International Journal of Software Engineering and Knowledge Engineering
© World Scientific Publishing Company

PROCEDURES AND ALGORITHMS FOR CONTINUOUS INTEGRATION IN AN AGILE SPECIFICATION ENVIRONMENT

MARTÍN LÓPEZ-NORES, JOSÉ J. PAZOS-ARIAS, JORGE GARCÍA-DUQUE, YOLANDA BLANCO-FERNÁNDEZ, REBECA P. DÍAZ-REDONDO, ANA FERNÁNDEZ-VILAS, ALBERTO GIL-SOLLA and MANUEL RAMOS-CABRER

Department of Telematics Engineering, University of Vigo
ETSE Telecomunicación, Campus Universitario s/n, 36310 Vigo (Spain)
mlnores@det.uvigo.es
jose@det.uvigo.es
jgd@det.uvigo.es
yolanda@det.uvigo.es
rebeca@det.uvigo.es
avilas@det.uvigo.es
agil@det.uvigo.es
mramos@det.uvigo.es

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

One of the main ideas of agile development is to perform *continuous integration*, in order to detect and resolve conflicts among several modular units of a system as soon as possible. Whereas this feature is well catered for at the level of programming source code, the support available in formal specification environments is still rather unsatisfactory: it is possible to analyze the composition of several modular units automatically, but no assistance is given to help modify them in case of problems. Instead, the stakeholders who build the specifications are forced to attempt manual changes until getting to the desired functionality, in a process that is far from being intuitive. In response to that, this paper presents procedures and algorithms that automate the whole process of doing integration analyses and generating revisions to solve the diagnosed problems. These mechanisms serve to complete an agile specification environment presented in a previous paper, which was designed around the principle of facilitating the creative efforts of the stakeholders.

Keywords: Formal specification; agile software development; continuous integration.

1. Introduction

The term *agile development* in software engineering refers to methodologies in which humans make frequent small changes to a system description, and where the continuous analysis of what has been done is the key to increase the knowledge about the desired system and advance its development [1, 5]. Many case studies [a] have proved that the principles of agility contribute to enhancing the productivity of programming tasks. Knowing that, there

[a] A growing list can be found at *http://www.agilejournal.com*, section "*Case studies*".

is a growing trend towards adopting the agile approach in formal specification settings, either for requirements engineering and modeling [3] or as a supplement to the programming itself [4, 21]. However, the approaches presented thus far in the field of *agile formal methods* [6, 8, 9, 11, 24, 33, 34, 37] merely focused on adopting some of the recommended best practices of agile programming methodologies (mainly those of *eXtreme Programming*), paying little or no attention to many important technical features.

In [19], we identified some of the technical flaws of agile formal methods, and introduced solutions with a twofold objective: to facilitate the creative work of developing formal specifications, and to deal effectively with changeable specifications. In this paper, we tackle an unresolved question related to one of the core ideas of agile development: *continuous integration* (CI). As explained in [13], CI is about analyzing the functionality that results from composing several modular units of a system, to detect and resolve problems as soon as possible. It is well known that the correctness of a system is not guaranteed by the correctness of its parts, considered in isolation; rather the opposite, the interactions among different functional features lead to the so-called *emergent behavior*, which can only be observed and analyzed over the composition of those parts [23, 28]. Thereby, a suitable development methodology should allow analyzing compositions and, in case some problems are detected, provide support for modifying the components involved.

When it comes to programming source code, the support available for CI is improving daily, with mechanisms like automated builds and self testing. In contrast, the support available in formal specification is rather deficient. On the one hand, many of the previous works on agile formal methods do not even allow organizing a specification in components, which forces the stakeholders to handle complex artifacts that gather all the functional features of a system. On the other hand, the few approaches that support modularization only inform the stakeholders of whether a specification passes or fails the integration analyses; thus, locating the source of the problems and modifying the composed units remains an entirely manual task, which is not intuitive at all. Out of the agile wave, we only find better support in [18, 29, 32], where some mechanisms were introduced to automatically evaluate changes at the composition level that would solve the problems detected in the analyses. As shown in Fig. 1, given a system $\mathcal{S}$ that results from composing $\mathcal{C}_1, \ldots, \mathcal{C}_n$, those mechanisms would be able to generate a revised version $\mathcal{S}^{rev}$ that solves any problems detected over $\mathcal{S}$. Unfortunately, there is no support to automatically modify the original $\mathcal{C}_1, \ldots, \mathcal{C}_n$ so that their composition yields $\mathcal{S}^{rev}$. Neither is there an answer to whether the changes made over $\mathcal{S}$ could require introducing new modular units.

These limitations (which we refer to as the *effective loss of modularization*) represent an important shortcoming in agile development, because they prevent CI. Indeed, the stakeholders have to choose between three equally bad alternatives:

(1) Not to do integration analyses until the modular units have been fully developed, which may lead to discovering errors late and, thereby, to significant increases in time and cost.
(2) To do integration analyses once, and then continue developing the system as a monolithic whole, with null separation of concerns.

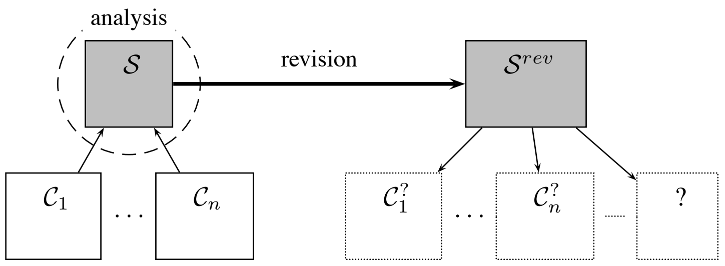


Figure 1. The effective loss of modularization after integration analysis.

(3) To do integration analyses often, attempting costly manual changes every time to maintain the modularization.

In this paper, we define the means to adequately support CI in formal specification environments, allowing the stakeholders to do integration analyses as often as they need, with the maximum automated support. To this aim, we define procedures that solve the effective loss of modularization, automating the whole process of doing integration analysis and translating any changes made to the composition into modifications of the components involved, or into new modular units if so preferred by the stakeholders. These procedures are introduced in general terms in Section 2. Then, following the introductory background of Section 3, we instantiate the proposal in Section 4 over specific formalisms, and use them to exemplify our approach to CI in Section 5. We discuss the results of practical experiments in Section 6, and the paper finishes with a summary of conclusions in Section 7.

2. Continuous Integration in Agile Formal Specification

The methodology of [19], which is the context for our approach to CI, was designed to help stakeholders develop requirements specifications in an iterative fashion, getting guidance from correctness checks performed during the intermediate stages. There, for the sake of generality, we considered a broad notion of software component, combining a set of *requirements* (e.g. expressed in temporal logic), a set of operational *models* (e.g. Kripke structures or labeled transition systems) and a set of *scenarios* (e.g. message sequence charts). Such a component will be hereafter represented as in Fig. 2.

$\mathcal{C}$

$$\boxed{\begin{array}{c} \mathcal{R}eq(\mathcal{C}) = \{\mathcal{R}_1, \mathcal{R}_2, \ldots\} \\ \hline \mathcal{M}od(\mathcal{C}) = \{\mathcal{M}_1, \mathcal{M}_2, \ldots\} \\ \hline \mathcal{S}ce(\mathcal{C}) = \{\mathcal{E}_1, \mathcal{E}_2, \ldots\} \end{array}}$$

Figure 2. Notation for components.

As shown in Fig. 3, development in [19] was posed as a cycle with two phases:

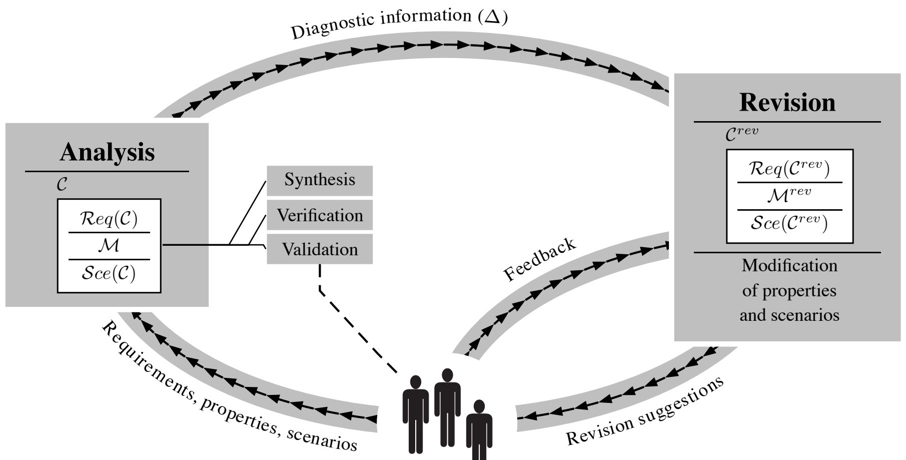

Diagnostic information (Δ)
Analysis
$\mathcal{C}$
$\mathcal{R}eq(\mathcal{C})$
$\mathcal{M}$
$\mathcal{S}ce(\mathcal{C})$
Synthesis
Verification
Validation
Revision
$\mathcal{C}^{rev}$
$\mathcal{R}eq(\mathcal{C}^{rev})$
$\mathcal{M}^{rev}$
$\mathcal{S}ce(\mathcal{C}^{rev})$
Modification of properties and scenarios
Feedback
Requirements, properties, scenarios
Revision suggestions

Figure 3. The approach of [19] for an agile analysis-revision cycle.

- In the **Analysis** phase, an operational model of the current specification is used to perform certain correctness checks, including (i) the *synthesis* of the model itself from the requirements, (ii) the *verification* of desirable safety or liveness properties through model-checking, and (iii) *validation* through manual inspection or against usage scenarios. In all cases, we gather *diagnostic information* for the problems detected, with each piece pointing out possible ways to alter parts of the model.
- The **Revision** phase uses the diagnostic information to automatically derive possible evolutions of the requirements, properties and scenarios provided by the stakeholders. If those evolutions solve (at least, partially) some of the problems, they are later presented to the stakeholders as *revision suggestions*, that they can accept, reject or ignore. The methodology learns from the stakeholders' decisions, maintaining a knowledge base to avoid insisting on unwanted solutions.

With the mechanisms presented in [19], we solved the problem of transforming a component $\mathcal{C}$ that did not pass a given analysis into a revised component $\mathcal{C}^{rev}$ which does. In Fig. 1, therefore, we can already take the step from $\mathcal{S}$ to $\mathcal{S}^{rev}$. What is missing is a way to present the changes to the stakeholders that overcomes the effective loss of modularization. Prior to explaining our proposal in this regard, we must introduce a novelty in the characterization of the specifications.

When managing modular specifications, it is important to note that compositionality stems from the operational models, as it is possible to define operators that, given models of the functionality of several components, return a model of their conjoint functionality. The same is not true for requirements, because joining those of several components does not yield a set of requirements for their composition —specifically, there would be no requirements specifying the emergent behavior. From this observation, we introduce here a distinction between two types of components, which is illustrated in Fig. 4:

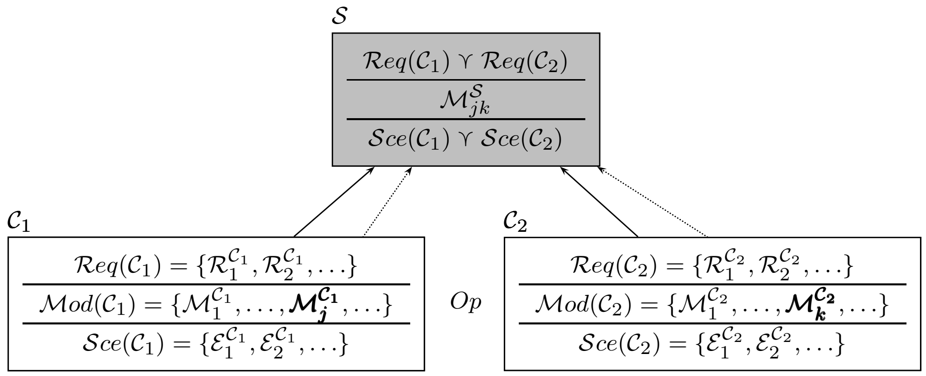

$\mathcal{S}$
$\mathcal{R}eq(\mathcal{C}_1) \curlyvee \mathcal{R}eq(\mathcal{C}_2)$
$\mathcal{M}^{\mathcal{S}}_{jk}$
$\mathcal{S}ce(\mathcal{C}_1) \curlyvee \mathcal{S}ce(\mathcal{C}_2)$
$\mathcal{C}_1$
$\mathcal{R}eq(\mathcal{C}_1) = \{\mathcal{R}^{\mathcal{C}_1}_1, \mathcal{R}^{\mathcal{C}_1}_2, \ldots\}$
$\mathcal{M}od(\mathcal{C}_1) = \{\mathcal{M}^{\mathcal{C}_1}_1, \ldots, \boldsymbol{\mathcal{M}^{\mathcal{C}_1}_j}, \ldots\}$
$\mathcal{S}ce(\mathcal{C}_1) = \{\mathcal{E}^{\mathcal{C}_1}_1, \mathcal{E}^{\mathcal{C}_1}_2, \ldots\}$
Op
$\mathcal{C}_2$
$\mathcal{R}eq(\mathcal{C}_2) = \{\mathcal{R}^{\mathcal{C}_2}_1, \mathcal{R}^{\mathcal{C}_2}_2, \ldots\}$
$\mathcal{M}od(\mathcal{C}_2) = \{\mathcal{M}^{\mathcal{C}_2}_1, \ldots, \boldsymbol{\mathcal{M}^{\mathcal{C}_2}_k}, \ldots\}$
$\mathcal{S}ce(\mathcal{C}_2) = \{\mathcal{E}^{\mathcal{C}_2}_1, \mathcal{E}^{\mathcal{C}_2}_2, \ldots\}$

Figure 4. White box and gray box components.

- We call **white box** any component whose functionality is completely determined by a set of requirements provided by the stakeholders. That set can be generally implemented by multiple operational models, though only one is usually handled (by default, the simplest one according to some criterion) [20, 35]. Components $\mathcal{C}_1$ and $\mathcal{C}_2$ in Fig. 4 are white boxes, just like the components we considered in [19].
- We call **gray box** any component whose functionality is represented by models which are not directly obtained from requirements provided by the stakeholders, but rather by composing or transforming other models. Nonetheless, the component contains requirements that specify part of that functionality, hence we do not talk of *black boxes* —that name would be applied to models which do not relate to requirements provided by the stakeholders (for instance, because they were constructed manually, or they have been reused without retrieving the requirements from which they were generated).

  Figure 4 represents the gray box $\mathcal{S}$ that results from composing the j-th model of $\mathcal{C}_1$ and the k-th one of $\mathcal{C}_2$ by means of a certain operator Op. The $\curlyvee$ symbols denote some operation between the sets of requirements and scenarios of $\mathcal{C}_1$ and $\mathcal{C}_2$, whose result generally depends of the selected models and the composition operator itself.

Having said that, we are already in the position to explain our approach to CI. Given a composition $\mathcal{S} = \mathcal{C}_1 \ldots Op \ldots \mathcal{C}_n$, the integration analysis is done over an operational model of $\mathcal{S}$. In case of problems, we use the diagnostic information to obtain a revised model that passes the analysis, and put it into a gray box $\mathcal{S}^{rev}$. Finally, when it comes to discharging the changes implied by the transformation of $\mathcal{S}$ into $\mathcal{S}^{rev}$ over the original modularization, we consider the two possibilities of Fig. 5:

(1) The first option is to discharge the changes entirely over some or all of the original components, to obtain revised components $\mathcal{C}^{rev}_1, \ldots, \mathcal{C}^{rev}_n$ that jointly bear the func-

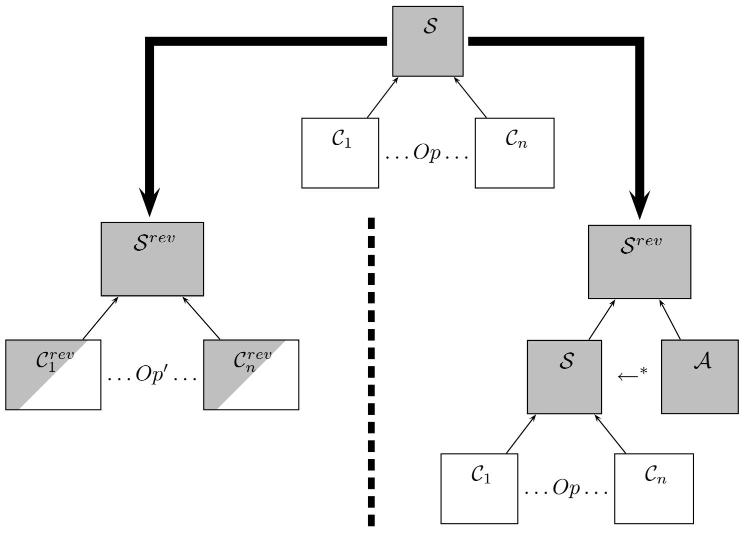


Figure 5. Possibilities to discharge the changes done at the composition level.

tionality of $\mathcal{S}^{rev}$. To this aim, the model of $\mathcal{S}^{rev}$ is split into modified versions of the models of $\mathcal{C}_1, \dots, \mathcal{C}_n$ that bore the original model of $\mathcal{S}$. This process generally leads to greater coupling between the components, which is illustrated in Fig. 5 in the change of the composition operator Op for Op'. Next, we modify the requirements and scenarios of $\mathcal{C}_1, \dots, \mathcal{C}_n$ to reflect the changes made to their models, considering the evolution types defined in [19]. In doing so, if the original components were white boxes, the revised ones can be characterized as gray ones, because part of the functionality of some $\mathcal{C}_j$ may become dependent on the functionality of another $\mathcal{C}_k$.

(2) The second option is to introduce as a new component that, combined with the composition of the original $\mathcal{C}_1, \dots, \mathcal{C}_n$, bears the functionality of $\mathcal{S}^{rev}$. The new component captures the crosscutting concerns that drive the integration analysis, and so we call it an *aspect* (hence the $\mathcal{A}$ in Fig. 5). Within this vision, we proceed by splitting the model of $\mathcal{S}^{rev}$ into two parts, one of which must be equal to the original model of $\mathcal{S}$; the other is put into a gray box for $\mathcal{A}$. The original components remain unchanged, and the same happens with the way they were composed.

In any of the two options, the changes implied by the model of $\mathcal{S}^{rev}$ over the original model of $\mathcal{S}$ are translated into specializations of any scenarios that were being considered at the composition level, applying follow-up mechanisms like the ones introduced in [26, 36]. The scenarios play an important role in CI, because their most common use it precisely to capture interactions between several modular units [15, 38].

To complete the approach to CI, the role of the analysis-revision cycle is to help the stakeholders browse the multiple revision possibilities that may exist in a general case. The scheme we propose to achieve this goal is analogous to the one we presented in [19]: an iterative procedure that prioritizes the simplest alternatives (even to the point of implying partial solutions to some of the diagnosed problems) and that learns from the interaction with the stakeholders. Importantly, it is always up to the stakeholders to select which option to use when discharging the changes from integration analysis. It will be clear from the example of Section 5 that the different possibilities of Fig. 5 lead to different views of the revised system and its components, and we want the stakeholders to handle the artifacts that make it easier for them to go on with the specification tasks.

3. Background on the SCTL-MUS Methodology

In Section 4, we shall explain how to implement our approach to CI in SCTL-MUS, a formal specification methodology introduced in [25], and enhanced later in [19] to adopt an agile approach. Prior to that, we provide some background on SCTL-MUS to help understand the posterior procedures and algorithms. Further details can be found in our previous publications.

3.1. *The requirements*

The requirements are expressed in a temporal logic called SCTL (*Simple and Causal Temporal Logic*), which allows stating conditions about the enabledness of the events that may occur during the operation of a system. The requirements follow the causal pattern $Premise \bigoplus Consequent$, with the symbol $\bigoplus$ representing one temporal operator from the set $\{\Rightarrow, \Rightarrow \bigcirc, \Rightarrow \bigodot\}$, to be interpreted as follows:

If $Premise$ is satisfied, then [simultaneously ($\Rightarrow$) $|$ immediately after ($\Rightarrow \bigcirc$) $|$ immediately before ($\Rightarrow \bigodot$)] $Consequent$ must be satisfied.

In addition to temporal operators, the requirements may include the logical connectives AND (denoted by $\wedge$), OR ($\vee$) and NOT ($\neg$).

3.2. *The operational models*

The operational models are expressed in a sort of state-machine formalism called MUS (*Model of Unspecified States*), a variant of the classical *Labeled Transition Systems* [22] prepared to model systems whose specification has not been completed yet. Thus, given a set of SCTL requirements, the synthesis algorithm presented in [25] generates a MUS model containing (i) *possible* transitions through the events specified as *true*, (ii) *non-possible* transitions through the events specified as *false*, and (iii) *unspecified* transitions corresponding to events not yet affected by the requirements ($\perp$).

Formally, a MUS model is a triple $\mathcal{M} = (S_{\mathcal{M}}, \Lambda, \rightarrow)$ with $S_{\mathcal{M}}$ a set of states, Λ a set of events (an alphabet), and $\rightarrow \subseteq S_{\mathcal{M}} \times \Lambda \times S_{\mathcal{M}} \times \mathcal{L}_3$ a transition relation. A generic element of this relation, denoted by (s_i, a, s_j, v) or $s_i \xrightarrow{\{a,v\}} s_j$, indicates a transition from state s_i into state s_j through event a, whose enabledness is given by the value $v \in \mathcal{L}_3 = \{\textit{false}, \perp, \textit{true}\}$. The set of states $S_{\mathcal{M}}$ includes a fixed one, called the *unspecified state* and denoted by s_{unsp}, that captures all the not-yet-specified states and transitions of the model.

Example 1. Figure 6 shows a MUS model whose alphabet is $\Lambda = \{a, b\}$. From the initial state of this model, it is possible to transition into s_2 through the event a ($s_1[a]_{\mathcal{M}} = \textit{true}$), whereas b is not allowed to occur ($s_1[b]_{\mathcal{M}} = \textit{false}$). In state s_2, a is an *unspecified* event ($s_2[a]_{\mathcal{M}} = \perp$), but event b is possible ($s_2[b]_{\mathcal{M}} = \textit{true}$) and takes the model back to s_1.

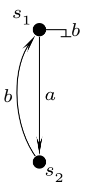


Figure 6. A sample MUS model.

Notation

- The initial state of a MUS graph is denoted by s_1.
- Possible transitions $s_i \xrightarrow{\{a,\textit{true}\}} s_j$ are represented by an a-labeled arrow from s_i to s_j. One example is $s_1 \xrightarrow{\{a,\textit{true}\}} s_2$ in Fig. 6; s_2 is called the *a-successor* of s_1.
- Unspecified transitions (like $s_2 \xrightarrow{\{a,\perp\}} s_{unsp}$ in Fig. 6) are not represented graphically.
- Non-possible transitions $s \xrightarrow{\{a,\textit{false}\}} s_{unsp}$ are represented by placing a symbol like $\neg\!\!\perp$ next to event a in state s. One example is $s_1 \xrightarrow{\{b,\textit{false}\}} s_{unsp}$ in Fig. 6.

3.3. *The scenarios*

The goal of a scenario is to capture traces of a system's functionality by means of event sequences. To this aim, the SCTL-MUS methodology employs a formalism called SLS (*Simple Language of Scenarios*), inspired by the classical *Message Sequence Charts* [17] but tailored to the characteristics of SCTL and MUS.

Every element of the sequence defined by an SLS scenario specifies an event as possible or non-possible. Following the classical notion introduced in [35], we define a materialization of an SLS scenario over a MUS model as any trace of the latter that contains the events of the scenario in the specified order and with the specified enabledness, maybe with occurrences of unrelated events in between.

Example 2. Figure 7(a) represents an SLS scenario that starts with event a_1, after which d_1 should never occur before b_1; b_1 is followed by c_1 and, finally, by d_1. This scenario has several materializations over the MUS model of Fig. 7(b), like the ones represented by thick lines over the traces $(s_1, s_2, s_6, s_{10}, s_{11}, s_{12}, s_{11})$ and $(s_1, s_2, s_6, s_{10}, s_{14}, s_{15}, s_{16}, s_{15})$.

3.4. *The analysis and revision mechanisms*

The forms of analysis in SCTL-MUS include the synthesis of MUS models from SCTL requirements, the model-checking of desirable SCTL properties, and the search for materializations of SLS scenarios in MUS models. In response to any conflicts between the

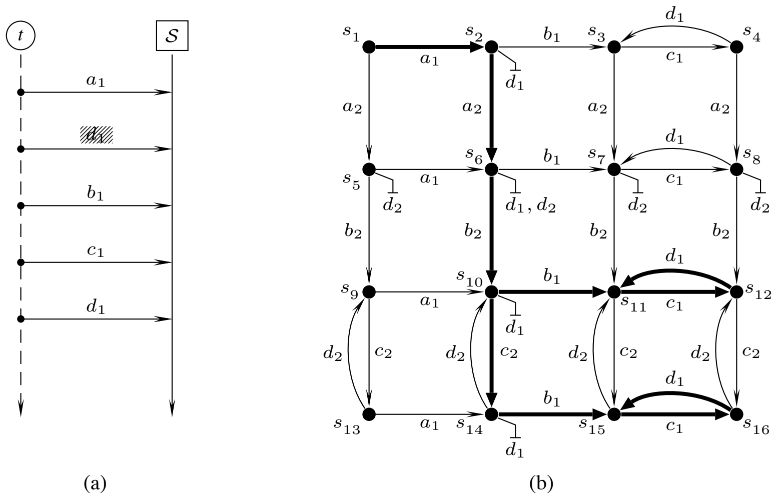


Figure 7. A sample SLS scenario and its materializations over a MUS model.

requirements, any violations of the properties or any evidence that a model cannot contain materializations of a scenario, we look at the computations performed to systematically generate pieces of diagnostic information (henceforth, Δs), pointing out the undesired findings in the models that were being synthesized, verified or validated (see [19] for further details). The notation

$$\Delta_{i,s_l} = s_j[a]_{\mathcal{M}} \rightsquigarrow \delta$$

represents the i-th piece of diagnostic information obtained over state s_l of a model, by which the enabledness of event a in state s_j (defined over the set $\{$*false*$, \perp,$ *true*$\}$) should be changed to a different value δ.

Having the diagnostic information, the revision mechanisms can identify the requirements that bore the behavior implemented in the problematic states. Thus, depending on the δ values and the enabledness of the events as specified by those requirements, three types of requirement revisions can be generated: *refinements* (from $\perp$ to *false* or *true*), *abstractions* (from *false* or *true* to $\perp$) and *retrenchments* (from *false* to *true*, or vice versa). Those revisions are evaluated one after the other (starting with those that modify fewer requirements) until finding one that solves (at least, partially) some of the diagnosed problems and gains the stakeholders' acceptance.

On the other hand, it is also straightforward to identify the scenarios whose materializations pass through the states indicated by the diagnostic information. Thus, it is easy to generate revised scenarios, introducing events that differentiate the materializations which are lost with the changes in the models (*counterexamples*) from those which are preserved (*witnesses*).

4. New Mechanisms for Continuous Integration

When enhancing the SCTL-MUS methodology (Section 3) to implement our approach to CI (Section 2), we focused on resolving the effective loss of modularization in cases of parallel composition, because this is the most important construct in the field of distributed reactive systems (see [2, 7, 22, 27]). Thus, the solutions presented in this section start up from the specification of a system $\mathcal{S}$ as the parallel composition of n components, $\mathcal{C}_1, \ldots, \mathcal{C}_n$:

$$\mathcal{S} = \mathcal{C}_1 \ldots |[\lambda]|_{\mathcal{M}} \ldots \mathcal{C}_n \tag{1}$$

$|[\lambda]|_{\mathcal{M}}$ is a variation of the classical *selective parallel composition* operator of process algebras [10], adapted to the many-valued semantics of MUS. In $\mathcal{C}_i|[\lambda]|_{\mathcal{M}}\mathcal{C}_j$, the enabledness of any event a in the (model of the) composition is computed from its enabledness in (the models of) $\mathcal{C}_i$ and $\mathcal{C}_j$, according to the truth tables below. Considering only *false* and *true* values, these tables coincide with the classical ones; the value $\perp$ simply acts as a neutral element, because it represents absence of knowledge.

If $a \notin \lambda$

$\mathcal{C}_i \diagdown \mathcal{C}_j$	*false*	$\perp$	*true*
false	*false*	*false*	*true*
$\perp$	*false*	$\perp$	*true*
true	*true*	*true*	*true*

If $a \in \lambda$

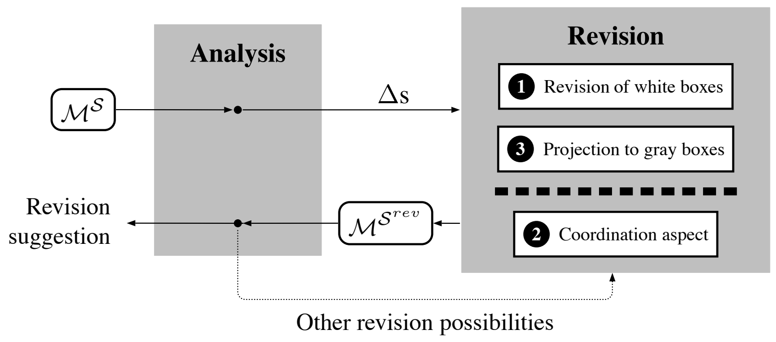
$\mathcal{C}_i \diagdown \mathcal{C}_j$	*false*	$\perp$	*true*
false	*false*	*false*	*false*
$\perp$	*false*	$\perp$	*true*
true	*false*	*true*	*true*

Starting from Eq. (1), as noticed in [16, 30], the goal is to remove unwanted emergent behavior observed in the model of $\mathcal{S}$. Parallel composition, in principle, interleaves the events of the different components in an arbitrary way, but most of the times only a reduced set of the resulting event sequences represent desired functionality. Accordingly, in order to remove behavior, the integration analyses will always require that certain events do not occur in certain states of the model in question, so we will be handling Δs of the form

$$\Delta_{i,s_l} = s_j[a]_{\mathcal{M}} \rightsquigarrow \textit{false};$$

that is, Δs pointing out changes of possible or unspecified transitions in the model of $\mathcal{S}$ into non-possible ones.

Driven by the Δs, as shown in Fig. 8, the revision can follow one of three procedures to reach the model of a revised system $\mathcal{S}^{rev}$. That model is subject to the same analysis as the original model of $\mathcal{S}$, to check whether $\mathcal{S}^{rev}$ must be presented to the stakeholders as a revision suggestion or, on the contrary, it is necessary to evaluate other possibilities.



Analysis
Revision
$\mathcal{M}^{\mathcal{S}}$
Δs
1 Revision of white boxes
3 Projection to gray boxes
2 Coordination aspect
Revision suggestion
$\mathcal{M}^{\mathcal{S}^{rev}}$
Other revision possibilities

Figure 8. General scheme of the support provided for CI.

The three procedures of Fig. 8 are instances of the generic ones explained in Section 2; we detail them in the following subsections. As a common feature, just like we did in [19], the first alternatives considered are always the simplest ones, and the stakeholders' feedback is exploited to learn about unwanted evolutions of the requirements, about validated traces that should be preserved in the models, etc.

4.1. *The first discharging procedure: Revisions that preserve white boxes*

The diagnostic information gathered over the model of the composition $\mathcal{S}$ can be used to revise one or several of the $\mathcal{C}_i$ components that are white boxes, using the revision mechanisms presented in [19] for individual components. For example, if we only modify component $\mathcal{C}_1$, the system of Eq. (1) is reexpressed as that of Eq. (2), graphically represented in Fig. 9:

$$\mathcal{S}^{rev} = \mathcal{C}_1^{rev} \; |[\lambda']|_{\mathcal{M}} \; \mathcal{C}_2 \ldots |[\lambda']|_{\mathcal{M}} \ldots \mathcal{C}_n \tag{2}$$

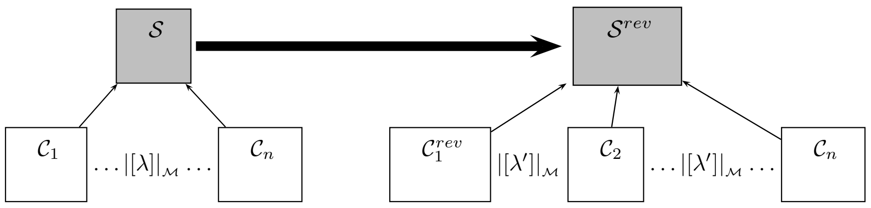
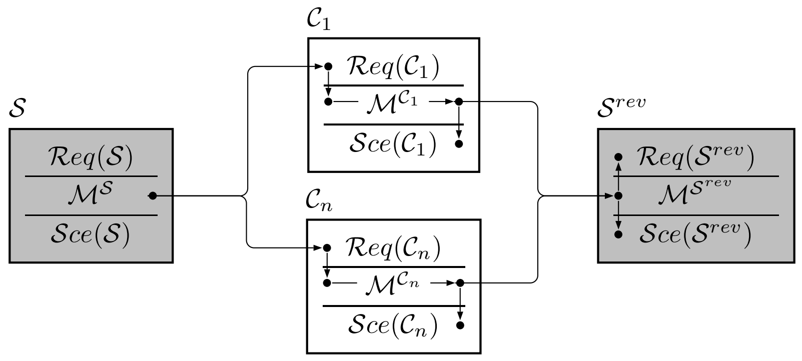


Figure 9. The first discharging procedure, in the case of Eq. (2).

The set of events λ in the composition operator becomes $\lambda' \supseteq \lambda$ because, as explained apropos Fig. 5, discharging changes made at the composition level over the original components leads to greater coupling between them —recall that, if $\lambda = \emptyset$, the effect of parallel composition is that of pure interleaving (i.e. complete decoupling).

The key to revise white boxes lies within the fact that every state of the MUS model of $\mathcal{S}$ refers directly to one state of each one of the composed models. By virtue of that, we define the revision procedure represented in Fig. 10:



$\mathcal{S}$
$\mathcal{R}eq(\mathcal{S})$
$\mathcal{M}^{\mathcal{S}}$
$\mathcal{S}ce(\mathcal{S})$
$\mathcal{C}_1$
$\mathcal{R}eq(\mathcal{C}_1)$
$\mathcal{M}^{\mathcal{C}_1}$
$\mathcal{S}ce(\mathcal{C}_1)$
$\mathcal{C}_n$
$\mathcal{R}eq(\mathcal{C}_n)$
$\mathcal{M}^{\mathcal{C}_n}$
$\mathcal{S}ce(\mathcal{C}_n)$
$\mathcal{S}^{rev}$
$\mathcal{R}eq(\mathcal{S}^{rev})$
$\mathcal{M}^{\mathcal{S}^{rev}}$
$\mathcal{S}ce(\mathcal{S}^{rev})$

Figure 10. The revision procedure that preserves white boxes.

- The changes pointed out over the model of $\mathcal{S}$ ($s_j[a]_{\mathcal{M}^{\mathcal{S}}} \rightsquigarrow$ *false*) are translated into changes over the models of one or several $\mathcal{C}_i$ ($s_k[a]_{\mathcal{M}^{\mathcal{C}_i}} \rightsquigarrow$ *false*).
- Next, we generate revisions of the requirements of the $\mathcal{C}_i$ components, applying the mechanisms introduced in [19].
- Having revised the requirements of $\mathcal{C}_i$, we update the model that were being considered as part of $\mathcal{S}$; next, we update the scenarios of $\mathcal{C}_i$.
- Finally, the model of the revised system $\mathcal{S}^{rev}$ is obtained from the revised models of the $\mathcal{C}_i$ components; thereupon, we update the requirements and the scenarios of $\mathcal{S}$.

With this procedure, the stakeholders can support their decision to accept or reject the revision suggestions by the evolutions of the requirements and the scenarios. Obviously, the simplest revisions —the ones which are suggested first— are those which modify fewer requirements in fewer components.

It is worth noting that, for simplicity, we take the requirements of $\mathcal{S}$ as the union of the requirements of the $\mathcal{C}_i$ components:

$$\mathcal{R}eq(\mathcal{S}) = \bigcup_{\forall i} \mathcal{R}eq(\mathcal{C}_i)$$

Being a gray box, the models of $\mathcal{S}$ are not obtained through synthesis from its requirements, but through composition of the models of its components. In consequence, the role of the requirements is to annotate as much information as possible about the stakeholders' statements that yielded those models.

4.2. *The second discharging procedure: Introduction of a coordinator aspect*

The procedure of the preceding section is only applicable when some of the components of $\mathcal{S} = \mathcal{C}_1 \ldots |[\lambda]|_{\mathcal{M}} \ldots \mathcal{C}_n$ is a white box. One first alternative, which can be applied in all cases, is to preserve the original components untouched, and introduce an aspect that modifies their conjoint behavior according to the results of the integration analyses. Thus, the system of Eq. (1) is reexpressed as follows, with $\lambda' \supseteq \lambda$:

$$\mathcal{S}^{rev} = \big(\mathcal{C}_1 \ldots |[\lambda]|_{\mathcal{M}} \ldots \mathcal{C}_n\big)\ |[\lambda']|_{\mathcal{M}}\ \textit{Coordinator} \tag{3}$$

The transformation from Eq. (1) into Eq. (3) is represented graphically in Fig. 11:

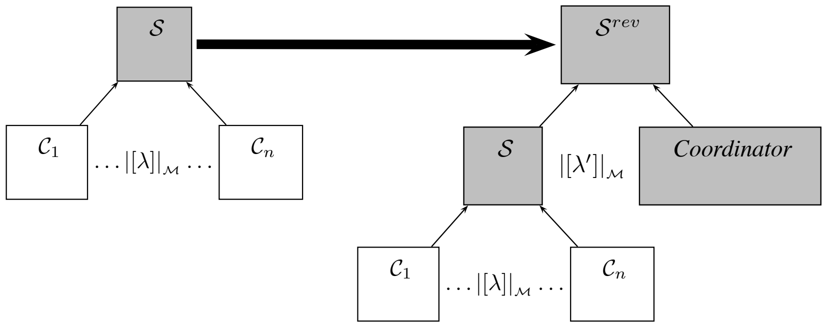


Figure 11. The second discharging procedure, in the case of Eq. (3).

The *Coordinator* aspect is obtained by transforming the analyzed model of $\mathcal{S}$ according to the pseudocode of Algorithm 1. The input to this algorithm (called the *reduction algorithm*) is the model of $\mathcal{S}$ itself, plus one or several Δs gathered during its analysis. The output is a MUS model for the aspect, whose characterization is completed with whichever SCTL expressions considered in the integration analyses.

The algorithm proceeds in four fundamental steps, which can be explained with the sample run of Fig. 12:

- First, we turn into unspecified all the non-possible events in the input model, to distinguish them from the events that will become non-possible due to the integration changes. If the input model is that of Fig. 12(a), we obtain the one of Fig. 12(b).
- Next, we enforce the Δs over the model, turning into non-possible events that were either possible or unspecified. In the example, we assume only one Δ that fixes c as a non-possible event in state s_3; the resulting model is shown in Fig. 12(c).
- The third step consists of an iterative procedure to reduce the model resulting from the previous step, merging *contiguous compatible states*, that is, states which are joined by possible transitions and which do not contain contradictory values of enabledness for any event (i.e. what is possible in one state cannot be non-possible in the other). In the model of Fig. 12(c), for instance, it is possible to merge states s_1 and s_2, since they agree on the fact that c can occur, and do not contradict each other regarding a (possible in s_1 and unspecified in s_2) or b (vice versa).[b]

[b]In the reduced models, the initial state is the one that subsumes the original s_1.

Algorithm 1 Reduction of MUS models

```
Input: One model $\mathcal{M}^{\mathcal{S}}$ and a set of $\Delta$s with changes of the form $s_k[a]_{\mathcal{M}^{\mathcal{S}}} \rightsquigarrow \mathit{false}$.
Output: The model of a coordinator aspect, $\mathcal{M}^{Coordinator}$.
 1: $\mathcal{M}^{temp} = \mathcal{M}^{\mathcal{S}} = (S_{\mathcal{M}^{temp}}, \Lambda, \rightarrow)$.
 2: Change the non-possible transitions $s_i \xrightarrow{\{b,\mathit{false}\}} s_{unsp} \in \rightarrow$ in $\mathcal{M}^{temp}$ for unspecified transitions $s_i \xrightarrow{\{b,\perp\}} s_{unsp}$.
 3: Enforce the changes indicated by the $\Delta$s over $\mathcal{M}^{temp}$, turning the corresponding possible transitions $s_i \xrightarrow{\{b,\mathit{true}\}} s_j$ or unspecified transitions $s_i \xrightarrow{\{b,\perp\}} s_{unsp}$ into non-possible transitions, $s_i \xrightarrow{\{b,\mathit{false}\}} s_{unsp}$.
 4: Label all the possible transitions in $\mathcal{M}^{temp}$ as non-tested.
 5: Select a non-tested transition, $\Upsilon \equiv s_i \xrightarrow{\{b,\mathit{true}\}} s_j$.
 6:    Merge the states $s_i$ and $s_j$ in $\mathcal{M}^{temp}$.
 7:       If there exists $b \in \Lambda$ such that $s_i[b]_{\mathcal{M}^{temp}} = \mathit{true}$ and $s_j[b]_{\mathcal{M}^{temp}} = \mathit{false}$ (or vice versa),
          then discard the possibility to merge $s_i$ and $s_j$, and label $\Upsilon$ as tested.
 8:       While there exists $b \in \Lambda$ such that $s_i[b]_{\mathcal{M}^{temp}} = \mathit{true}$ and $s_j[b]_{\mathcal{M}^{temp}} = \mathit{true}$, and $Succ(s_i, b) \neq Succ(s_j, b)$,
 9:          Go to line 6 and try to merge $Succ(s_i, b)$ and $Succ(s_i, b)$. If that merging is discarded,
             then discard also the merging of $s_i$ and $s_j$, and label $\Upsilon$ as tested.
10:       Modify the origin of all the non-possible transitions $s_j \xrightarrow{\{b,\mathit{false}\}} s_{unsp}$ such that $s_i[b]_{\mathcal{M}^{temp}} \neq \mathit{true}$, turning them into $s_i \xrightarrow{\{b,\mathit{false}\}} s_{unsp}$.
11:       Modify the origin of all the possible transitions $s_j \xrightarrow{\{b,\mathit{true}\}} s_l$ such that $s_i[b]_{\mathcal{M}^{temp}} = \perp$, turning them into $s_i \xrightarrow{\{b,\mathit{true}\}} s_l$.
12:       Remove state $s_j$ from $\mathcal{M}^{temp}$, changing all the possible transitions into $s_j$ for transitions into $s_i$.
13: If there exist non-tested transitions, then start a new iteration from line 5.
14: Otherwise, simplify $\mathcal{M}^{Coordinator} = \mathcal{M}^{temp}$, and finish.
```

The merging of two states turns the possible transitions that joined them into unit loops. It can also imply merging other states, particularly those which are successors of the merged ones through the same events. Thus, as shown in Fig. 12(d), the merging of s_1 and s_2 implies merging s_4 and s_5 as well, because these are their respective c-successors. Finally, although not shown in the example, the state that results from the merging becomes the destination of all the possible transitions which led into either one of the merged states.

- Once it is not possible to go on merging states, the model obtained is simplified by turning unit loops into unspecified transitions, because the events linked to those loops are not relevant for the integration analysis in question. In the example, once we have got to Fig. 12(e), it is not possible to merge more states because $s_{1,2,4,5}$ and s_3 contain contradictory values of enabledness for event c. The result can be simplified by removing the unit loops, which bears the model of Fig. 12(f). Looking at the original model and the Δ considered, it is easy to see that event a was indeed irrelevant for the

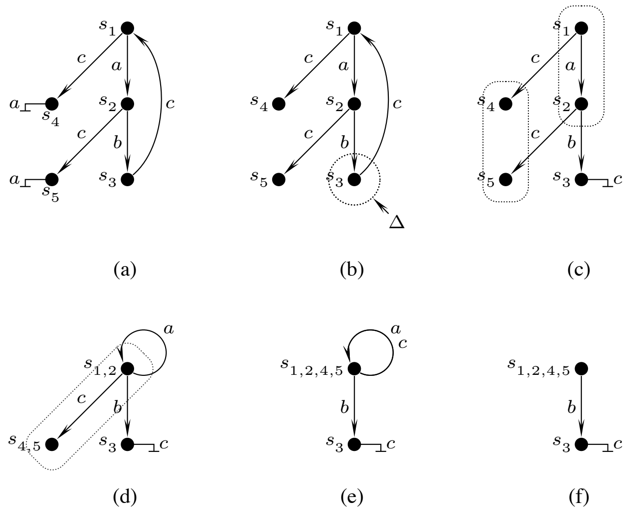


Figure 12. Sample run of the reduction algorithm.

integration analysis: whereas event c was possible in all states, it becomes non-possible only after event b has occurred.

When the reduction algorithm has finished computing a model for the aspect, the model of the revised system $\mathcal{S}^{rev}$ is computed according to Eq. (3), to check whether it solves problems detected over $\mathcal{S}$. If the analysis results are not better, we can try with other possible ways to reduce the model after applying the same Δs (note the alternatives opened by line 5 of Algorithm 1), or consider other Δs and their combinations. On the contrary, if the model of $\mathcal{S}^{rev}$ returns better analysis results, the revision procedure finishes by modifying the scenarios of $\mathcal{S}$ to reflect the changes made to the system model. The stakeholders' reasoning about the revision suggestions is mainly supported by the evolutions of those scenarios, because the requirements and the scenarios of the $\mathcal{C}_i$ components remain unchanged.

4.3. *The third discharging procedure: Projection of the coordinator aspect over the components*

Having obtained a coordinator aspect through the procedure of Section 4.2, it is always possible to revise the system $\mathcal{S} = \mathcal{C}_1 \ldots |[\lambda]|_{\mathcal{M}} \ldots \mathcal{C}_n$ by projecting that aspect onto some or all of the $\mathcal{C}_i$ components, thus preserving the original modularization. For example, when projecting the aspect only onto the first component, the system of Eq. (3) is reexpressed as follows:

$$\mathcal{S}^{rev} = \mathcal{C}_1^{rev} \; |[\lambda']|_{\mathcal{M}} \; \mathcal{C}_2 \ldots |[\lambda']|_{\mathcal{M}} \ldots \mathcal{C}_n, \tag{4}$$

where

$$\mathcal{C}_1^{rev} = \mathcal{C}_1 \; |[\lambda'']|_{\mathcal{M}} \; \textit{Coordinator}.$$

The transformation from Eq. (1) into Eq. (4) is represented graphically in Fig. 13:

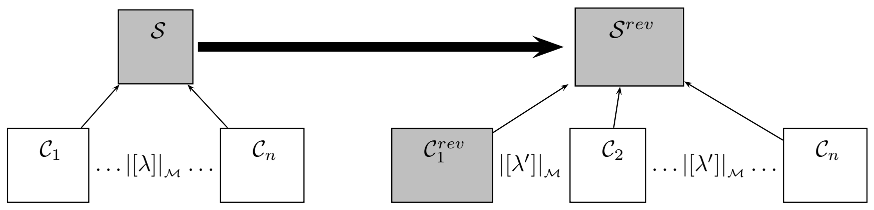


Figure 13. The third discharging procedure, in the case of Eq. (4).

This revision procedure always turns the modified components into gray boxes, because part of their functionality arises from the combination with the aspect, and not only from their requirements through a synthesis process. Nonetheless, for each one of the revised components, we evaluate the original requirements to update the specifications according to the following (not mutually exclusive) rules:

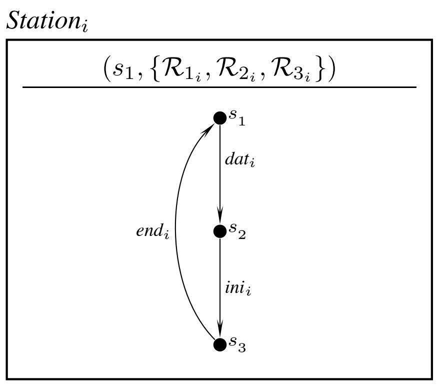
- If a requirement $\mathcal{R}$ is satisfied in at least one state of the revised model, then it remains unaltered in the revised set of requirements.
- If a requirement $\mathcal{R}$ is violated in at least one state of the revised model, then the revised set of requirements includes a retrenched version $\mathcal{R}^*$ that negates the consequent of $\mathcal{R}$.

Just like in Section 4.2, the sets of requirements of the new gray boxes include any SCTL expressions considered in the integration analysis. Besides, we revise both the scenarios of $\mathcal{S}$ and those of the modified components. Again, the scenarios play an important role to support the stakeholders' reasoning about the revision suggestions. It is particularly worth noting that, in cases where the two rules above are applicable, the revised scenarios help identifying the particular situations where the different versions of the same original requirement hold.

To finish, note that the simplest revisions according to this procedure —the ones which are suggested first— are obviously those which modify fewer components.

5. An Application Example

In this section, we illustrate the mechanisms presented in Section 4 to support continuous integration in SCTL-MUS, applied to the specification of a network with n stations that transmit data over a shared channel. With no loss of generality, we shall describe the case $n = 2$ to avoid the complex graphical representation of the models resulting from the composition of more stations. The example shows the different revision possibilities that can be considered after detecting that the composition of the two stations (originally specified as white-box components) violates two SCTL properties.



$Station_i$
$(s_1, \{\mathcal{R}_{1_i}, \mathcal{R}_{2_i}, \mathcal{R}_{3_i}\})$
s_1
dat_i
end_i
s_2
ini_i
s_3

Figure 14. Initial specification of the stations.

5.1. *Starting point*

To begin with, we assume that the two stations of the example are originally identical, though we shall not treat them as instances of a same class anymore —that is, we will allow modifying the two stations in disparate ways. Their specification involves the following events, with the corresponding intended meaning:

- dat_i: $Station_i$ receives new data to transmit over the channel.
- ini_i: $Station_i$ initiates a transmission.
- end_i: $Station_i$ finishes a transmission.

The starting point is a set of requirements by which a station simply receives data and send them, regardless of the fact that it must use a shared channel:

- *"$Station_i$ can initiate a transmission when it has data to send":*
  $\boldsymbol{\mathcal{R}_{1_i}} \equiv dat_i \Rightarrow \bigcirc ini_i$
- *"If $Station_i$ initiates a transmission, then it will be able to finish it":*
  $\boldsymbol{\mathcal{R}_{2_i}} \equiv ini_i \Rightarrow \bigcirc\, end_i$
- *"$Station_i$ can receive new data to send after it has finished other transmission":*
  $\boldsymbol{\mathcal{R}_{3_i}} \equiv end_i \Rightarrow \bigcirc\, dat_i$

Besides, the specification includes the fact that, upon startup, the stations can receive data to transmit:

- $s_1[dat_i] = true$

Figure 14 shows the white box that initially represents the stations, including the MUS model that is synthesized from the aforementioned requirements. Noting that there is no overlapping between the alphabets of the two stations, the global system *Network* is obtained by composing them using the $|||_{\mathcal{M}}$ operator (that is, $|[\lambda]|_{\mathcal{M}}$ with $\lambda = \emptyset$):

$$Network = Station_1 \; |||_{\mathcal{M}} \; Station_2 \tag{5}$$

In addition to the requirements, we assume that the stakeholders consider the scenario $\mathcal{E}$ of Fig. 15, which they interpret as "*the fact that the Station*$_2$ *receives data to send before Station*$_1$ *does not imply that it will transmit them first*".

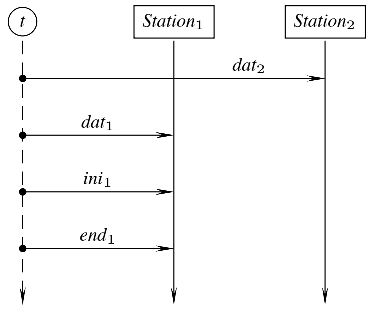


Figure 15. $\mathcal{E}$, a scenario for the composition of the two original stations.

The integration analysis in this example consists of verifying two properties to ensure mutual exclusion in the access to the shared channel, specifying that "*one station cannot initiate a transmision right after the other has started transmitting*":

$$\begin{aligned} \mathcal{P}_1 &\equiv ini_1 \Rightarrow \bigcirc \neg ini_2 \\ \mathcal{P}_2 &\equiv ini_2 \Rightarrow \bigcirc \neg ini_1 \end{aligned}$$

The verification is done over the model $\mathcal{M}^{Network}$ of Fig. 16, that results from composing the models of the stations according to Eq. (5).[c] Obviously, both properties are violated, because the free interleaving of the *Station*$_i$ components allow situations that the two stations are transmitting at the same time.

The *false* satisfaction values in state s_5 of $\mathcal{M}^{Network}$ take the model-checking algorithm to generate the following pieces of diagnostic information:

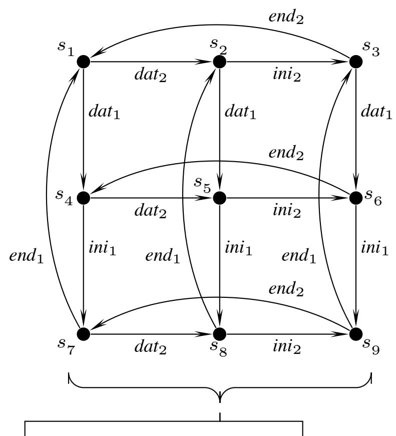
$$\begin{aligned} \Delta_{1,s_5} &= s_6[ini_1]_{\mathcal{M}^{Network}} \rightsquigarrow false \\ \Delta_{2,s_5} &= s_8[ini_2]_{\mathcal{M}^{Network}} \rightsquigarrow false \end{aligned} \qquad (6)$$

From these Δs, the revision mechanisms start looking for solutions to the verification problems, considering the alternatives described in the following subsections.

5.2. *Revisions that preserve white boxes*

The states of $\mathcal{M}^{Network}$ correspond directly to states of the models of *Station*$_i$. Figure 17 shows the requirement revisions that stem from Δ_{1,s_5} following the procedure introduced in [19]: a retrenchment of $\mathcal{R}_{1_1}$ and a refinement of $\mathcal{R}_{2_2}$.

[c]The rows and columns of the tables in Fig. 16 —and in the subsequent ones— correspond to the different states of the graphical representation of the model $\mathcal{M}^{Network}$.



$[(\boxed{}, s) \models^6 \mathcal{P}_1]$		
$\perp$	$\perp$	$\perp$
$\perp$	*false*	$\perp$
$\perp$	$\perp$	$\perp$

$[(\boxed{}, s) \models^6 \mathcal{P}_2]$		
$\perp$	$\perp$	$\perp$
$\perp$	*false*	$\perp$
$\perp$	$\perp$	$\perp$

Figure 16. The original model of *Network*, with the results of verifying $\mathcal{P}_1$ and $\mathcal{P}_2$.

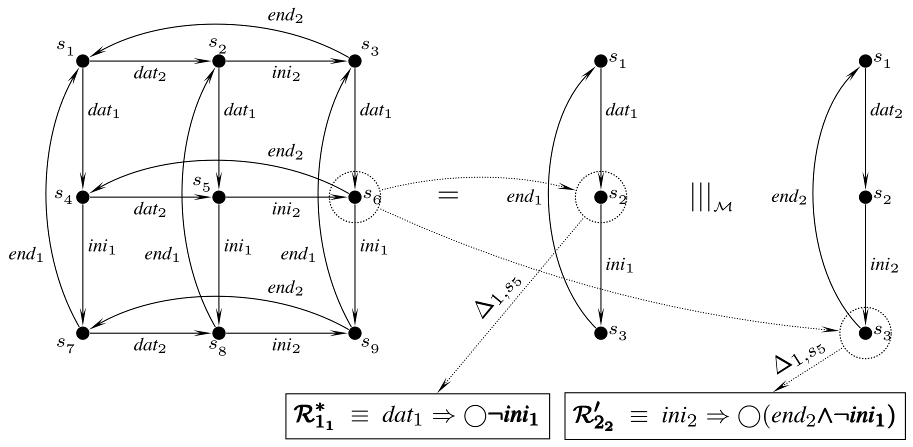
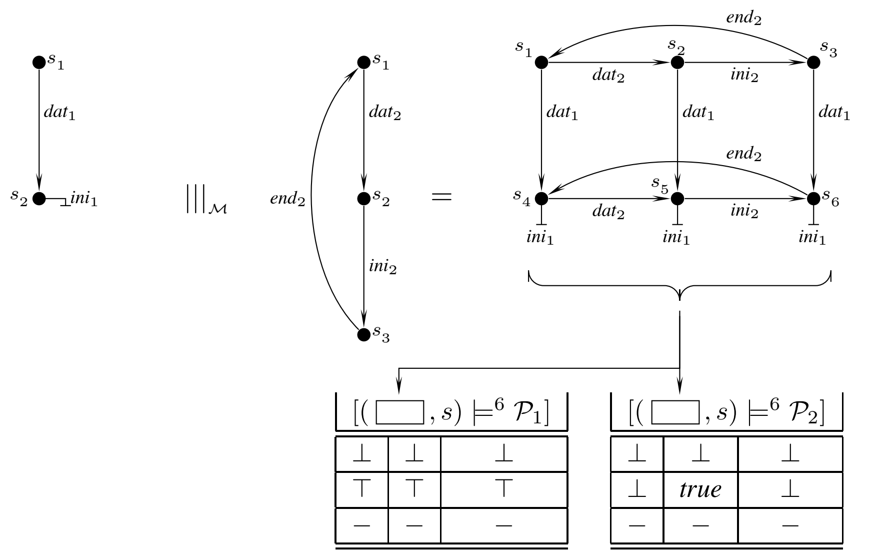

end_2
s_1
s_2
s_3
dat_2
ini_2
dat_1
s_4
s_5
s_6
ini_1
end_1
s_7
s_8
s_9
$=$
$|||_{\mathcal{M}}$
Δ_{1,s_5}
$\mathcal{R}^*_{1_1} \equiv dat_1 \Rightarrow \bigcirc\neg \boldsymbol{ini_1}$
$\mathcal{R}'_{2_2} \equiv ini_2 \Rightarrow \bigcirc(end_2 \wedge \neg \boldsymbol{ini_1})$

Figure 17. Revisions of the requirements of $Station_i$ from Δ_{1,s_5}.

Assuming that we consider first the retrenchment of $\mathcal{R}_{1_1}$, the revision proceeds by updating the model of the first station to implement the requirements $\{\mathcal{R}^*_{1_1}, \mathcal{R}_{2_1}, \mathcal{R}_{3_1}\}$. Next, we update the model of the network, to check that the changes lead to better verification results for $\mathcal{P}_1$ and $\mathcal{P}_2$. Figure 18 shows that, indeed, the *false* satisfaction values disappear, and so the revision is presented to the stakeholders.[d]



$[(\boxed{}, s) \models^6 \mathcal{P}_1]$

$\perp$	$\perp$	$\perp$
$\top$	$\top$	$\top$
$-$	$-$	$-$

$[(\boxed{}, s) \models^6 \mathcal{P}_2]$

$\perp$	$\perp$	$\perp$
$\perp$	*true*	$\perp$
$-$	$-$	$-$

Figure 18. Checking the first revision derived from Δ_{1,s_5}.

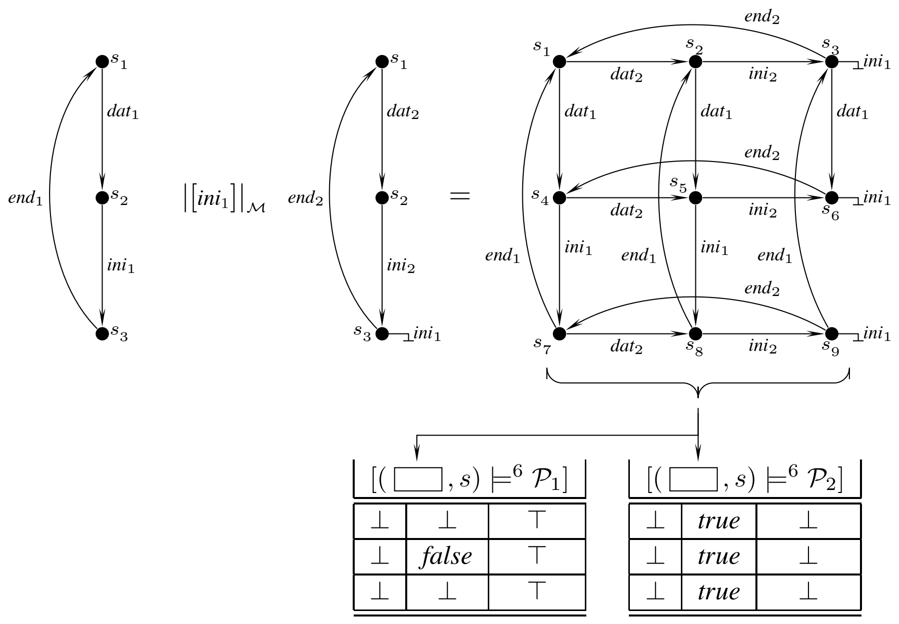
In this case, the stakeholders immediately reject the suggestion, noticing that the requirement $\mathcal{R}^*_{1_1}$ prevents the first station from ever transmitting any data. So, the next revision alternative evaluated is the refinement of $\mathcal{R}_{2_2}$ into $\mathcal{R}'_{2_2}$. As shown in Fig. 19, this refinement modifies state s_3 of the model of the second station, which in turn affects several states of the model of the network. This time, the changes lead to a partial solution of the verification problems: the *false* value for $\mathcal{P}_1$ persists in s_5, but the *false* value for $\mathcal{P}_2$ has been turned into *true*.

Therefore, the stakeholders are asked about changing the original specification of the second station for the white box *Station*$^1_2$ of Fig. 20, reexpressing the system of Eq. (5) as follows:

$$\mathit{Network} = \mathit{Station}_1 \; |[\mathit{ini}_1]|_{\mathcal{M}} \; \mathit{Station}^1_2 \tag{7}$$

The revision suggestion also includes revisions of the scenario $\mathcal{E}$. The first scenario of Fig. 21 —a counterexample of $\mathcal{E}$— is obtained from the materializations which are lost over the traces $(s_1, s_2, s_5, s_6, s_9, s_3)$ and $(s_1, s_2, s_3, s_6, s_9, s_3)$, evidencing that the first station cannot send data over the channel while the other station is transmitting. The two other revised scenarios —witnesses of $\mathcal{E}$— stem from the materializations which persist in the model of the network, characterized by the fact that event *ini*$_2$ does not occur before *ini*$_1$.

[d]The $\top$ values in Fig. 18 indicate that there is no point in checking $\mathcal{P}_1$ in some states, because its premise is not satisfied there. As explained in [19], this is obviously not an unwanted verification result.



$[(\boxed{}, s) \models^6 \mathcal{P}_1]$		
$\perp$	$\perp$	$\top$
$\perp$	*false*	$\top$
$\perp$	$\perp$	$\top$

$[(\boxed{}, s) \models^6 \mathcal{P}_2]$		
$\perp$	*true*	$\perp$
$\perp$	*true*	$\perp$
$\perp$	*true*	$\perp$

Figure 19. Checking the second revision derived from Δ_{1,s_5}.

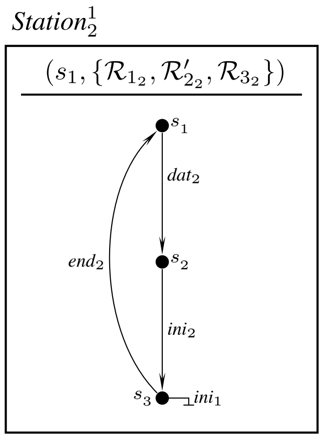

$Station_2^1$
$(s_1, \{\mathcal{R}_{1_2}, \mathcal{R}'_{2_2}, \mathcal{R}_{3_2}\})$
s_1
dat_2
s_2
end_2
ini_2
s_3
$\perp ini_1$

Figure 20. A suggestion to revise the second station as a white box.

Interpreting the evolutions of the requirement $\mathcal{R}'_{2_2}$ and the scenario $\mathcal{E}$, the stakeholders would accept the revision suggestion, because everything is clearly in line with the goal of ensuring the use of the communications channel in mutual exclusion. Particularly, reading $\mathcal{R}'_{2_2}$ one understands that "*if the second station initiates a transmission, then it impedes transmissions of the first one*".

Now, using Δ_{2,s_5}, we would get to a complete solution to the verification problems

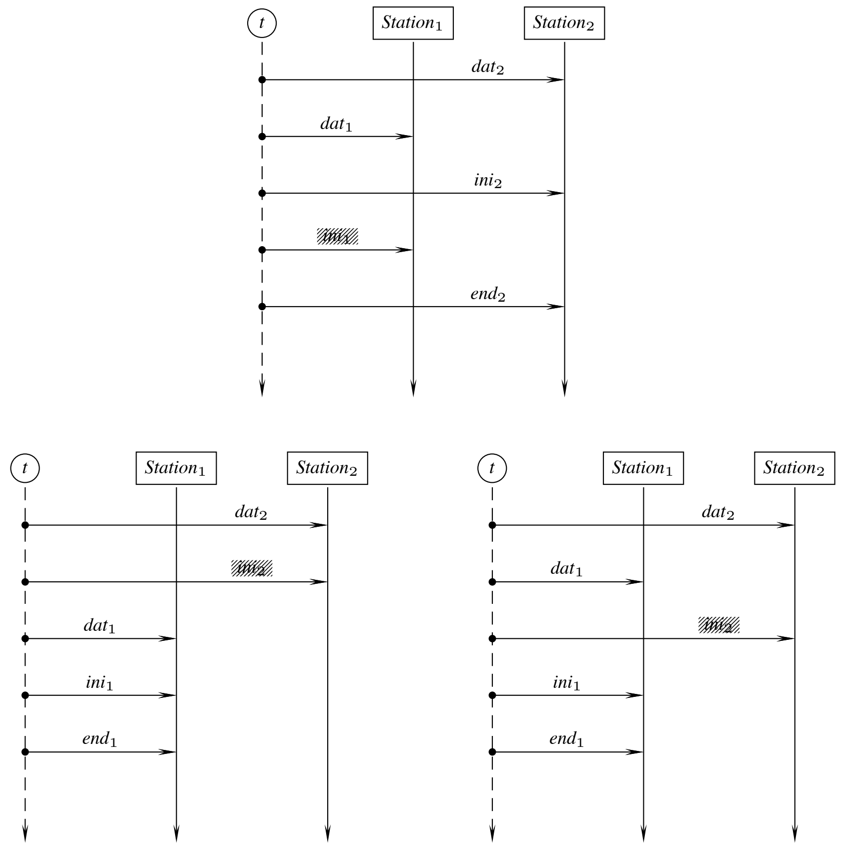


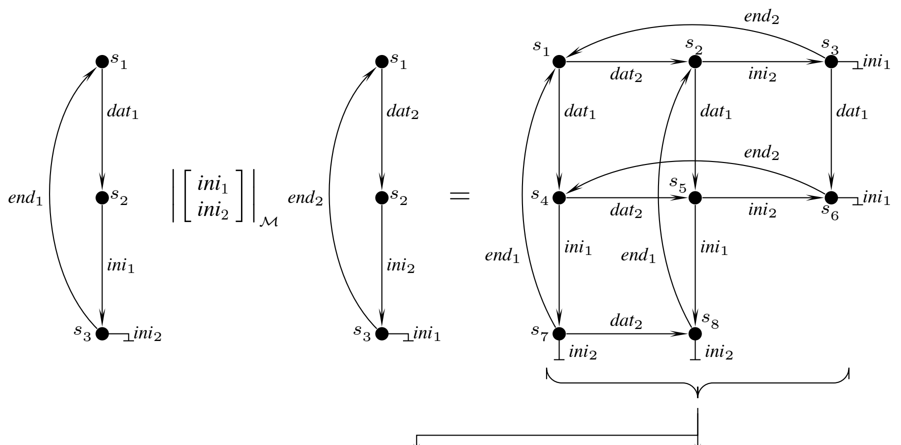
Figure 21. Revisions of scenario $\mathcal{E}$ from the transformation of the model of Fig. 16 into that of Fig. 19.

in a completely analogous way, refining $\mathcal{R}_{2_1}$ into $\boldsymbol{\mathcal{R}'_{2_1}} \equiv ini_1 \Rightarrow \bigcirc(end_1 \wedge \neg \boldsymbol{ini_2})$. The update of the model of the network is shown in Fig. 22, where it is clear that $\mathcal{P}_1$ and $\mathcal{P}_2$ are no longer violated.

The revision suggestion presented to the stakeholders implies rewriting the system as follows, using the components represented in Fig. 23:

$$Network = Station_1^1 \;||[ini_1, ini_2]||_{\mathcal{M}}\; Station_2^2 \tag{8}$$

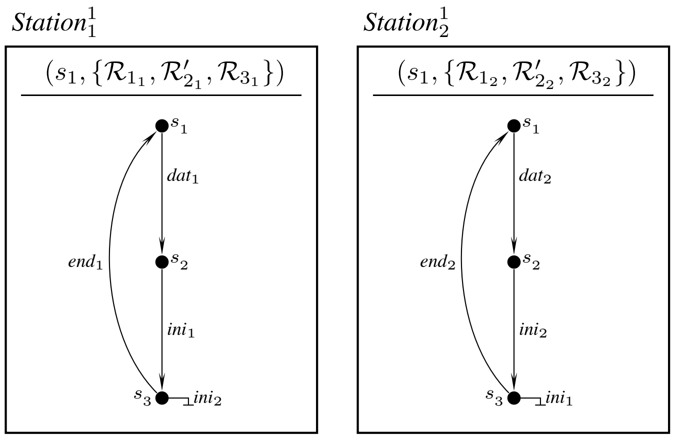
We also provide revisions of the scenario $\mathcal{E}$. In addition to the scenarios of Fig. 21, we provide that of Fig. 24 to notify that the new changes to the model of the network imply losing the materialization of $\mathcal{E}$ over the trace $(s_1, s_2, s_5, s_8, s_9, s_3)$. In sum, the revision suggestion is clearly acceptable, following the same reasoning that led to accepting the previous partial solution.



$[(\boxed{}, s) \models^6 \mathcal{P}_1]$		
$\perp$	$\perp$	$\top$
true	*true*	$\top$
$\perp$	$\perp$	$-$

$[(\boxed{}, s) \models^6 \mathcal{P}_2]$		
$\perp$	*true*	$\perp$
$\perp$	*true*	$\perp$
$\top$	$\top$	$-$

Figure 22. Checking the revision derived from combining Δ_{1,s_5} and Δ_{2,s_5}.



$Station_1^1$
$(s_1, \{\mathcal{R}_{1_1}, \mathcal{R}'_{2_1}, \mathcal{R}_{3_1}\})$
s_1
dat_1
s_2
ini_1
end_1
s_3
$\neg ini_2$
$Station_2^1$
$(s_1, \{\mathcal{R}_{1_2}, \mathcal{R}'_{2_2}, \mathcal{R}_{3_2}\})$
s_1
dat_2
s_2
ini_2
end_2
s_3
$\neg ini_1$

Figure 23. A suggestion to revise the two stations as white boxes.

5.3. *Introduction of a coordinator aspect*

Having explained the revisions that preserve the white-box character of the stations, we describe next the suggestions that would be obtained through the procedure of introducing a coordinator aspect, again from the analysis shown in Fig. 16 and the Δs of Eq. (6).

The first possibility explored is to apply the changed indicated by Δ_{1,s_5} over the original model of *Network* (shown in Fig. 16), and then execute the reduction algorithm. The

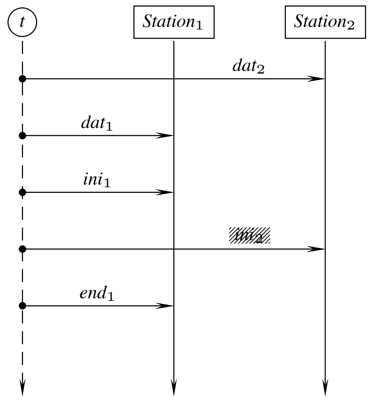


Figure 24. A new revision of scenario $\mathcal{E}$ from the transformation of the model of Fig. 16 into that of Fig. 22.

process is illustrated in Fig. 25. Note that the first step of turning non-possible transitions into unspecified ones is not necessary, because the input model does not contain non-possible transitions.

Once we have obtained the model of the coordinator, we check that, composed with the model of *Network*, it improves the verification results of properties $\mathcal{P}_1$ and $\mathcal{P}_2$. In this case, the result coincides with the model of Fig. 19 which, as previously stated, represents a partial solution to the verification problems. Therefore, the stakeholders are faced with a revision suggestion that consists of introducing the aspect *Coordinator*^{1} of Fig. 26, to reexpress the system of Eq. (5) as follows:

$$Network = \left(Station_1 \; |||_{\mathcal{M}} \; Station_2\right) \; |[ini_1, ini_2, end_2]|_{\mathcal{M}} \; Coordinator^1 \qquad (9)$$

The aspect being a gray box, the stakeholders' decision to accept or reject the revision suggestion can be supported mainly by the evolutions of scenario $\mathcal{E}$, previously shown in Fig. 21.

From the partial solution, we can find again a complete one by applying Δ_{2,s_5}. The resulting model of the aspect is added to the model extracted before, and the result is composed with the model $\mathcal{M}^{Network}$ of Fig. 16 by means of the operator $|[ini_1, ini_2, end_1, end_2]|_{\mathcal{M}}$. We obtain the model of Fig. 22, which, as we already know, raises no problems with properties $\mathcal{P}_1$ and $\mathcal{P}_2$. Thus, the stakeholders are faced with a revision suggestion that consists of introducing the aspect *Coordinator*^{2} of Fig. 27, and reexpressing the system of Eq. (5) as follows:

$$Network = \left(Station_1 \; |||_{\mathcal{M}} \; Station_2\right) \; |[ini_1, end_1, ini_2, end_2]|_{\mathcal{M}} \; Coordinator^2 \qquad (10)$$

The stakeholders would note that the revision suggestion is acceptable by looking at the evolutions of scenario $\mathcal{E}$, which are shown in Figs. 21 and 24. Indeed, it would be easy for them to recognize *Coordinator*^{2} as a mutual exclusion semaphore.

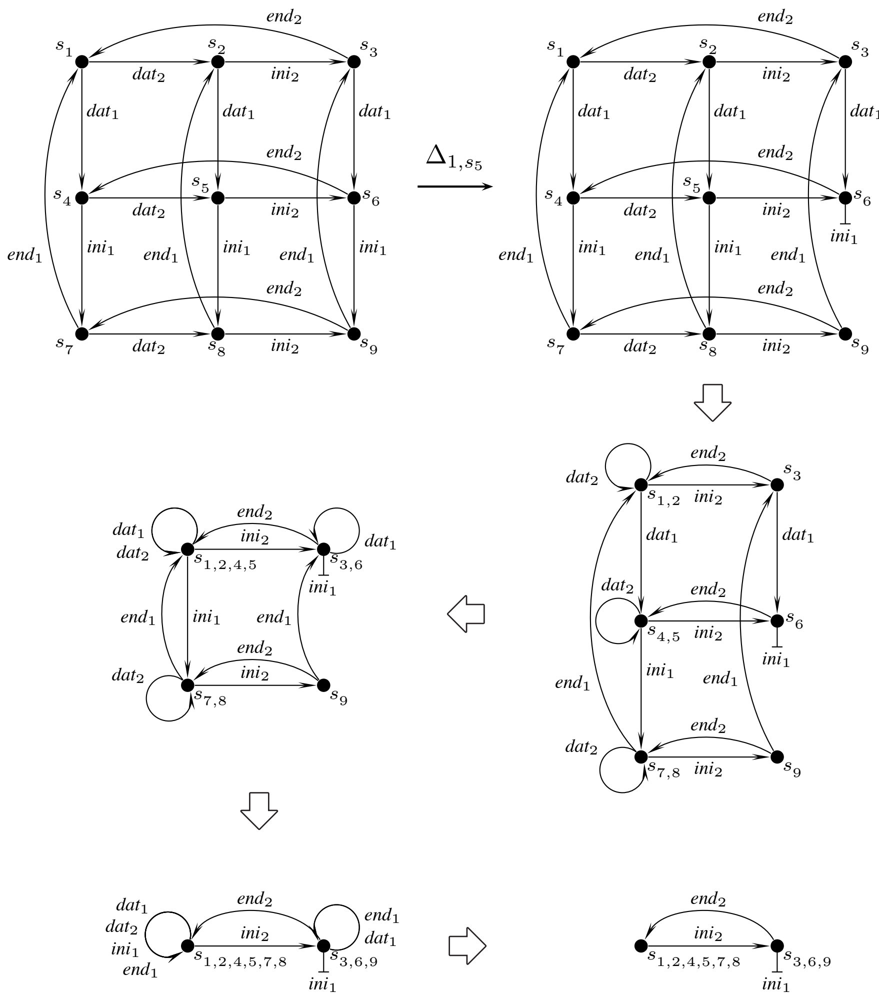

Δ_{1,s_5}
s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9
dat_1 dat_2 ini_1 ini_2 end_1 end_2
$s_{1,2}$ $s_{4,5}$ $s_{7,8}$ $s_{1,2,4,5}$ $s_{3,6}$ $s_{1,2,4,5,7,8}$ $s_{3,6,9}$

Figure 25. Running the reduction algorithm over the model of Fig. 16 and Δ_{1,s_5}.

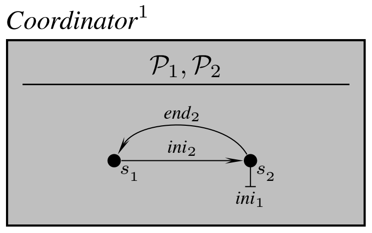


Figure 26. A coordinator aspect that leads to a partial solution.

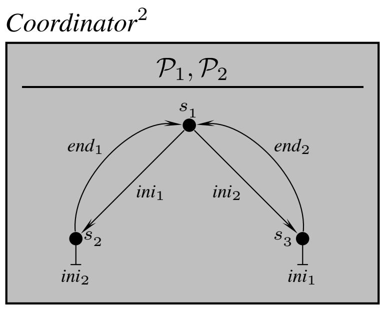


Figure 27. A coordinator aspect that leads to a complete solution.

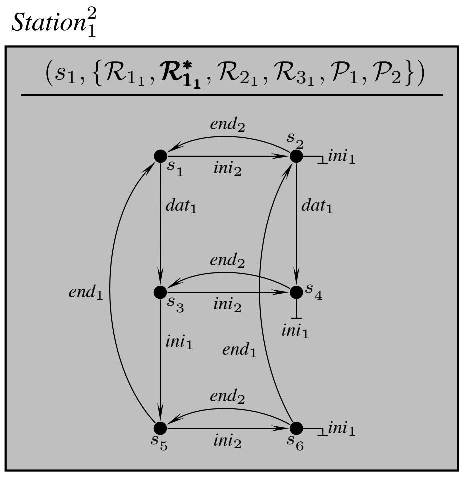


Figure 28. A suggestion to revise the first station as a gray box, leading to a partial solution.

5.4. *Projection of the coordinator aspect over the components*

The third way to discharge integration changes is to turn the stations into gray boxes, by projecting the coordinator aspect over them. In this case, we would obtain the same partial solution of Fig. 19 by projecting the aspect *Coordinator*^{1} of Fig. 26 over any of the two stations. For example, choosing the first station, the stakeholders would be asked to change its specification as a white box for the gray box $Station_1^2$ of Fig. 28, and to reexpress the system of Eq (5) as follows:

$$Network = Station_1^2 \;|[ini_1, ini_2, end_2]|_{\mathcal{M}}\; Station_2 \tag{11}$$

It is worth noting that the revised specification for the first station contains both the original requirement $\mathcal{R}_{1_1}$ and its retrenched version $\boldsymbol{\mathcal{R}^*_{1_1}} \equiv dat_1 \Rightarrow \bigcirc \neg \boldsymbol{ini_1}$. The reason is that $\mathcal{R}_{1_1}$ holds in state s_1 of the model of $Station_1^2$, whereas $\mathcal{R}^*_{1_1}$ holds in s_2. The revisions of scenario $\mathcal{E}$, which coincide with the ones represented in Fig. 21, allow the stakeholders to identify the particular situations in which any of the two versions holds, i.e.

the situations when the first station can initiate a transmission right after receiving data and the situations when it cannot.

One step further, the aspect *Coordinator*^{2} that results from enforcing Δ_{2,s_5} (Figure 27) can be projected over either $Station_1^2$ or $Station_2^1$ to reach the same complete solution of Fig. 22. Choosing again the first station, the stakeholders would be asked to change its specification for the gray box $Station_1^3$ of Fig. 28, and reexpressing the system of Eq. (5) as follows:

$$Network = Station_1^3 \;|[ini_1, end_1, ini_2, end_2]|_{\mathcal{M}}\; Station_2 \tag{12}$$

Once again, the revision suggestion would include the same evolutions of scenario $\mathcal{E}$ which are shown in Figs. 21 and 24.

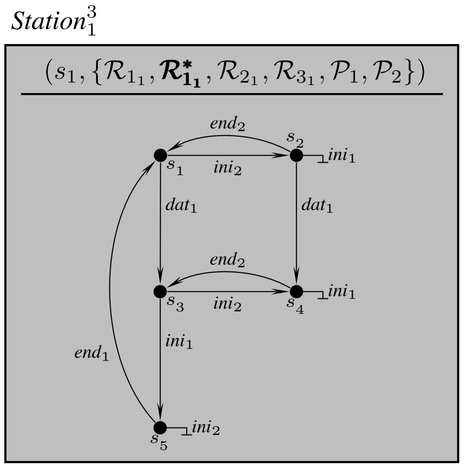

$Station_1^3$
$(s_1, \{\mathcal{R}_{1_1}, \boldsymbol{\mathcal{R}^*_{1_1}}, \mathcal{R}_{2_1}, \mathcal{R}_{3_1}, \mathcal{P}_1, \mathcal{P}_2\})$
end_2
s_1
s_2
ini_2
ini_1
dat_1
dat_1
end_2
s_3
s_4
ini_2
ini_1
ini_1
end_1
s_5
ini_2

Figure 29. Another revision of the first station to yield a complete solution.

5.5. *Discussion*

The preceding example makes it clear that the three revision procedures lead to different views of a system, enabling great flexibility for the stakeholders to select the most suitable modularization at any time:

- The procedure to revise white boxes modifies the original components to incorporate the behavior that they prohibit in others. In Fig. 23, for example, the model of $Station_1^1$ prevents the second station from initiating a transmission while the first one is transmitting (*ini*$_2$ is a non-possible event in state s_3, where it was originally unspecified).
- On the other hand, the transformation into gray boxes modifies each component to condition its own behavior to the previous relevant events of the others. In Fig. 29, for example, the model of $Station_1^3$ shows that the behavior of the first station is modified to take into account that it cannot initiate transmissions while the second one is occupying

the shared channel (i.e. between the occurrences of ini_2 and end_2). During that time, the station can only receive new data to transmit, if it had not already done so.

- Finally, the introduction of an aspect provides the vision of delegating the coordination of the components to a separate entity, that captures the inherently crosscutting nature of the $\mathcal{P}_1$ and $\mathcal{P}_2$ properties. That entity can continue being developed to define more complex coordination policies, enjoying the best separation of concerns. Note in Eqs. (9) and (10) that this procedure preserves the original coupling between the components (reflected in the $|||_\mathcal{M}$ operator), whereas the revisions that modify them lead to greater coupling: in Eqs. (7), (8), (11) and (12), $|||_\mathcal{M}$ is changed for $|[ini_1]|_\mathcal{M}$, $|[ini_1, ini_2]|_\mathcal{M}$, $|[ini_1, ini_2, end_2]|_\mathcal{M}$ and $|[ini_1, end_1, ini_2, end_2]|_\mathcal{M}$, respectively.

The procedure to revise white boxes is the one that enables the most extensive reasoning for the stakeholders, because the revised requirements still determine the whole functionality of the components affected. Nonetheless, preserving the white-box character can imply more changes than strictly necessary in a general case, because the modification of a requirement affects all the states of the MUS models where its premise is satisfied. The procedure of the gray boxes provides greater flexibility in this regard, inasmuch as it allows contradictory versions of a requirement to coexist; however, the stakeholders' decision is only supported by evolutions of the scenarios, which is also true for the introduction of a coordinator aspect. It is also important to note that second and third procedures are always applicable, whereas the first one can only be applied, obviously, when at least one of the components of the analyzed system is a white box.

6. Practical Evaluation

Our approach to CI is implemented and running in a development environment that we use to teach formal methods and software engineering methodologies in postgraduate courses. Actually, the mechanisms we have implemented correspond to the real-time extension of SCTL-MUS (described in [12]), in which the events of requirements, models and scenarios can be possible, non-possible or unspecified depending on specific time instants.

Knowing the intended scope of agile development methodologies (small to medium-sized projects, reduced teams and short deadlines [1]), we put our mechanisms into practice by assigning several development projects to postgraduate students, who should work in groups of three or more people, and with four months (the length of an academic cycle) to complete their work. The assignments involved real-time systems of an average complexity, excerpts from the Shuttle System case study [14] and NASA's CTAS air-traffic control system [31]. These were indeed the same assignments we made to evaluate the agile mechanisms introduced in [19] with the students of the previous academic year; thus, we had the opportunity to objectively measure the improvements due to the new mechanisms for CI.

In these experiments, we have found that the development times were reduced by an average of nearly 15%. This improvement was certainly due to the fact that, by automating the discharge of the integration changes, we eliminate the burden of what was formerly an entirely manual task, with very little methodological support. In turn, this encouraged the students to make integration checks more often, which lead to better quality in the developed specifications (i.e. fewer errors and non-considered situations).

Not surprisingly, all of the students started out the developments with white-box components, and tried to keep them white as long as possible. Thereby, the procedures to revise white boxes and to introduce coordinator aspects were the most widely used during the early stages of development, whereas the revision of gray boxes gained importance towards the latest stages.

7. Conclusions

Continuous integration (CI) is an important practice in agile development, because it provides for early detecting and resolving problems with the composition of various modular units of a system. To date, this feature was not conveniently supported by formal specification environments, due to the problem of effective loss of modularization. In response to that, we have introduced a general and flexible approach to integrate CI in an analysis-revision cycle, with automated support to discharge any changes made at the composition level into modifications of the original components, or into aspects that capture the cross-cutting nature of the integration checks. This way, we combine the benefits of handling modular specifications with the ability to keep emergent behavior under control during all the development process. Our experiments over the SCTL-MUS methodology have confirmed the advantages of this approach in the kind of projects that fall within the agile scope.

Acknowledgements

This work was funded by the Xunta de Galicia research project PGIDIT04PXIB32201PR.

References

[1] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. *Agile software development methods. Review and analysis.* VTT Publications, 2002.

[2] K. Altisen, F. Maraninchi, and D. Stauch. Exploring aspects in the context of reactive systems. In *Proc. of the Workshop on Foundations of Aspect-Oriented Languages, in conjunction with AOSD*, Lancaster, UK, Mar. 2004.

[3] S. Ambler. Agile Modeling (AM) home page: Effective practices for modeling and documentation. http://www.agilemodeling.com/.

[4] H. Baumeister. Combining formal specifications with Test-Driven Development. In *Proc. of the 4th Intl. Conf. on Extreme Programming and Agile Methods*, Calgary, Canada, Aug. 2004.

[5] M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, K. Schwaber, J. Sutherland, and D. Thomas. Manifesto for agile software development. http://agilemanifesto.org/, 2001.

[6] M. Boger, T. Baier, F. Wienberg, and W. Lamersdorf. *Extreme programming examined*, chapter Extreme modeling. Addison Wesley, 2001.

[7] T. Bolognesi. Toward constraint-object-oriented development. *IEEE Transactions on Software Engineering*, 26(7):594–616, 2000.

[8] Y. Bontemps, P. Heymans, and P. Schobbens. Lightweight formal methods for scenario-based software engineering. In *Proc. of the Intl. Workshop on Scenarios: Models, Transformations and Tools*, Dagstuhl Castle, Germany, Sept. 2005.

[9] M. Breen. Experience of using a lightweight formal specification method for a commercial embedded system product line. *Requirements Engineering*, 10:161–172, 2005.

[10] E. Brinksma. Specification modules in LOTOS. In *Proc. of the 2nd IFIP TC/WG6.1 Intl. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols*, Vancouver, Canada, Dec. 1989.

[11] G. Eleftherakis and A. Cowling. An agile formal development methodology. In *Proc. of the 1st South-East European Workshop on Formal Methods*, Thessaloniki, Greece, Nov. 2003.

[12] A. Fernández-Vilas, J. Pazos-Arias, A. Gil-Solla, R. Díaz-Redondo, J. García-Duque, and B. Barragáns-Martínez. Incremental specification with SCTL/MUS-T: A case study. *Journal of Systems and Software*, 70(2):189–208, 2004.

[13] M. Fowler. Continuous integration. *http://www.martinfowler.com*, 2006.

[14] H. Giese and F. Klein. Autonomous shuttle system case study. *Lecture Notes in Computer Science*, 3466:90–94, 2004.

[15] M. Glinz. Improving the quality of requirements with scenarios. In *Proc. of the World Congress for Software Quality*, Yokohama, Japan, Sept. 2000.

[16] S. Graf and B. Steffen. Compositional minimization of finite state systems. In *Proc. of the 2nd Intl. Workshop on Computer-Aided Verification*, New Brunswick (NJ), USA, June 1990.

[17] ITU. Message sequence charts. Recommendation Z.120, 1996.

[18] J. Lennox and H. Schulzrinne. *Feature Interaction in Telecommunications and Software Systems VI*, chapter Feature interaction in Internet telephony. IOS Press, 2000.

[19] M. López-Nores, J. Pazos-Arias, J. García-Duque, Y. Blanco-Fernández, R. Díaz-Redondo, A. Fernández-Vilas, A. Gil-Solla, and M. Ramos-Cabrer. Bringing the agile philosophy to formal specification settings. *Intl. Journal of Software Engineering and Knowledge Engineering*, 16(6):951–986, 2006.

[20] E. Mäkinen and T. Systä. MAS – an interactive synthesizer to support behavioral modelling in UML. In *Proc. of the 23rd Intl. Conf. on Software Engineering*, Toronto, Canada, May 2001.

[21] L. McIness. The agility of lightweight formal methods. *http://www.kuro5hin.org/story/2006/10/11/163923/00*, 2006.

[22] R. Milner. *Communication and concurrency*. Intl. Series in Computer Science. Prentice Hall, 1989.

[23] J. Mogul. Emergent (mis)behavior vs. complex software systems. *ACM SIGOPS Operating Systems Review*, 40(4):293–304, 2006.

[24] R. Paige and P. Brooke. Agile formal method engineering. In *Proc. of the 5th Intl. Conf. on Integrated formal Methods*, Eindhoven, The Netherlands, Dec. 2005.

[25] J. Pazos-Arias and J. García-Duque. SCTL-MUS: A formal methodology for software development of distributed systems: A case study. *Formal Aspects of Computing*, 13:50–91, 2001.

[26] J. Pazos-Arias, J. García-Duque, M. López-Nores, and B. Barragáns-Martínez. Eliciting requirements and scenarios using the SCTL-MUS methodology. The shuttle system case study. *ACM Software Engineering Notes*, 30(4), 2005.

[27] C. Prehofer. Feature interactions in statechart diagrams or graphical composition of components. In *Proc. of the 2nd Intl. Workshop on Aspect-oriented Modeling with UML, in conjunction with UML*, Dresden, Germany, Oct. 2002.

[28] E. Pulvermueller, A. Speck, J. O. Coplien, M. D'Hondt, and W. de Meuter, editors. *Feature interaction in composed systems*. Springer, 2002.

[29] S. Robak and B. Franczyk. Feature interaction and composition problems in software product lines. In *Proc. of the Workshop on Feature Interaction in Composed Systems, in conjunction with ECOOP*, Budapest, Hungary, June 2001.

[30] A. Salah, R. Mizouni, R. Dssouli, and B. Parreaux. Formal composition of distributed scenarios. In *Proc. of the 24th IFIP WG 6.1 Intl. Conf. on Formal Techniques for Networked and Distributed Systems*, Madrid, Spain, Sept. 2004.

[31] B. D. Sanford, K. Harwood, S. Nowlin, H. Bergeron, H. Heinrichs, G. Wells, and M. Hart. Center/TRACON automation system: Development and evaluation in the field. In *Proc. of the 38th Annual Air Traffic Control Association Conference*, Washington D.C., USA, Sept. 1994.

[32] B. Stepien and L. Logrippo. Feature interaction detection using backward reasoning with LOTOS. In *Proc. of the 14th IFIP WG6.1 Intl. Symposium on Protocol Specification, Testing and Verification*, Vancouver, Canada, June 1994.

[33] S. Suhaib, D. A. Mathaikutty, S. K. Shukla, and D. Berner. XFM: An incremental methodology for developing formal models. *ACM Transactions on Design Automation of Electronic Systems*, 10(4):589–609, 2005.

[34] C. Thomson and M. Holcombe. Applying XP ideas formally: The story card and extreme X-machines. In *Proc. of the 1st South-East European Workshop on Formal Methods*, Thessaloniki, Greece, Nov. 2003.

[35] S. Uchitel and J. Kramer. A workbench for synthesising behaviour models from scenarios. In *Proc. of the 23rd Intl. Conf. on Software Engineering*, Toronto, Canada, May 2001.

[36] S. Uchitel, J. Kramer, and J. Magee. Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM Transactions on Software Engineering and Methodology*, 13(1):37–85, 2004.

[37] F. Valles-Barajas. A formal model for a requirements engineering tool. In *Proc. of the ACM SIGSOFT 1st Alloy Workshop, in conjunction with FSE*, Portland (OR), USA, Nov. 2006.

[38] J. Whittle and I. Krüger. A methodology for scenario-based requirements capture. In *Proc. of the Intl. Workshop on Scenarios and State Machines, in conjunction with ICSE*, Edinburgh, UK, May 2004.