ArChes - Automatic Generation of Component Fault Trees from Continuous Function Charts

Marc Zeller, Kai Höfig, Jean-Pascal Schwinn
Siemens AG, Corporate Technology
Otto-Hahn-Ring 6, 81379 Munich, Germany
Email: {marc.zeller, kai.hoefig, jean-pascal.schwinn}@siemens.com

Abstract—The growing size and complexity of software in embedded systems poses new challenges to the safety assessment of embedded control systems. In industrial practice, the control software is mostly treated as a black box during the system's safety analysis. The appropriate representation of the failure propagation of the software is a pressing need in order to increase the accuracy of safety analyses. However, it also increase the effort for creating and maintaining the safety analysis models (such as fault trees) significantly. In this work, we present a method to automatically generate Component Fault Trees from Continuous Function Charts. This method aims at generating the failure propagation model of the detailed software specification. Hence, control software can be included into safety analyses without additional manual effort required to construct the safety analysis models of the software. Moreover, safety analyses created during early system specification phases can be verified by comparing it with the automatically generated one in the detailed specification phased.

I. INTRODUCTION

The importance of safety-critical software systems in many application domains of embedded systems, such as aerospace, railway, health care, automotive and industrial automation, is continuously growing. In order to guarantee the high quality demands in these application domains, also the effort for safety assessment is increasing. The goal of the safety assessment process is to identify all failures that cause hazardous situations and to demonstrate that their probabilities are sufficiently low. In the application domains of safety-critical systems the safety assurance process is defined by the means of safety standards (e.g. the IEC 61508 standard [1]). Traditionally, the analysis of a system in terms of safety consists of bottom-up safety analysis approaches, such as *Failure Mode and Effect Analysis (FMEA)*, and top-down ones, such as *Fault Tree Analysis (FTA)*, to identify failure modes, their causes, and effects with impact on the system safety. With *Component Fault Trees (CFTs)* [2] there is a model- and component-based methodology for FTA, which supports a modular and compositional safety analysis strategy. Fault tree elements are related to their development artifacts and can be reused along with the respective development artifact.

In industry, software within safety-critical systems is currently increasing in size and importance. Hence, also the influence of software in safety analysis is increasing. However, in practice software is mostly treated as a black box within the safety analysis. The representation of the failure propagation of the software is a pressing need in order to increase the

accuracy of the safety analyses, also the effort for creating and maintaining the safety analysis models is increasing significantly. Moreover, in order to ensure the quality of the safety assessment manual and time-consuming reviews of the failure propagation model in terms of completeness and correctness are required.

In this work, we present a method to fully automatically generate Component Fault Trees from *Continuous Function Charts (CFCs)*. This methodology aims at generating the failure propagation model of the detailed software specification automatically. Hence, safety analyses in form of Fault Tree Analysis (FTA), can be performed without manual effort required to construct the safety analysis models of the software. Moreover, the failure propagation model specified during system design can be verified by comparing it with the automatically generated one.

There are a number of concepts to automatically generate failure propagation models from the system design. In [3], [4] fault tree models are generated from UML models to perform safety analysis, while [5]–[8] use a system architecture model, such as AADL, EAST-ADL, etc., as input to generate fault tree models. Moreover, some approaches deal with the automated generation of failure propagation models from a data flow language such as the one used by Matlab/Simulink [9]–[11].

However, all these approaches focus on the system architecture as input. Often the required manual modeling and preparation efforts are very high in order to be able to generate the failure propagation models. In this work, we focus on the automatic generation of failure propagation models from a detailed software specification in form of continuous function charts.

The rest of the paper is organized as follows: In Section ?? we briefly summarize relevant related work. Afterwards, we outline the concepts of CFCs in Section II and CFTs in Section III. Section IV presents our approach to automatically CFTs from CFCs. The paper is concluded in Section V.

II. CONTINUOUS FUNCTION CHARTS

Continuous Function Chart (CFC) is graphical programming language for *Programmable Logic Controller (PLC)* to design complex control and regulation tasks as an extension of the IEC 61131-3 standard [12]. Instead of using a sequence of commands in textual notation, function blocks are combined and interconnected graphically. The CFC diagrams for

programmable controller resemble electronic circuit diagrams. The function to be performed by the control system is represented in the form of the interconnected graphic elements. Since pre-defined function blocks only need to be connected to one another, complex functions can be programmed easily by developers coming from various engineering disciplines.

A CFC diagram consists of function blocks and linkages between these blocks. Each function block has different types of input and output parameters. It processes the input parameters according to a specific automation function and produces output parameters for other function blocks. The automation function of each function block is defined manually by a developer. The function blocks' outputs may be linked to the inputs of other function blocks in CFC diagrams. Thereby, each linkage indicates that an input parameter of a function block obtains its value from the specific output parameters of another function block. Therefore, CFC diagrams have a precise syntax and each function block has a well defined semantics. CFC diagrams can be created graphically using tools such as SIBAS.G or SIMATIC S7 among others. SIBAS.G is an engineering tool for the development of software for the vehicle control of trains. SIMATIC S7 is used to design complex control-engineering tasks in the industrial automation domain.

III. COMPONENT FAULT TREES

A *Component Fault Tree (CFT)* is a Boolean model associated to system development elements such as components [2]. It has the same expressive power as classic fault trees [13]. CFTs (as well as classic fault trees) are used to model the failure behavior of safety-critical systems. This failure behavior is used to document that a system is safe and can also be used to identify drawbacks of the design of a system.

In CFTs, a separate *CFT element* is related to a component. Failures that are visible at the outport of a component are models using *Output Failure Modes* which are related to a specific outport. To model how specific failures propagate from an inport of a component to the outport, *Input Failure Modes* are used. The internal failure behavior that also influences the output failure modes is modeled using the Boolean gates such as *OR* and *AND* as well as *Basic Events*.

Every CFT can be transformed to a classic fault tree by removing the input and output failure modes elements. The CFT model allows, additionally to the Boolean formula that are also modeled within the classic fault tree, to associate the specific failure modes to the corresponding ports where these failures can appear. By using CFT methodology, benefits during the development, such as an increased maintainability of the safety analysis model, can be observed [14].

IV. GENERATION OF CFTS FROM CFCS

In this section, we present our approach to automatically generate CFTs from continuous function charts (CFCs). In the following, this method is described formally and illustrated using an example as depicted in Fig. 1.
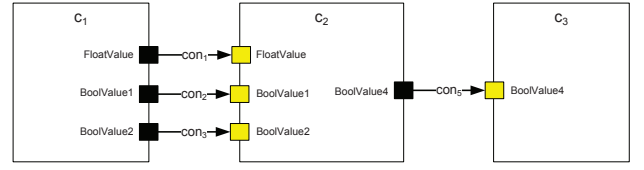

Fig. 1. Exemplary system

Let the System S consist of a set of components $C = \{c_1, ..., c_n\}$. Each component $c \in C$ includes a set of inports $IN(c) = \{in_1, ..., in_p\}$ and a set of outports $OUT(c) = \{out_1, ..., out_q\}$. The information flow between the outport of a component $c_i \in C$ and the inport of another component $c_j \in C$ (with $c_i \neq c_j$) of the system is represented as a set of connections

$$\forall c_i, c_j \in C : CON \subseteq OUT(c_i) \times IN(c_j) \qquad (1)$$

The example system as depicted in Fig. 1 is defined by:

$$
\begin{aligned}
C &= \{c_1, c_2, c_3\} \\
IN(c_1) &= \emptyset \\
IN(c_2) &= \{FloatValue, BoolValue1, \\
&\quad BoolValue2\} \\
IN(c_3) &= \{BoolValue4\} \\
OUT(c_1) &= \{FloatValue, BoolValue1, \\
&\quad BoolValue2\} \\
OUT(c_2) &= \{BoolValue4\} \\
OUT(c_3) &= \emptyset \\
CON &= \{(FloatValue, FloatValue), \\
&\quad (BoolValue1, BoolValue1), \\
&\quad (BoolValue2, BoolValue2), \\
&\quad (BoolValue4, BoolValue4)\}
\end{aligned}
$$

The behavior of each component $c_i \in C$ is defined by a CFC diagram $cfc_i \in CFC$ with $C\tilde{F}C(c_i) = cfc_i$ and $cfc \neq \emptyset$.

Each CFC is defined by a tuple

$$cfc_i = (FB(cfc_i), LINK(cfc_i), IN(cfc_i), OUT(cfc_i)) \qquad (2)$$

where $FB(cfc_i) = \{fb_1, ..., fb_m\}$ is a set of function blocks, $LINK(cfc_i)$ is a set of linkages, $IN(cfc_i) = IN(c_i)$ is a set of input parameters of the CFC and equals the set of inports of the corresponding component c_i, and $OUT(cfc_i) = OUT(c_i)$ is a set of output parameters of the CFC and equals the set of outports of the corresponding component c_i.

A function block $fb_i \in FB(cfc_i)$ of a CFC $cfc_i \in CFC$ is defined as a tuple

$$fb_i = (t(fb_i), f(fb_i), IN(fb_i), OUT(fb_i)) \qquad (3)$$

where $t(fb_i)$ is the unique type of a function block, $f(fb_i)$ is the automation function, $IN(fb_i) = \{in_{i,1}, ..., in_{i,u}\}$ is a set of input parameters of the function block, and $OUT(fb_i) = \{out_{i,1}, ..., out_{i,v}\}$ is a set of output parameters of the function block.
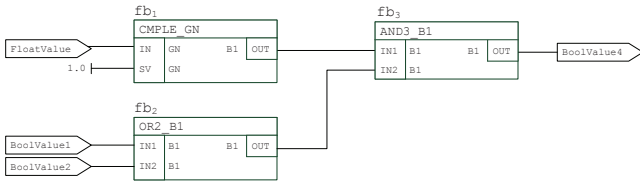
Fig. 2. CFC diagram of component c_2 (in SIBAS.G)

A linkage $link_{j,i} \in LINK(cfc_i)$ of a CFC $cfc_i \in CFC$ is a relation

$$link_{j,i} = ((x_k, y_l) \mid x_k \in OUT(fb_j) \cup IN(cfc_i),$$
$$y_l \in IN(fb_i) \cup OUT(cfc_i)) \quad (4)$$

where $out_{j,k}$ is either the kth output parameter of function block fb_j or the kth input parameter of the CFC and $in_{i,l}$ is either the lth input parameter of function block fb_i or the lth output parameter of the CFC.

An automation function $f(fb_i)$ of a function block $fb_i \in FB(cfc_i)$ is a relation between its input and output parameters (e.g. logical relations, such as *or, and, not*, etc.). It is defined as $f(fb_i) \subseteq (IN(fb_i) \times OUT(fb_i))$, where for all $in_{i,x} \in IN(fb_i)$ and $out_{i,y_1}, out_{i,y_2} \in OUT(fb_i)$, if $(in_{i,x}, out_{i,y_1}) \in f(fb_i)$ and $(in_{i,x}, out_{i,y_2}) \in f(fb_i)$ then $out_{i,y_1} = out_{i,y_2}$.

Each input parameter $in_{i,k} \in IN(fb_i)$ and output parameter $out_{i,l} \in OUT(fb_i)$ of a function block $fb_i \in FB(cfc_i)$ has a specific *connector type* $CTY(x_i) = cty$ with $x_i \in IN(fb_i) \cup OUT(fb_i)$ (e.g. Boolean, integer, etc.). If $link_{j,i} = (x_a, y_b) = (out_{fb_j,a}, in_{fb_i,b})$ then $CTY(out_{fb_j,a}) = CTY(in_{fb_i,b})$.

In our example, the CFC diagram cfc_2 of component c_2 presented in Fig. 2 is defined by:

$$
\begin{aligned}
FB(cfc_2) &= \{fb_1, fb_2, fb_3\} \\
IN(cfc_2) &= \{FloatValue, BoolValue1, \\
&\quad BoolValue2\} \\
OUT(cfc_2) &= \{BoolValue4\} \\
LINK(cfc_2) &= \{(FloatValue, IN_{1,1}), \\
&\quad (BoolValue1, IN1_{2,1}), \\
&\quad (BoolValue2, IN2_{2,2}), \\
&\quad (OUT1, 1, IN1_{3,1}), \\
&\quad (OUT_{2,1}, IN2_{3,1}), \\
&\quad (OUT_{3,1}, BoolValue4)\} \\
t(fb_1) &= CMPLE\_GN \\
IN(fb_1) &= \{IN_{1,1}, SV_{1,2}\} \\
OUT(fb_1) &= \{OUT_{1,1}\}
\end{aligned}
$$

$$
\begin{aligned}
t(fb_3) &= AND3\_Bl \\
IN(fb_3) &= \{IN1_{3,1}, IN2_{3,2}\} \\
OUT(fb_3) &= \{OUT_{3,1}\} \\
CTY(IN_{1,1}) &= CTY(SV_{1,2}) = GN \\
CTY(IN1_{2,1}) &= CTY(IN2_{2,2}) = Bl \\
CTY(IN1_{3,1}) &= CTY(IN2_{3,2}) = Bl \\
CTY(OUT_{1,1}) &= CTY(OUT_{2,1}) \\
&= CTY(OUT_{3,1}) = Bl
\end{aligned}
$$

If $c_i \in C$ has a CFT element $cft_i \in CFT$, then it is $C\tilde{F}T(c_i) = cft_i$ with $cft_i \neq \emptyset$.

Each CFT element $cft_i \in CFT(c_i)$ of a component $c_i \in C$ may have input failure modes $IFM(in_k) = \{ifm_1, ..., ifm_s\}$ which are related each to an inport $in_k \in IN(c_i)$ as well as output failure modes $OFM(out_l) = \{ofm_1, ..., ofm_t\}$ which are related each to an outport $out_l \in OUT(c_i)$.

In order to specify the semantics of the failure modes within component fault tree an unambiguous failure type fty is assigned to each input and output failure mode. The different failure types as well as the relation between them are specified in a so-called *failure type system* T [15]:

$$FTY(fm) = fty$$
$$\text{with } fm \in \bigcup_{i=1}^{p} IFM(in_i) \cup \bigcup_{j=1}^{q} OFM(out_j)$$
$$\text{and } fty \in T$$
$$(5)$$

Moreover, each CFT element $CFT(c_i) \neq \emptyset$ of a component $c_i \in C$ may have a set of gates $G = \{g_1, ..., g_r\}$. Each gate $g_i \in G$ has exactly one output $g_i.out$, one or more inputs $g_i.IN = \{g_i.in_1, ..., g_i.in_s\}$, and a Boolean formula b (e.g. $g.out = g.in_1 \vee g.in_2$).

Input and output failure modes as well as gates are connected by a set of directed edges

$$E \subseteq \{(out_x, in_y) \mid out_x \in \bigcup_{i=1...p} IFM(in_p) \bigcup_{j=1...r} g_j.out$$
$$in_y \in \bigcup_{k=1...r} g_k.IN \bigcup_{l=1...q} OFM(out_l)\}$$
$$(6)$$

The generation of a CFT from a CFC diagram is performed in three steps, which are defined as follows:

A. Generation of CFT elements

At first, a CFT element is created for each CFC diagram within a specific project:

$$\forall cfc_i \in CFC \text{ with } cfc_i = C\tilde{F}C(c_i): C\tilde{F}T(c_i) = cft_i$$
$$(7)$$

Thus, $\forall c_i \in C : \exists cft_i \in CFT$.

Moreover, based on the inputs and outputs defined in each CFC diagram, inports and outports are generated and

interconnected based on the unique names of the inputs and outputs of the CFC diagrams:

$$\forall\, cfc_i \in CFC:\ IN(c_i) \rightarrow IN(cfc_i) \tag{8}$$

$$\forall\, cfc_i \in CFC:\ OUT(c_i) \rightarrow OUT(cfc_i) \tag{9}$$

and

$$\begin{aligned}
&\forall\, cfc_i, cfc_j \in CFC \text{ with } cfc_i \neq cfc_j: \\
&\quad \forall\, out_{i,k} \in OUT(cfc_i), in_{j,l} \in IN(cfc_j): \\
&\quad\quad \rightarrow \{(out_{i,k}, in_{j,l}) \mid name(out_{i,k}) = name(in_{j,l})\}
\end{aligned} \tag{10}$$

For the exemplary system as depicted in Fig. 1 & 2, the following CFT elements are generated (see Fig. 3:

$$\begin{aligned}
C\tilde{F}T(c_1) &= cft_1 \\
C\tilde{F}T(c_2) &= cft_2 \\
C\tilde{F}T(c_3) &= cft_3 \\
IN(cfc_2) &= \{FloatValue, BoolValue1, \\
&\qquad BoolValue2\} \\
OUT(cfc_2) &= \{BoolValue4\} \\
CON &= \{(FloatValue, FloatValue), \\
&\qquad (BoolValue1, BoolValue1), \\
&\qquad (BoolValue2, BoolValue2), \\
&\qquad (BoolValue4, BoolValue4)\}
\end{aligned}$$

B. Generation of Input & Output Failure Modes

In the next step, the input and output failure modes are generated for each of the previously created CFT elements.

The generation of the failure modes is based on a generic mapping between the connector types in the CFC diagram and the failure types of the failure modes in the CFT element. Whereas, each connector type corresponds to a set of failure types from the generic failure type system T [15]:

$$MAP:\ CTY(x_i) \mapsto \{fty_1, ..., fty_n\} \in T \tag{11}$$

with $x_i \in IN(cft_i) \cup OUT(cft_i)$ and $cft_i \in CFT$ and $fty_j \in T$.

The generic mapping from connector types in CFCs to failure types in CFTs is presented in Table I. For each

Connector Type	Failure Type
Boolean	false positive, false negative, omission, commission, too early, too late
Integer, Float, Time	too high, too low, omission, commission, too early, too late

TABLE I
MAPPING OF CONNECTOR TYPES IN CFCs TO FAILURE MODES IN CFT

CFT element $cft_i \in CFT$, a set of input failure modes

as well as a set of output failure modes is generated based on the connector types of the inputs $IN(cfc_i)$ and outputs $OUT(cfc_i)$ of the corresponding CFC diagram $cfc_i \in CFc$, where $cfc_i = C\tilde{F}C(c_i)$ and $cft_i = C\tilde{F}T(c_i)$ with $c_i \in C$:

$$\begin{aligned}
&\forall\, cfc_i \in CFC:\ \forall\, in_j \in IN(cfc_i): \\
&\quad \rightarrow \{ifm_k \mid MAP(CTY(in_j)) = FTY(ifm_k)\}
\end{aligned} \tag{12}$$

and

$$\begin{aligned}
&\forall\, cfc_i \in CFC:\ \forall\, out_j \in OUT(cfc_i): \\
&\quad \rightarrow \{ofm_k \mid MAP(CTY(out_j)) = FTY(out_k)\}
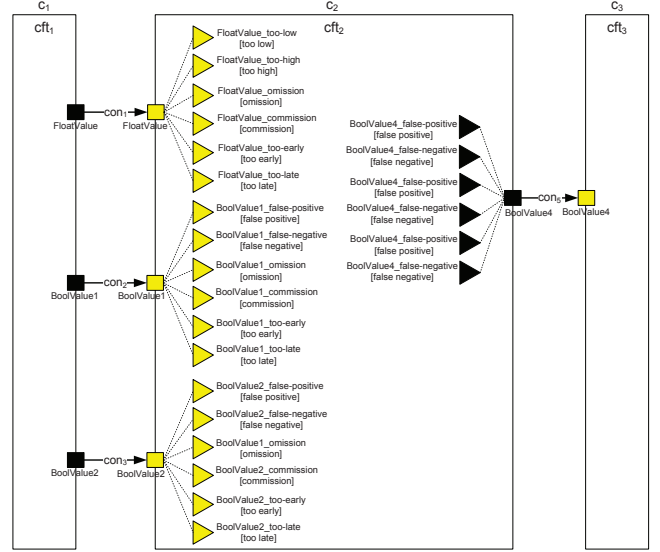\end{aligned} \tag{13}$$

Fig. 3. Exemplary system: Generated input and output failure modes

For the component c_2 of the exemplary system presented in Fig. 1 and Fig. 2, a set of input and output failure modes are generated as depicted in Fig. 3 where each input or output failure mode has a specific failure type FTY (displayed in square brackets).

C. Generation of the failure propagation

In a last step, the failure propagation from the input failure modes of each CFT element $cft_i \in CFT$ to its output failure modes is generated based on the definition of the corresponding CFC diagram $cfc_i = C\tilde{F}C(c_i) \in CFC$. Therefore, input and output failure modes of the CFT element cft_i are connected using Boolean gates [2].

At first, a set of Boolean gates G is generated based on specific predefined rules for each function block $fb_j \in FB(cfc_i)$. Therefore, a set of *rules* $R(t(fb_i)) = \{r_1, ..., r_s\}$ must be defined for each type of function block $t(fb_j) \in FB(cfc_i)$ of the CFC. It describes for all possible failure types how the output parameter of a function block are related to the possible failure types its input parameters (see Eq. 14). The possible failure types of the input and output parameters of a function block in the CFC are defined by their connector type according to the mapping MAP (see Table I). For instance, if the connector type of the output parameter is a *Boolean*, two

$$\forall\, cfc_i \in CFC : \; \forall\, fb_j \in FB(cfc_i) : \; \forall\, out_{k,l} \in OUT(fb_j) :$$
$$R(t(fb_j)) = \{r_i \mid r_i : MAP(CTY(out_{k,l})) \mapsto MAP(CTY(in_{k,1})) \circ \ldots \circ MAP(CTX(n_{k,u})\}$$
$$\text{with } \circ = \{\neg, \wedge, \vee, \oplus\} \tag{14}$$
$$\text{and } i = 1, \ldots, |MAP(CTY(out_{k,l}))|$$

rules must be defined: one for the failure type *false negative* and one rule for the failure type *false positive*.

In case that no rule is predefined for a type of function block $fb_j \in FB(cfc_i)$ used in the CFC diagram $cfc_i \in CFC$, only the worst case scenario for the failure propagation can be assumed. This worst case scenario is defined in Eq. 15.

For the function blocks of component c_2 in our example as depicted in Fig. 2 the following rules are defined:

$R(CMPLE\_GN) =$
$\quad \{OUT.false\text{-}positive = IN.too\text{-}low,$
$\quad\; OUT.false\text{-}negative = IN.too\text{-}high,$
$\quad\; OUT.omission = IN.omission,$
$\quad\; OUT.commission = IN.commission,$
$\quad\; OUT.too\text{-}early = IN.too\text{-}early,$
$\quad\; OUT.too\text{-}late = IN.too\text{-}late\}$

$R(OR2\_Bl) =$
$\quad \{OUT.false\text{-}positive = IN1.false\text{-}positive \vee$
$\qquad\qquad\qquad\quad IN2.false\text{-}positive,$
$\quad\; OUT.false\text{-}negative = IN1.false\text{-}negative \wedge$
$\qquad\qquad\qquad\quad IN2.false\text{-}negative,$
$\quad\; OUT.omission = IN1.omission \wedge$
$\qquad\qquad\qquad\quad IN2.omission,$
$\quad\; OUT.commission = IN1.commission \vee$
$\qquad\qquad\qquad\quad IN2.commission,$
$\quad\; OUT.too\text{-}early = IN1.too\text{-}early \vee$
$\qquad\qquad\qquad\quad IN2.too\text{-}early,$
$\quad\; OUT.too\text{-}late = IN1.too\text{-}late \wedge$
$\qquad\qquad\qquad\quad IN2.false\text{-}late\}$

$R(AND3\_Bl) =$
$\quad \{OUT.false\text{-}positive = IN1.false\text{-}positive \wedge$
$\qquad\qquad\qquad\quad IN2.false\text{-}positive,$
$\quad\; OUT.false\text{-}negative = IN1.false\text{-}positive \vee$
$\qquad\qquad\qquad\quad IN2.false\text{-}positive,$
$\quad\; OUT.omission = IN1.omission\wedge$
$\qquad\qquad\qquad\quad IN2.omission,$
$\quad\; OUT.commission = IN1.commission \vee$
$\qquad\qquad\qquad\quad IN2.commission,$
$\quad\; OUT.too\text{-}early = IN1.too\text{-}early \vee$
$\qquad\qquad\qquad\quad IN2.too\text{-}early,$
$\quad\; OUT.too\text{-}late = IN1.too\text{-}late \wedge$
$\qquad\qquad\qquad\quad IN2.false\text{-}late\}$

Based on these rules, the following Boolean gates are gener-

ated for the CFT element cft_2 of component c_2:

$$AND1\text{-}1 = AND1\text{-}2 = AND1\text{-}3$$
$$= AND2\text{-}1 = AND2\text{-}2 = AND2\text{-}3$$
$$= (AND.out, \{AND.in_1, AND.in_2\},$$
$$AND.out = AND.in_1 \wedge AND.in_2)$$
$$OR1\text{-}1 = OR1\text{-}2 = OR1\text{-}3$$
$$= OR2\text{-}1 = OR2\text{-}2 = OR2\text{-}3$$
$$= (OR.out, \{OR.in_1, OR.in_2\},$$
$$OR.out = OR.in_1 \vee OR.in_2)$$

Afterwards, the input and output failure modes as well as the Boolean gates of each CFT element $cft_i \in CFT$ are interconnected according to the corresponding CFC's linkage $LINK(cfc_i)$. For creating the interconnections, we start from the output failure modes and connect them with the available input failure modes through the Boolean gates. Therefore, a set of direct edges is created as follows:

$$\forall\, out_j \in OUT(cfc_i) : \; \forall\, \forall ofm_k \in OFM(out_j) :$$
$$\rightarrow \{(x, ofm_k) \mid x \in IFM(in_l) \vee x = g_r.out,$$
$$\exists\, link_z \in LINK(cfc_i) = (y, OUT(cfc_i)), \tag{16}$$
$$y = in_l \vee y \cup OUT(fb_d), fb_s \Rightarrow g_r\}$$

and

$$\forall\, g_j.IN \in G : \; \forall\, g_j.in_k \in g_j.IN :$$
$$\rightarrow \{(x, g_j.in_k) \mid x \in IFM(in_l) \vee x = g_r.out, g_r \neq g_j$$
$$\exists\, link_z \in LINK(cfc_i) = (y, OUT(cfc_i)),$$
$$y = in_l \vee y \cup OUT(fb_s), fb_s \Rightarrow g_r\}$$
$$\tag{17}$$

The result of the generation process for the exemplary system is presented in Fig. 4.

V. CONCLUSIONS

Our approach to generate Component Fault Trees (CFTs) from Continuous Function Charts (CFCs) automatically creates a failure propagation model based on thte detailed software specification of the system. No additional manual effort is needed to perform a safety analysis. The resulting CFT can be used for Fault Tree Analyses (FTA) of the overall system including the software as a white box. Thus, the accuracy of the safety analysis is increased without additional effort needed for the construction and maintenance of the safety analysis model for the controller software of the system. Compared to traditionally used methods for the analysis of software such as root cause analysis or fault injection tests, in which the quality of the results is depending on the accuracy of the

$$\forall\, out_{k,l} \in OUT(fb_j):$$

$$R(\text{worst case}) = \left\{ r_i \mid r_i : MAP(CTY(out_{k,l})) \bigvee \bigvee_{s=1}^{u} MAP(CTY(in_{k,t})) \right\} \qquad (15)$$

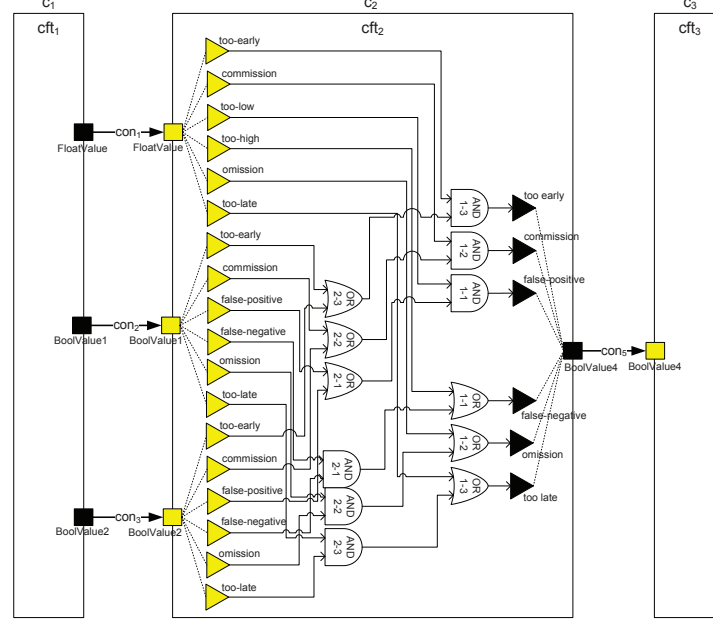$$\text{with } i = 1, ..., |MAP(CTY(out_{k,l}))|$$

Fig. 4. Exemplary system: Generated CFT element for component c_2

input data which are generated manually by a team of experts (e.g. by brainstorming), our approach provides a complete set of input data while no manual effort is required. Moreover, the automatically generated CFT may be compared to a CFT (or Fault Tree) created manually by a safety engineering during the system specification (e.g. as described by [16]). Hence, it is possible to verify if the failure propagation model specified during system design is build correctly in terms of consistency and completeness.

REFERENCES

[1] International Electrotechnical Commission (IEC), "IEC 61508: Functional safety of electrical/electronic/programmable electronic safety related systems," 1998.
[2] B. Kaiser, P. Liggesmeyer, and O. Mäckel, "A new component concept for fault trees," in *Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software*, 2003, pp. 37–46.
[3] A. Bondavalli, I. Majzik, and I. Mura, "Automated Dependability Analysis of UML Designs," *IEEE International Symposium on Object-oriented Real-time distributed Computing*, vol. 2, 1999.
[4] M. A. de Miguel, J. F. Briones, J. P. Silva, and A. Alonso, "Integration of safety analysis in model-driven software development," *Software, IET*, vol. 2, no. 3, pp. 260–280, 2008.
[5] M. Bretschneider, H. J. Holberg, E. Bode, and I. Bruckner, "Model-based safety analysis of a flap control system," *Proc. 14th Annual INCOSE Symposium*, 2004.
[6] Y. Papadopoulos, J. A. McDermid, R. Sasse, and G. Heiner, "Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure," *Int. Journal of Reliability Engineering and System Safety*, vol. 71, no. 3, pp. 229–247, 2001.
[7] A. Rae and P. Lindsay, "A behaviour-based method for fault tree generation," *Proc. of the 22nd Int. System Safety Conference*, pp. 289 – 298, 2004.
[8] G. Szabo and G. Ternai, "Automatic Fault Tree Generation as a Support for Safety Studies of Railway Interlocking Systems," *IFAC Symposium on Control in Transportation Systems*, 2009.
[9] Y. Papadopoulos and M. Maruhn, "Model-Based Synthesis of Fault Trees from Matlab-Simulink Models," *International Conference on Dependable Systems and Networks*, 2001.
[10] F. Tajarrod and G. Latif-Shabgahi, "A Novel Methodology for Synthesis of Fault Trees from MATLAB-Simulink Model," *World Academy of Science, Engineering & Technology*, vol. 2, no. 5, pp. 1181–1187, 2008.
[11] S. Buono, V. Ramich, B. Kaiser, and J. Zander, "An industry case study on semi-automated generation of component fault trees from simulink-models," in *Gemeinsamer Tagungsband der Workshops der Tagung Software Engineering 2015*, 2015, pp. 41–50.
[12] International Electrotechnical Commission (IEC), "IEC 61131-3: Programmable controllers - Part 3: Programming languages," 2013.
[13] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl, *Fault Tree Handbook*. US Nuclear Regulatory Commission, 1981.
[14] J. Jung, A. Jedlitschka, K. Höfig, D. Domis, and M. Hiller, "A controlled experiment on component fault trees," in *Computer Safety, Reliability, and Security*, 2013, pp. 285–292.
[15] J. McDermid and D. Pumfrey, "A development of hazard analysis to aid software design," in *Proceedings of the COMPASS '94*, 1994, pp. 17–25.
[16] D. Domis, K. Höfig, and M. Trapp, "A consistency check algorithm for component-based refinements of fault trees," in *2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE)*, 2010, pp. 171–180.