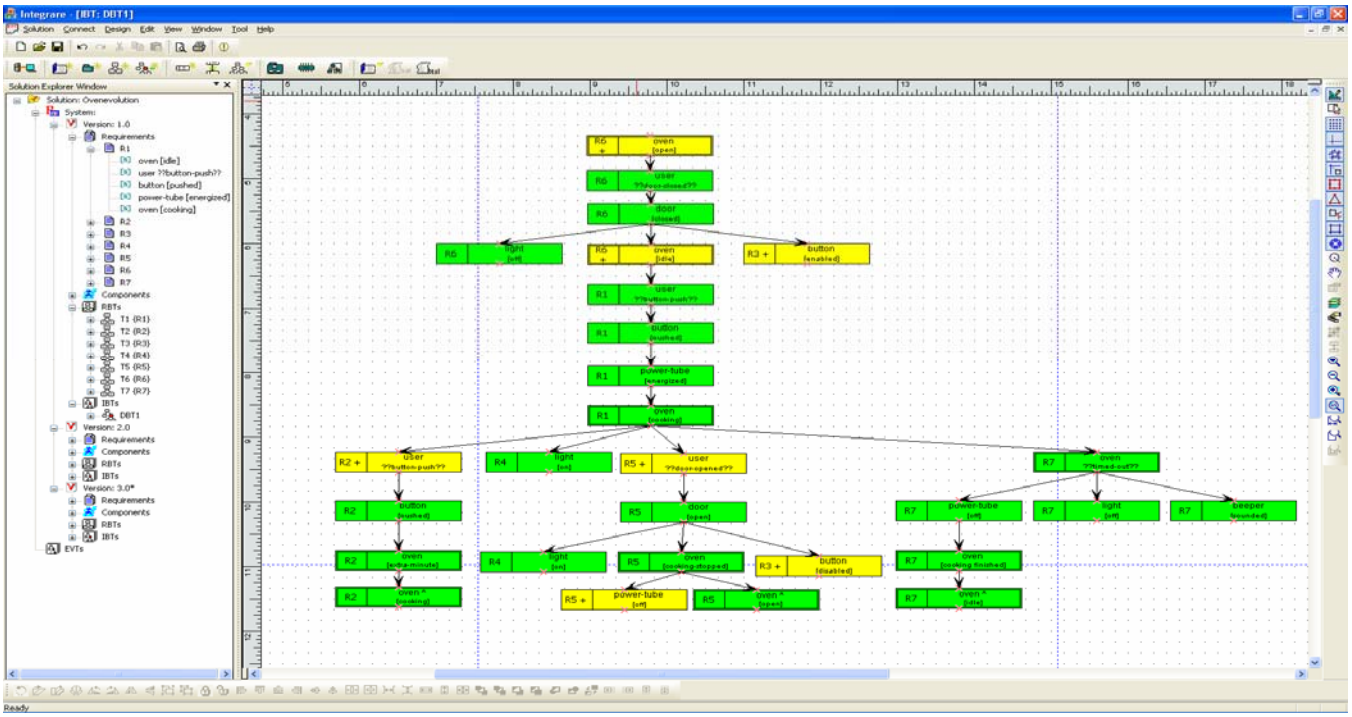

Figure 8. The DBT of version 1 of the microwave oven case study

Based on the original requirement, we add a new requirement R8 and remove one unnecessary node in the version 1 and create the DBT of the version 2 shown in left part of Figure 9.

R8: When the oven is idle, if the user opens the door, the door will be open, and the oven will be in the status open.

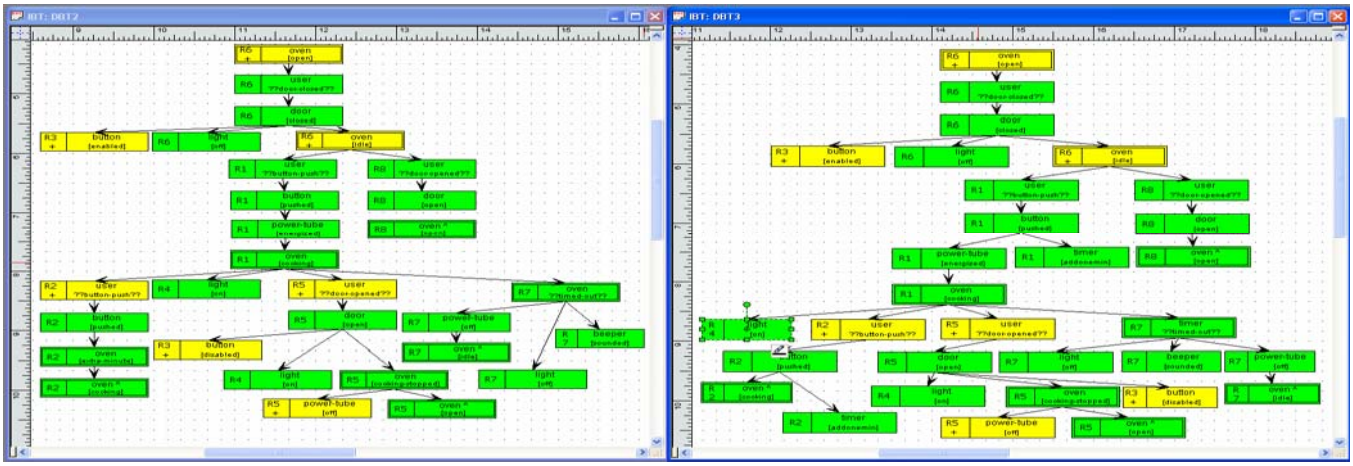

Figure 9. The DBT of version 2 and version 3 of the microwave oven case study

In version 3, we have introduced a new component TIMER to provide the timing functions, the requirements R1, R2, R7 are rewritten as:

R1: There is a single control button available for the user of the oven. If the oven is idle with the door is closed and you push the button, the timer will be set to one minute, and the oven will start cooking (that is, energize the power-tube)

R2: If the button is pushed while the oven is cooking it will cause the timer to add one extra minute

R7: If the timer times-out, the light and power-tube are turned off and then a beeper emits a sound to indicate that the cooking is finished.

The DBT of version 3 is shown in the right part of Figure 9.

Then the tool can compare the three versions of DBT and automatically generate the evolutionary DBT, which is shown in Figure 10.

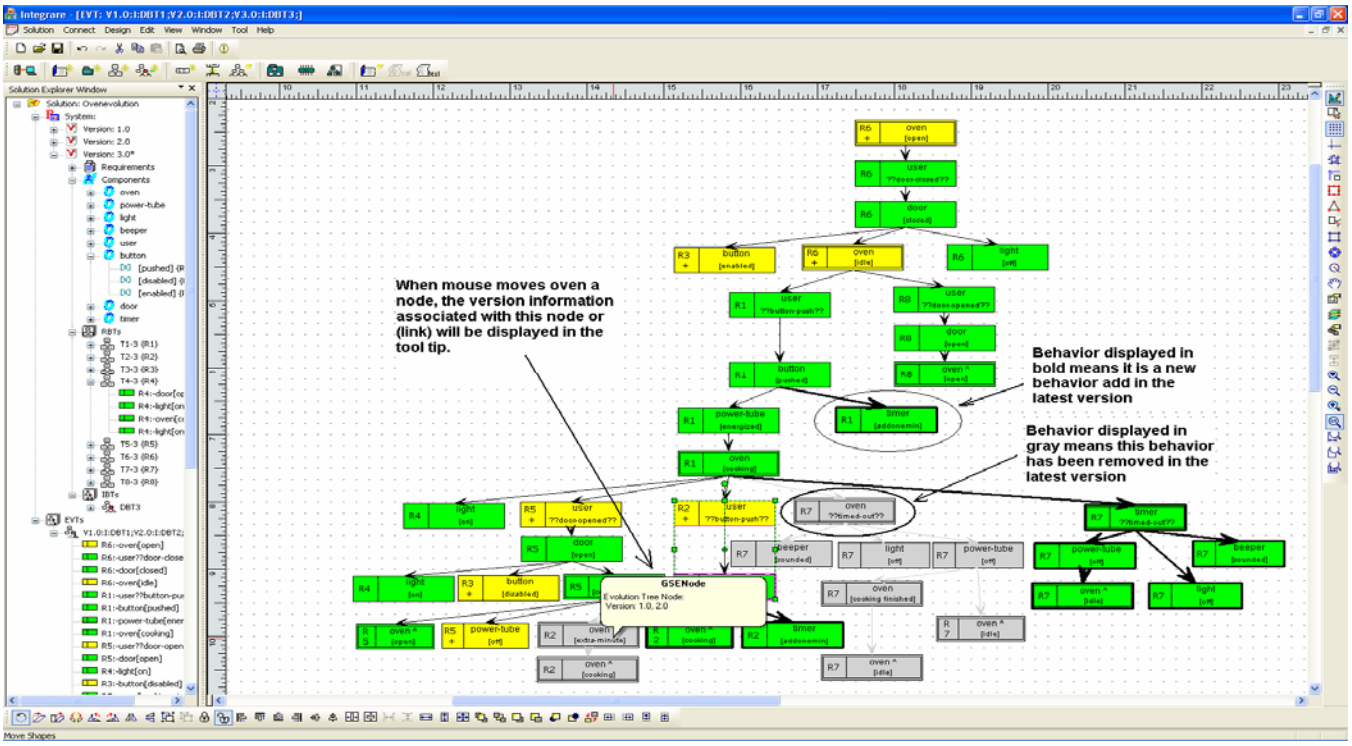

Figure 10. The evolutionary design behavior tree (EvDBT) generated from compare the three different versions of DBT

In Figure 10, behaviors (nodes) of different versions are displayed in different styles so that evolutions can be easily identified. The behavior added in the latest version is displayed in bold and the behavior has been removed in the latest version is displayed in gray. When the mouse cursor is moved on a node (or link), the version information will be displayed in the tool tip as well.

From an EvDBT, other evolutionary design documents can be automatically generated. In Figure 11, we show the evolutionary component integration network (EvCIN) and the evolutionary component behavior tree (EvCBT) of component OVEN. The version information is visualized by different display styles and the tool tip as in the EvDBT.

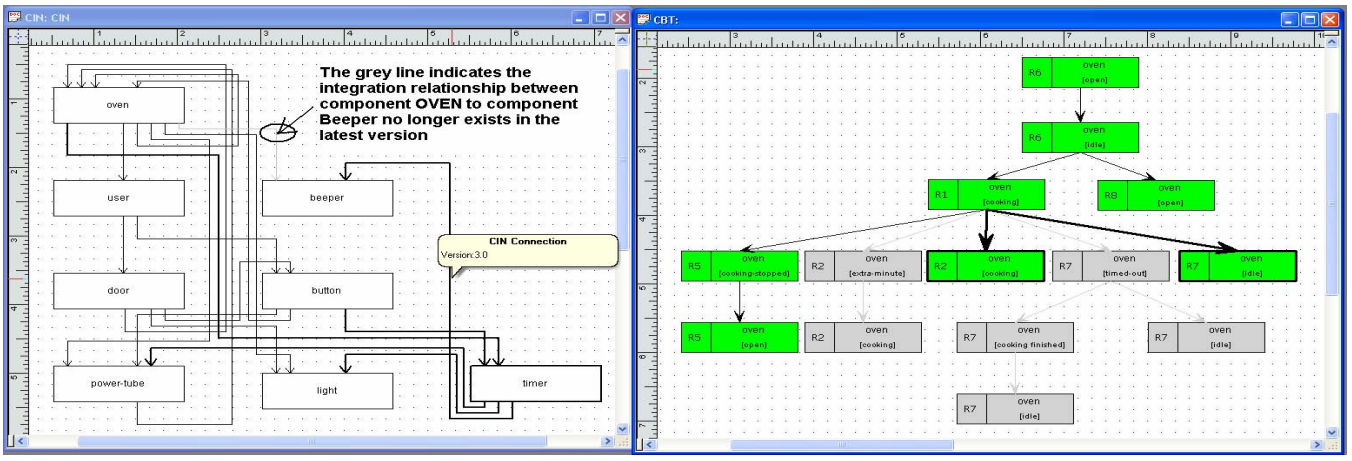

Figure 11. The evolutionary CIN and evolutionary CBT of component OVEN generated from the EvDBT in Figure 10.