Abstract

We adapt in this paper an analysis-revision cycle to the SCTL-MUS methodology which we will use to support the modification and evolution of requirements specifications in a multi-perspective environment. We take advantage of using our approach in the analysis and revision phases, being able to translate the diagnostic information into system requirements refinements close to the system domain of each viewpoint.

To illustrate these advantages, we employ two viewpoints of a thermostat system. Both viewpoints are merged in order to be able to reason over the properties of the composed system. The refinements over the merged model are transferred into each one of the viewpoints and then into the requirements in which are expressed both ones. So, we obtain requirements refinements close to the system domain of each viewpoint, facilitating to the stakeholders the decision of what system requirements refinements must be included in the system requirements specification.

Keywords: Requirements Refinement, Viewpoints, Unspecification, Requirements Specification Evolution, Multi-Perspective Environment.

1 Introduction

Software engineers make use of a large number of different models that are constructed and updated by different developers at different times during development. These models incorporate information from multiple sources, reflecting different points of view. Such models have significant overlaps because they typically describe orthogonal aspects of a system, rather than hierarchical decompositions.

An effective way to specify individual concerns (they can come, for example, from software engineers playing different roles in software development or from stakeholders with different views on the system) is via viewpoint–based approaches [13, 8]. These approaches allow to gather and maintain (possibly incomplete and inconsistent) information gathered from multiple sources. They explicitly separate the descriptions provided by different stakeholders, and concentrate on identifying and resolving conflicts between them. Clearly, many concerns overlap and interact. To make the concept of separation of concerns truly general and usable, we need to reason about properties of the composed system [3].

We are interested in reasoning about information merged from multiple sources, in environments where different aspects of a system are described in separate models or viewpoints.

On the other hand, in [6], authors argue that the development of requirements specifications can be supported by a cycle composed of two phases: analysis and revision. The analysis phase is responsible for checking whether a number of desirable properties of a system is satisfied by its partial specification. It also provides appropriate diagnostic information when a certain property is violated by the specification. The revision phase should change the given specification into a new (partial) specification – by making use of the diagnostic information obtained from the analysis phase – in such a way that the new specification no longer violates the system’s property in question.

The particular instance of this cycle proposed in [6] uses techniques of abductive reasoning during the analysis phase to (i) discover whether a given system’s description \( \mathcal{D} \) satisfies a system property \( \mathcal{P} \) \( (\mathcal{D} \models \mathcal{P}) \) and (ii) if not, generate appropriate diagnostic information \( \Delta \); and inductive learning during the revision phase to change the description \( \mathcal{D} \) into a new description \( \mathcal{D}' \) whenever it violates the property \( \mathcal{P} \), using a machine learning algorithm. The two techniques are combined together by using the diagnostic information \( \Delta \) generated by abduction, to derive a number of training examples \( \Delta' \) for inductive learning.

In this paper, we adopt the analysis–revision cycle pro-
posed above in order to apply it in a multi-perspective environment. This cycle is adapted to the SCTL-MUS methodology [14, 9] in the following way:

- The analysis phase is carried out over a merged model (obtained by combining the different viewpoints) where we verify a desirable property in order to detect whether it is satisfied and to generate diagnostic information if the property is not satisfied.

- The revision phase is carried out in two steps. Firstly, we obtain system refinements from the provided diagnostic information. Secondly, those refinements are translated into requirements refinements of the requirements specification of each viewpoint, providing in this way, refinements close to the application domain of each viewpoint. Employing this method, we can refine the requirements of each viewpoint separately to obtain the merged model satisfying a given set of properties.

To illustrate the advantages of our approach, we use two viewpoints of a thermostat system. Both viewpoints are merged in order to be able to reason over the properties of the composed system. The refinements over the merged model are transferred into each one of the viewpoints and then into the requirements in which they are expressed both ones. So, we obtain requirements refinements close to the domain of each viewpoint while preserving the main requirements goals and properties.

The paper is organized as follows. Section 2 provides a description of our approach and its advantages. Section 3 describes the example which we will use throughout this paper, shows the detailed application of our approach to achieve that the merged model satisfies a given property and how it affects the requirements specifications of the two viewpoints. Section 4 concludes and discusses directions for future work.

2 Approach

Our approach provides an adequate framework to facilitate the evolution of requirements specifications when the kind of properties which involve not-specified elements at the current (partial) specification are checked. We can distinguish four completely differentiated parts in our approach:

2.1 Viewpoints specification: SCTL-MUS

Our approach is based on the formal methodology called SCTL-MUS. It proposes an incremental development model [111] which formalizes system evolutions by defining unspecified elements which can evolve into specified ones. To achieve it, an extension of classical Labeled Transition Systems, referred to as MUS (Model of Unspecified States), is used to model the system. The elements (states, actions or labels and ares) of a MUS graph can be specified as: true (1); false (0); and unspecified (½) which can evolve into 1 or 0.

We have defined a multi-valued logic [15, 2], referred to as SCTL (Simple Causal Temporal Logic), with six degrees of satisfaction which allow to reason about system evolutions. Besides the classical truth values (true (1) and false (0)), we have defined three additional ones to model properties which are not totally specified at the current system specification (it is developed in an incremental way): ½ or unspecified, the satisfaction of the specified property can become true and false; ¾, it cannot become true; 2, it cannot become false. SCTL has a causal semantics [12]:

“If ... (Premise) ... is possible, then Simultaneously (⇒) / Previously (□) / Next (◊) ... (Consequence) ... must be possible”

That is the reason why we have defined a new truth value called contradictory or not-applicable (½) which is assigned to properties whose premise is not satisfied. An atomic SCTL requirement is defined as a SCTL requirement where its premise is the constant true and its consequence is an action which can be preceded by the negation operator (¬). Therefore, atomic SCTL requirements have the following form: true ⇒ ¬a, where a ∈ {□, ○}. To simplify the notation, they will be denoted by: ¬a.

2.2 Merging viewpoints

This stage consists in obtaining a merged model from a set of viewpoints. Given a set of MUS viewpoints, they can be combined using the framework proposed in [7]. To define how the truth values (0, ½, 1) used in the source MUS viewpoints are combined as logical values in the merged model, we propose to use the function called union1. It is our value map function. The combined actions which obtain the “?” value are called inconsistent actions (A incons) and they may cause a disagreement among the combined viewpoints. The merged MUS contains the actions of each viewpoint i for all a ∈ {A incons}. It allows to reason about levels of agreement among the different viewpoints [10]. Our approach allows designers to reason over a merged model using specifications in an intermediate development phase, since we propose an incremental development model. Moreover, this approach provides information to reason about how levels of agreement can evolve according to the different possible evolutions (at next development phases) of each viewpoint.

1 a ∪ b = a (if a = b or b = ½); b (if a = ½); “?” (otherwise).
2.3 Analysis phase

The problem of finding whether \( \mathcal{D} \models \mathcal{P} \) is translated into the equivalent problem of showing that it is not possible to find a set \( (\Delta) \) of state transitions consistent with \( \mathcal{D} \) and, that together with \( \mathcal{D} \), proves the negation of \( \mathcal{P} \) (\( \mathcal{D} \cup \Delta \models \neg \mathcal{P} \)). If \( \Delta \) (wrong state transitions) is found then \( \Delta \) acts as a counter–example to the validity of \( \mathcal{P} \). Our approach allows us to distinguish not–specified (unspecified) elements at the current specification, which are considered as false in the classical models with two values (true and false) of specification, from elements specified as false.

It allows to define a multi–valued logic [2] with a new degree of satisfaction (called unspecified and denoted by \( \frac{1}{2} \)) of a property \( \mathcal{P} \) with the following meaning: current (partial) system specification does not satisfy \( \mathcal{P} \) and it also does not violate that property. Depending on how the system specification evolves, the property \( \mathcal{P} \) will be or not satisfied [10]. From this reasoning, we can obtain diagnostic information \( (\Delta) \) including potential wrong state transitions and a set of evolutions which make that \( \mathcal{P} \) is violated.

2.4 Revision phase

We use a method similar to the proposed one in [6] to generate appropriate system behaviours \( (\Delta') \) from the diagnostic information \( (\Delta) \) since \( \Delta' \) is generated by changing some elements of the counter–example \( \Delta \), which we obtain by using a multi–valued model–checker [4, 5]. Therefore, we can obtain alternative systems refinements \( (\Delta') \) from \( \Delta \) by only changing the set of evolutions which make that \( \mathcal{P} \) is violated. It allows to generate a new specification \( \mathcal{D}' \) consistent with \( \mathcal{D} \) since \( \mathcal{P} \) is not satisfied by \( \mathcal{D} \) because \( \mathcal{P} \) involves elements unspecified in \( \mathcal{D} \).

Alternative systems refinements \( (\Delta') \) obtained from the diagnostic information \( (\Delta) \) can be translated into system requirements refinements \( (\mathcal{R}') \), getting close the diagnostic information to the system domain of each viewpoint, bridging the gap between analysis and revision.

3 Illustrating the approach

In this paper, we illustrate our approach by using two viewpoints of a thermostat system\(^2\). This example is used to demonstrate how to refine requirements specifications of viewpoints which are being developed in an incremental way.

3.1 Viewpoints specification: SCTL-MUS

The first viewpoint \( (\mathcal{V}_{\text{heater}}) \) describes a very simple thermostat that can run a heater if the temperature falls below desired. The system has two indicators \((\text{below}_\text{on} \text{ and } \text{below}_\text{off})\), a switch to turn it off and on \((\text{off} \text{ and } \text{running})\) and a variable indicating whether the heater is running \((\text{heat}_\text{on})\).

A (possibly incorrect) description \( \mathcal{D}_{\text{heater}} \) of our thermostat includes the following SCTL requirements:

\[
\begin{align*}
\mathcal{R}_1 &\equiv \text{running} \Rightarrow \neg \text{running} \\
\mathcal{R}_2 &\equiv \text{running} \Rightarrow \neg \text{running} \\
\mathcal{R}_3 &\equiv \text{below}_\text{on} \land \neg \text{below}_\text{off} \\
\mathcal{R}_4 &\equiv \text{heat}_\text{on} \equiv \text{heat}_\text{off} \\
\mathcal{R}_5 &\equiv \text{off} \Rightarrow \text{below}_\text{on} \land \text{below}_\text{off} \\
\end{align*}
\]

From the specification of these five SCTL requirements, we can synthesize a MUS graph representing the specified behaviour. It is made by using the synthesis algorithm developed in the SCTL-MUS methodology. Figure 1 shows the MUS graph \( \mathcal{M}_{\text{heater}} \) of the system whose description \( \mathcal{D}_{\text{heater}} \) is given by the SCTL requirements \( \{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4, \mathcal{R}_5\} \). The system modeled by \( \mathcal{M}_{\text{heater}} \) starts in state \( E_1 \) and transitions into \( E_2 \) when it is turned on (the action \( \text{running} \) occurs), where it awaits the reading of the temperature indicator. Once the temperature is determined, the system transitions into \( E_3 \), if the \( \text{below}_\text{on} \) indicator is activated or remains into \( E_2 \), in other case. From both states \( E_2 \) and \( E_3 \), the system can be turned off. Unspecified actions are omitted from figures to simplify them. For example, in the state \( E_1 \) of the MUS graph \( \mathcal{M}_{\text{heater}} \), the action \( \text{running} \) is possible (it is specified as true, \( E_1[\text{running}] = 1 \)), the action \( \text{off} \) is unspecified (\( E_1[\text{off}] = \frac{1}{2} \)) and the actions \( \text{below}_\text{on}, \text{below}_\text{off} \) and \( \text{heat}_\text{on} \) are specified as false (\( E_1[\text{below}_\text{on}] = 0 \), \( E_1[\text{below}_\text{off}] = 0 \) and \( E_1[\text{heat}_\text{on}] = 0 \)).

---

\(^2\)This example is adopted from [4].

![Figure 1. MUS graph \( \mathcal{M}_{\text{heater}} \) of \( \mathcal{V}_{\text{heater}} \)](image-url)
the thermostat system: running the air conditioner. The behaviour of the system modeled by $M_{\text{air}}$ is similar to that of the heater, with one difference: this system handles the failure of the temperature indicator.

A (possibly incorrect) description $D_{\text{air}}$ of this aspect of the thermostat includes the following SCTL requirements:

$$\begin{align*}
R_6 & \equiv \text{running} \Rightarrow \neg \text{above}_\text{on} \land \text{off} \land \text{read_error} \land \neg \text{running} \\
R_7 & \equiv \text{above}_\text{on} \Rightarrow \neg \text{above}_\text{off} \land \text{off} \land \text{read_error} \land \text{air}_\text{on} \land \neg \text{running} \\
R_8 & \equiv \text{above}_\text{off} \Rightarrow \neg \text{above}_\text{on} \\
R_9 & \equiv \text{read_error} \Rightarrow \neg \text{read_ok} \land \text{off} \land \neg \text{running} \\
R_{10} & \equiv \text{read_ok} \Rightarrow \neg \text{above}_\text{on} \\
R_{11} & \equiv \text{air}_\text{on} \Rightarrow \neg \text{air}_\text{on} \lor \text{above}_\text{on} \\
R_{12} & \equiv \text{off} \Rightarrow \neg \text{running} \land \neg \text{above}_\text{off} \land \neg \text{above}_\text{on} \land \neg \text{air}_\text{on}
\end{align*}$$

The MUS graph $M_{\text{air}}$, synthesizing these seven SCTL requirements $\{R_6, R_7, R_8, R_9, R_{10}, R_{11}, R_{12}\}$ is shown in figure 2. In this case, if the temperature cannot be obtained in states $E_2$ or $E_4$, the system transitions into state $E_5$.

![Figure 2. MUS graph $M_{\text{air}}$ of $V_{\text{air}}$](image)

### 3.2 Merging viewpoints

The next step consists in obtaining the merged model of both viewpoints. Given a set of MUS viewpoints, they can be combined as we have explained in section 2.2, obtaining in our case the merged model in figure 3. It shows the MUS graph $M_D$ describing the behaviour of the thermostat that can run both the heater and the air conditioner.

![Figure 3. MUS graph $M_D$ of the merged model](image)

#### 3.3 Analysis phase over the merged model

The system property that we would like the merged model $D$ to satisfy is given by the following SCTL requirement $\mathcal{P}$:

$$\begin{align*}
\mathcal{P} & \equiv \text{running} \Rightarrow \neg \text{above}_\text{off} \land \neg \text{below}_\text{off} \\
& \equiv \text{running} \Rightarrow \neg \text{above}_\text{off} \land \neg \text{below}_\text{off}
\end{align*}$$

To find a set ($\Delta$) of state transitions such that $D \cup \Delta \models \neg \mathcal{P}$ we first find the states of the MUS graph $M_D$ in which $\mathcal{P}$ is applicable. These states are obtained by verifying the premise of $\mathcal{P}$ in each state of $M_D$. Only the states in which the degree of satisfaction of the premise of $\mathcal{P}$ is smaller than $\frac{1}{2}$ are considered states of applicability for the property $\mathcal{P}$ (denoted by $E_{a,\mathcal{P}}$), since according to the semantics defined by SCTL, requirements whose premise cannot be satisfied (0, $\frac{1}{2}$ or $\frac{3}{2}$) have no sense.

The multi-valued model-checker developed in the SCTL-MUS methodology uses the operation called causal (denoted by $\rightarrow$) to obtain (recursively) the degree of satisfaction of a SCTL requirement from the degree of satisfaction of its premise and consequence (see table 1, where first operand (column) is the degree of satisfaction of the Premise, and second operand (row) is the degree of satisfaction of the Consequence). The operation called atomic satisfaction$^4$ (denoted by $\uparrow$) is used to obtain the degree of satisfaction of an action ($0, \frac{1}{2}, 1$).

$^4 a \uparrow b = 1$ (if $a = b$) or $a \uparrow b = \frac{1}{2}$ (if $a \neq b$); $0$ (otherwise).

First operand is the action specific to the current system MUS graph, and second operand is the action specific to the current system MUS graph.
Table 1. Function causal (→)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1/2</th>
<th>1/4</th>
<th>3/4</th>
<th>1/2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1/2</td>
<td>1</td>
<td>1/4</td>
<td>3/4</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>1/2</td>
<td>1</td>
<td>1</td>
<td>1/4</td>
<td>3/4</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>1/4</td>
<td>1</td>
<td>1</td>
<td>1/4</td>
<td>3/4</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>3/4</td>
<td>1</td>
<td>1</td>
<td>1/4</td>
<td>3/4</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>1/2</td>
<td>1</td>
<td>1</td>
<td>1/4</td>
<td>3/4</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1/4</td>
<td>3/4</td>
<td>1/2</td>
<td>1</td>
</tr>
</tbody>
</table>

Premise of property \( P \) (running) is not satisfied in the states \( E_2, E_3, E_4 \) and \( E_5 \). However, it is satisfied in state \( E_1 \) because this action is specified as true in that state. Therefore, \( \mathcal{E}_{appr} = \{ E_1 \} \).

Once we have the states of applicability of the property \( P \), the procedure to find \( \Delta \) starts by negating the property \( P \) to obtain \( \neg P \). Applying the theorem 1:

**Theorem 1** Let \( R_1 = \neg (P \oplus C) \) and \( R_2 = P \oplus \neg C \) be two SCTL requirements, and let \( E_j \) a state of a MUS graph \( \mathcal{M} \) such that \( \models (P, E_j)^5 > \frac{1}{2} \models (P, E_j) \in \{ \frac{1}{2}, \frac{3}{4}, 1 \} \), then:

\[
\models (R_1, E_j) = \models (R_2, E_j)
\]

we obtain:

\[
\neg P \equiv \neg (\text{running} \Rightarrow \bigcirc \neg \text{above,off} \land \neg \text{below,off})
\equiv \text{running} \Rightarrow \bigcirc \text{above,off} \lor \text{below,off}
\]

To find \( \Delta \) such that \( D \cup \Delta \models \neg P \), we check the satisfaction of \( \neg P \) in all \( E_j \in \mathcal{E}_{appr} \).

Equation 1 describes the steps made by the model-checker to obtain the degree of satisfaction of property \( \neg P \) in state \( E_1 \):

\[
\models (\neg P, E_1) = \models (\text{running}, E_1) \rightarrow \models (\text{above,off}, E_2) \lor \models (\text{below,off}, E_2)
\]

\[
= \models (1, E_1[\text{running}]) \rightarrow \models (1, E_2[\text{above,off}]) \lor \models (1, E_2[\text{below,off}])
\]

\[
= \models (1, 1) \rightarrow \models (1, \frac{1}{2}) \lor \models (1, \frac{1}{2})
\]

\[
= 1 \rightarrow \frac{1}{2} \lor \frac{1}{2} = 1 \rightarrow \frac{1}{2} = \frac{1}{2}
\]

Therefore, \( \neg P \) is unspecified in \( E_1 \). This means that system can evolve (in future refinements of the requirements specifications), without violating the current system description \( D \), making that property \( \neg P \) be satisfied in that state. It occurs for example (see equation 1) if \( E_2[\text{above,off}] \sim 1 \) (\( a \sim b \) means that \( a \) evolves into \( b \)). Only unspecified actions can evolve since our first goal is preserving the main requirements and properties. From this reasoning, we can obtain a counter-example \( \Delta \) to the validity of \( P \) which can be used as diagnostic information.

In our approach, the counter-example consists of a set of evolutions (losing unspecification) of the specification of unspecified actions, referred to as actions refinements (an action refinement is the specification as true or false of an unspecified action):

\[
\Delta = \{ E_2[\text{above,off}] \sim 1 \lor E_2[\text{below,off}] \sim 1 \}
\]

A crucial aspect of the analysis-revision cycle is how to use the diagnostic information provided (\( \Delta \)) to generate alternative system refinements (\( \Delta' \)) which can be included in the system description \( D \) to obtain a new system description \( D' \) that guarantees that \( \Delta \) is no longer an explanation for the violation of the property \( P \).

We have previously obtained a counter-example (\( \Delta \)) because the degree of satisfaction obtained of \( \neg P \) in \( E_1 \) was \( \frac{1}{2} \). This means that current (partial) system description \( D \) does not satisfy \( \neg P \) and it also does not violate that property, but system description \( D \) can be refined into a new system description \( D' \) which satisfies or violates \( \neg P \) (depending on how \( D \) is refined), being \( D' \) consistent with \( D \).

We have based on previous reasoning to obtain \( \Delta \). We can obtain from this diagnostic information an alternative system refinement \( \Delta' \) by changing the actions refinements of \( \Delta \) since these actions refinements make that \( \neg P \) is or not satisfied. Therefore,

\[
\Delta' = \{ E_2[\text{above,off}] \sim 0 \land E_2[\text{below,off}] \sim 0 \}
\]

is an alternative system refinement which makes that \( \Delta \) is no longer an explanation for the violation of the property \( P \). Therefore, the alternative system refinement \( \Delta' \) consists of two actions refinements.

Since the alternative system refinements (\( \Delta' \)) have been obtained by changing actions refinements (unspecified actions in the original system description) into the counter-examples (\( \Delta \)), these alternative system refinements are consistent (they are not in contradiction) with the system description \( D \). Therefore, we can obtain a new system description \( D' \) including the previous description \( D \) and the provided alternative system refinements. This is possible because the system MUS graph distinguishes actions which are specified in the current (partial) system description from
those which are not specified (the unspecified actions), in comparison with the classical models with only two specification values (true or false) where not-specified actions are considered as false. Figure 4 shows the MUS graph $\mathcal{M}_{\mathcal{D}}^*$ obtained when we applied the previous actions refinements.

### 3.4 Revision phase over each viewpoint

The next step in our example consists in transferring the changes we have made in the merged model to the initial models of the thermostat. Figures 5 and 6 show the MUS graphs $\mathcal{M}_{\mathcal{D}_{\text{heater}}}^*$ and $\mathcal{M}_{\mathcal{D}_{\text{air}}}^*$ (consistent with $\mathcal{M}_{\mathcal{D}_{\text{heater}}}$ and $\mathcal{M}_{\mathcal{D}_{\text{air}}}$, respectively) which satisfy the property $\mathcal{P}$. As we can see, the two viewpoints are affected by the changes carried out over the merged model, so in order to obtain that $\mathcal{P}$ is satisfied it is necessary to modify both models.

![Figure 4. MUS graph $\mathcal{M}_{\mathcal{D}}^*$](image1)

![Figure 5. MUS graph $\mathcal{M}_{\mathcal{D}_{\text{heater}}}^*$](image2)

![Figure 6. MUS graph $\mathcal{M}_{\mathcal{D}_{\text{air}}}^*$](image3)

Nevertheless, the form in which these alternative system refinements are expressed (by indicating actions refinements which are consistent with $\mathcal{D}_{\text{heater}}$ and $\mathcal{D}_{\text{air}}$) does not provide enough information to decide if they can be adopted by the stakeholders. Our approach allows to get close this kind of alternative system refinements to the system domain, facilitating to the stakeholders the decision of accepting the proposed alternative system refinements.

To achieve it, we translate the actions refinements of the alternative system refinements ($\Delta'$) into system requirements refinements ($\{\mathcal{R}_{\text{1}}', \ldots, \mathcal{R}_{\text{n}}'\}$), specializing the SCTL requirements which specify the descriptions $\mathcal{D}_{\text{heater}}$ and $\mathcal{D}_{\text{air}}$. This is possible because, as we have explained above, the alternative system refinements are consistent (they are not in contradiction) with the current (partial) system descriptions $\mathcal{D}_{\text{heater}}$ and $\mathcal{D}_{\text{air}}$.

This translation process can obtain different possible system requirements refinements. Stakeholders will be required to decide what of them are accepted. The main advantage of our approach here is that the diagnostic information ($\Delta$) is used to provide system refinement information ($\{\mathcal{R}_{\text{1}}', \ldots, \mathcal{R}_{\text{n}}'\}$) close to the system domain.

Next, we outline how to obtain the system requirements refinements from the alternative system refinements $\Delta'$. In general, given a set of SCTL requirements $\{\mathcal{R}_{\text{1}}, \ldots, \mathcal{R}_{\text{n}}\}$ specifying a system description $\mathcal{D}$, the SCTL-MUS methodology provides an algorithm which synthesizes a MUS graph $\mathcal{M}$ representing the specified behaviour. Every state $E_j$ of the MUS graph $\mathcal{M}$ has a link to the SCTL requirements ($\{\mathcal{R}_{E_j}\}$) which are synthesized in that state. Remark that, each of these requirements can be a specified SCTL requirement $\mathcal{R}_k$, or any SCTL requirement which is included in the specified ones. In particular, if a SCTL requirement $\mathcal{R}_k$ is synthesized in a state $E_j$ of $\mathcal{M}$, then the premise of $\mathcal{R}_k$ (another SCTL requirement denoted by $\mathcal{P}_{\mathcal{R}_k}$) is also synthesized in $E_j$, and its consequence (another SCTL requirement denoted by $\mathcal{U}_{\mathcal{R}_k}$) is synthesized in the states of applicability according to the temporal operator of $\mathcal{R}_k$. 


Let $\Delta = \{ E_2 [a_1] \rightarrow 1(0) \}$ be an alternative system refinement obtained according to the method previously proposed. It can be translated into a system requirement refinement $(R_k')$ as follows: if $R_k$ is synthesized in $E_j$ then the specification of the action $a_t$ (if it is unspecified in $P_{R_k}$) can be added as true (false) to the premise of $R_k$, obtaining a new system requirement refinement $R_k'$; if the consequence of $R_k$ is synthesized in $E_j$ then the specification of the action $a_t$ (if it is unspecified in $C_{R_k}$) can be added as true (false) to the consequence of $R_k$, obtaining a new system requirement refinement $R_k'$.

Returning to our example, tables 2 and 3 show the requirements synthesized in each state of the MUS graphs $M_{D_{heater}}$ and $M_{D_{air}}$, respectively.

<table>
<thead>
<tr>
<th>State</th>
<th>Synthesized Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_1$</td>
<td>${ R_1, P_{R_1}, C_{R_1} }$</td>
</tr>
<tr>
<td>$E_2$</td>
<td>${ C_{R_2}, R_2, P_{R_2}, C_{R_3}, C_{R_4}, R_5, P_{R_5} }$</td>
</tr>
<tr>
<td>$E_3$</td>
<td>${ C_{R_2}, R_3, P_{R_2}, R_4, P_{R_4}, C_{R_4}, R_5, P_{R_5} }$</td>
</tr>
</tbody>
</table>

Table 2. Requirements synthesized in each state of $M_{D_{heater}}$

<table>
<thead>
<tr>
<th>State</th>
<th>Synthesized Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_1$</td>
<td>${ R_6, P_{R_6}, C_{R_6} }$</td>
</tr>
<tr>
<td>$E_2$</td>
<td>${ C_{R_6}, R_7, P_{R_7}, C_{R_6}, R_9, P_{R_8}, C_{R_{10}}, C_{R_{11}}, }$</td>
</tr>
<tr>
<td>$E_3$</td>
<td>${ C_{R_6}, R_9, P_{R_8}, R_9, P_{R_8}, R_{11}, P_{R_{11}}, C_{R_{11}}, }$</td>
</tr>
<tr>
<td>$E_5$</td>
<td>${ C_{R_6}, R_{10}, P_{R_{10}}, R_{12}, P_{R_{12}} }$</td>
</tr>
</tbody>
</table>

Table 3. Requirements synthesized in each state of $M_{D_{air}}$

### 3.4.1 Requirements Specification Refinements of the Viewpoint $V_{heater}$

The alternative system refinement:

$$\Delta' = \{ E_2 [\text{above}\_\text{off}] \rightarrow 0 \land E_2 [\text{below}\_\text{off}] \rightarrow 0 \}$$

only affects the requirements synthesized in the state $E_2$. In this case, we only take into account the second refinement $(E_2 [\text{below}\_\text{off}] \rightarrow 0)$ because it is the only one that affects unspecified actions in the application domain of $V_{heater}$. Then, we will facilitate to the stakeholders the decision of accepting the proposed system requirements refinements over $V_{heater}$ that we can see in table 4.

<table>
<thead>
<tr>
<th>Proposed Refined Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1' \equiv \text{running} \Rightarrow \text{below}_\text{on} \land \text{off} \land \neg \text{running}$</td>
</tr>
<tr>
<td>$R_2' \equiv \text{below}_\text{on} \land \neg \text{below}_\text{off} \Rightarrow \text{below}_\text{on}$</td>
</tr>
<tr>
<td>$R_3' \equiv \text{below}_\text{off} \Rightarrow \text{below}_\text{on} \land \neg \text{running}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inconsistent Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_4' \equiv \text{heat}_\text{on} \Rightarrow (\text{heat}_\text{on} \lor \text{below}_\text{on}) \land \neg \text{below}_\text{off}$</td>
</tr>
<tr>
<td>$R_5' \equiv \text{off} \land \neg \text{below}_\text{off} \Rightarrow \text{running} \land \neg \text{below}_\text{off}$</td>
</tr>
</tbody>
</table>

Table 4. Obtained requirements refinements over $V_{heater}$ after verifying property $P$

The two last system requirements refinements $R_4'$ and $R_5'$ are not shown to the stakeholders because they both cause an inconsistency in the state $E_5$. For example, the premise of $R_5$ is synthesized in the states $E_2$ and $E_5$ (the action $\text{off}$ is specified as true in both states). If we add the refinement $\neg \text{below}\_\text{off}$ to $P_{R_5}$, we will have an inconsistency in the state $E_5$, because the action $\text{off}$ is already specified as true. As we can see, our approach allows us to check the consistency of the provided system requirements refinements.

In order to be able to continue reasoning over our example, we will suppose the stakeholders choose the system requirements refinement $R_4'$ from all proposed ones. The remaining requirements refinements ($\{ R_1', R_2' \}$) will be discarded. This choice is based on a consistent reasoning. Since the system requirements refinements are close to the system domain, this task is made easier. It seems logical, as we can see expressed in $R_3'$, that after the temperature is above the minimum, the below\_off indicator cannot be activated in the next state. This action will be possible again when the temperature falls below the minimum.

### 3.4.2 Requirements Specification Refinements of the Viewpoint $V_{air}$

From the same alternative system refinement:

$$\Delta' = \{ E_2 [\text{above}\_\text{off}] \rightarrow 0 \land E_2 [\text{below}\_\text{off}] \rightarrow 0 \}$$

now, we only take into account the first refinement $(E_2 [\text{above}\_\text{off}] \rightarrow 0)$ because it is the only one that affects unspecified actions in the application domain of $V_{air}$. In this case, we will facilitate to the stakeholders the decision...
of accepting the proposed system requirements refinements over $\mathcal{V}_{air}$ that we can see in table 5.

<table>
<thead>
<tr>
<th>Proposed Refined Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{R}_6 \equiv \text{running} \Rightarrow \bigcirc \text{above,}\text{on} \land \text{off} \land \text{readError} \land \neg \text{running} \land \neg \text{above,off}$</td>
</tr>
<tr>
<td>$\mathcal{R}_7 \equiv \text{above,}\text{on} \land \neg \text{above,off} \Rightarrow \bigcirc \text{above,off}$</td>
</tr>
<tr>
<td>$\mathcal{R}_8 \equiv \text{off} \land \text{readError} \land \text{air,}\text{on} \land \neg \text{running}$</td>
</tr>
<tr>
<td>$\mathcal{R}_9 \equiv \text{above,off} \Rightarrow \bigcirc \text{above,}\text{on} \land \neg \text{above,off}$</td>
</tr>
<tr>
<td>$\mathcal{R}_{10} \equiv \text{read,}\text{ok} \Rightarrow \bigcirc \text{above,}\text{on} \land \neg \text{above,off}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inconsistent Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{R}_6' \equiv \text{readError} \land \neg \text{above,off} \Rightarrow \bigcirc \text{read,}\text{ok} \land \text{off} \land \neg \text{running}$</td>
</tr>
<tr>
<td>$\mathcal{R}_7' \equiv \text{air,}\text{on} \Rightarrow \bigcirc (\text{air,}\text{on} \lor \text{above,}\text{on})$</td>
</tr>
<tr>
<td>$\mathcal{R}_8' \equiv \text{off} \land \neg \text{above,off}$</td>
</tr>
<tr>
<td>$\mathcal{R}_9' \equiv \text{above,off} \land \neg \text{above,off} \Rightarrow \bigcirc \text{running} \land \text{off} \land \neg \text{above,off} \land \neg \text{above,off}$</td>
</tr>
</tbody>
</table>

Table 5. Obtained requirements refinements over $\mathcal{V}_{air}$ after verifying property $\mathcal{P}$

The three last system requirements refinements $\mathcal{R}_6', \mathcal{R}_7', \text{ and } \mathcal{R}_9'$ are not shown to the stakeholders because they cause an inconsistency in the state $E_3$.

In order to be able to continue reasoning over our example, we will suppose, in this case, the stakeholders choose the system requirements refinement $\mathcal{R}_6'$ from all proposed ones. The remaining requirements refinements ($\{\mathcal{R}_6, \mathcal{R}_7, \mathcal{R}_{10}\}$) will be discarded. It seems logical, as we can see expressed in $\mathcal{R}_6'$, that after the temperature is below the maximum, the above,off indicator cannot be activated in the next state. This action will be possible again when the temperature rises above the maximum.

Therefore, if we apply again the analysis-revision cycle in order to obtain the merged model satisfying another property, we would have to refine the new requirements specifications of $\mathcal{V}_{heater}$ and $\mathcal{V}_{air}$, given by the sets of SCTL requirements $\{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4, \mathcal{R}_5\}$ and $\{\mathcal{R}_6, \mathcal{R}_7, \mathcal{R}_8, \mathcal{R}_9, \mathcal{R}_{10}, \mathcal{R}_{11}, \mathcal{R}_{12}\}$, respectively.

4 Conclusions and Future Work

Multiple viewpoints are often used in Requirements Engineering to facilitate traceability to stakeholders, to structure the requirements process, and to provide richer modeling by incorporating multiple conflicting descriptions. In viewpoint–based requirements engineering, an emphasis is placed on capturing separate descriptions of the viewpoints of different stakeholders, and on identifying and resolving conflicts between them.

On the other hand, the development of requirements specifications can be supported by a cycle composed of two phases: analysis and revision. The analysis phase is responsible for checking whether a number of desirable properties of a system is satisfied by its partial specification. It also provides appropriate diagnostic information when a certain property is violated by the specification. The revision phase should change the given specification into a new (partial) specification – by making use of the diagnostic information obtained from the analysis phase – in such a way that the new specification no longer violates the system’s property in question.

In this paper, we have used the SCTL-MUS methodology to facilitate the evolution of requirements specifications of state transition systems in a multi-perspective environment. We have adopted the analysis-revision cycle, showing the advantages of using a third value of specification, which allows us to distinguish not-specified elements (partial specification) from those which are specified as false.

The principal advantages provided by our approach are the following ones:

- Using a third value of specification allows to define a multi–valued logic with a new degree of satisfaction (called unspecified and denoted by $\frac{1}{2}$) of a property $\mathcal{P}$ with the following meaning: current (partial) system specification does not satisfy $\mathcal{P}$ and it also does not violate that property. Depending on how the system specification evolves, the property $\mathcal{P}$ will be or not satisfied. From this reasoning, we can obtain diagnostic information ($\Delta$) including potential wrong state transitions and a set of evolutions which make that $\mathcal{P}$ is violated.

- We can obtain alternative systems refinements ($\Delta'$) from $\Delta$ by only changing the set of evolutions which make that $\mathcal{P}$ is violated. It allows to generate a new specification $\mathcal{D}'$ consistent with $\mathcal{D}$ since $\mathcal{P}$ is not satisfied by $\mathcal{D}$ because $\mathcal{P}$ involves elements unspecified in $\mathcal{D}$.

- It allows to translate the diagnostic information into system requirements refinements close to the system domain of each viewpoint, facilitating to the stakeholders the decision of what system requirements refinements must be included in the system requirements specification.

- It allows to check the consistency of the provided system requirements refinements.

To illustrate these advantages, we have employed two viewpoints of a thermostat system. Both viewpoints have been merged in order to be able to reason over the properties of the composed system. The refinements over the merged
model have been transferred into each one of the viewpoints and then into the requirements in which are expressed both ones. So, we have obtained requirements refinements close to the system domain of each viewpoint.

Nowadays, our work is focused on formalizing a method to select the possible system requirements refinements. The translation process, from alternative system refinements into system requirements refinements can obtain several different system requirements refinements, as we have shown in previous sections. It is necessary to filter the system requirements refinements, so stakeholders can reason about an adequate number of possible refinements. One possibility is to select system requirements refinements which include a simple action refinement since they can be easily adopted by the stakeholders. Nevertheless, it is also interesting to select the system requirements refinements which translate the provided alternative system refinements (Δ') into the minimal specializations of the system requirements.

Moreover, we are always obtaining the counter-examples from evolutions in the actions of the consequence. For the moment, we have obviated the cases in which the premise of the property is unspecified. It also constitutes a line of future work.

On the other hand, in order to support real time requirements, we are defining an extension of MUS (MUS-T [16]) which introduces time as a dense domain, since dense models are more expressive and suitable for composition and refinement [1]. It is also our goal extending the SCTL logic (SCTL-T) in order to express time restrictions in a requirement.

References