Electronic Communications of the EASST
Volume 40 (2011)

Proceedings of the 4th International Workshop on Petri Nets and Graph Transformation (PNGT 2010)

Modelling Emergency Scenarios using Algebraic High Level Net Transformation Systems with Net Patterns

Frank Trollmann, Maximilian Kern and Sahin Albayrak

35 pages

Guest Editors: Claudia Ermel, Kathrin Hoffmann
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Modelling Emergency Scenarios using Algebraic High Level Net Transformation Systems with Net Patterns

Frank Trollmann, Maximilian Kern and Sahin Albayrak

DAI-Labor, TU Berlin
Faculty of Electrical Engineering
and Computer Science
Frank.Trollmann@dai-labor.de, Maximilian.Kern@dai-labor.de, Sahin.Albayrak@dai-labor.de

Abstract: Emergency operations are a good case study for dynamic systems. Their size and high dynamicity make modelling them a challenging task. Algebraic high level net transformation systems are a well suited technique for modelling such dynamic systems. They consist of an algebraic high level net and a set of graph transformation rules. The net reflects the initial state of the operation and the transformation rules can be used to adapt this state to reflect the dynamicity of the operation. The applicability of graph transformation rules depends on the existence of a match morphism. While designing the algebraic high level net transformation system the designer has to ensure the existence of the right match morphisms for all reachable runtime states. This can be a tedious and error prone task for the designer. This paper uses a case study for modelling emergency operations with algebraic high level net transformation systems to show how the notion of net patterns can help the designer to cope with rule applicability.

Keywords: Graph Transformation, Algebraic High Level Nets, Design Patterns

1 Introduction

It is a challenging task to model emergency operations. Their complexity and the fact that the plan of the operation changes frequently at runtime can cause serious problems to the designer. Algebraic high level nets (AHL-Nets) are similar to Petri nets but are able to handle and process data. For this reason they are a well-suited technique to model such complex workflows. An AHL-Net transformation system additionally contains a set of graph transformation rules. These transformation rules can be used to adapt the AHL-Net structure in order to reflect a change in the plan of the emergency operation.

The applicability of graph transformation rules to an AHL-Net depends on the existence of a morphism between the left hand side of the transformation rule and the net, called match morphism. While modelling the AHL-Net transformation system the designer has to ensure the existence of the correct match morphisms. For this task she has to model the AHL-Net and the left hand sides of the transformation rules in a way that allows for these morphisms to exist.

This task can be tedious and error prone. The reason for this is the fact that the design of transformation rules and net structure are interrelated. The designer has to assure correct applicability of the graph transformation rules to all possible runtime structures. Starting from an

initial net structure at start-up of the application the possible runtime structures can be reached by incrementally applying transformation rules.

In this paper, we use pipeline emergency operations of the fire brigade as a case study to illustrate these problems and show how the application of design patterns for rule application, called net patterns, can be used to ease them. We first model the emergency operation without regard to rule application and show problems stemming from this approach. Afterwards, we introduce and apply a net pattern to show how this technique can be used to avoid these problems.

The paper is structured as follows. First, in Section 2 the language of algebraic high level net transformation systems is introduced. Afterwards the emergency scenario that serves as a case study for this paper is introduced in Section 3. This section also describes an approach to model these scenarios without regards to rule application and the resulting problems. A description of the approach of applying net patterns is then given in Section 4. Section 5 illustrates how such a pattern can be applied to the case study. This approach is then compared to the first approach in Section 6. Section 7 concludes this paper and hints at future work.

2 Algebraic High Level Net Transformation Systems

This section introduces AHL-Nets and AHL-Net transformation systems as a foundation for the case study. For a detailed description of algebraic high level nets the reader may refer to [PER95]. A detailed overview on graph transformation and related concepts can be found in [EEPT06]. The definitions in this section are also based on the definitions in these two references.

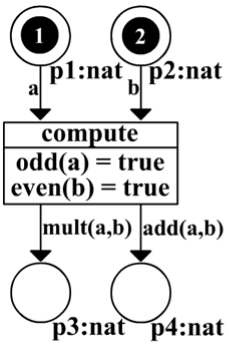
Algebraic high level nets are an extension of Petri nets. They also contain places, transitions and edges between them. The main difference to Petri nets is that AHL-Nets are able to process data. The types of data and operations used in the net are given by an algebraic specification. The actual data and implementation of the operations is contained in an algebra over this specification.

In an AHL-Net tokens are identified with data elements. Each place is associated with a sort of the specification and can only hold tokens of this type. Transitions can be used to process these data elements. Each edge is annotated with a term over the signature. This term specifies which kind of data is used and produced by the transition. A transition can additionally contain a set of equations that further constrain in which situations the transition can be fired.

Algebraic high level nets are formally described in Definition 1. An example for an algebraic high level net is given in Example 1.

Definition 1 (algebraic high level net) An algebraic high level net is consists of an eight-tuple $(SP, P, T, pre, post, cond, type, A)$, where $SP = (S, OP, E, X)$ is an algebraic specification, A is an (S, OP, E, X)-algebra, P is a set of places, T is a set of transitions, $pre, post : T \to (T_{OP}(X) \otimes P)^{\oplus}$ are functions denoting the set of places, connected to a transition via incoming and outgoing edges and the terms inscribed in these edges, $type : P \to S$ defines the type of each place and $cond : T \to \mathscr{P}_{fin}(Eqns(S, OP, X))$ defines the equations for each transition.

Example 1 *An example for an algebraic high level net is depicted on the left hand side of Figure 1. This net consists of one transition compute and four places p1 to p4. The purpose*



Signature Nat

sorts:
nat, boolean
opns:
true: boolean
false: boolean
mult: nat x nat -> boolean
add: nat x nat -> boolean
odd: nat -> boolean
even: nat -> boolean

Algebra Nat

sorts :
$Nat_{nat} = \mathbb{N}$
$Nat_{boolean} = \{true, false\}$
Opns:
$true_{Nat} = true$
$false_{Nat} = false$
$mult_{Nat}(a,b) = a * b$
$add_{Nat}(a,b) = a + b$
$odd_{Nat}(a) = \begin{cases} true & | \, a \,\%\, 2 = 1 \\ false & | \, else \end{cases}$
$even_{Nat}(a) = \begin{cases} true & | \, a \,\%\, 2 = 0 \\ false & | \, else \end{cases}$

Figure 1: Example - AHL-Net(left), Signature (middle) and Algebra (right).

of the net is to do some calculations on natural numbers. For this reason the net is typed over the signature of natural numbers, containing types for natural numbers and boolean values and several operations. This signature is depicted in the center of Figure 1. The signature does not contain any equations or variables. The right hand side of this figure shows the algebra that is used in the AHL-Net.

All places in the AHL-Net are typed with nat. p1 and p2 are already marked with the numbers one and two. The terms are inscribed on each arc. According to these terms the transition compute consumes two numbers from p1 and p2 and places their product on p3 and their sum on p4. The equations associated with compute state that this transition can only fire if the number taken from p1 is odd and the number taken from p2 is even.

The firing behavior of AHL-Nets is similar to that of place/transition nets. In addition to requiring the correct number of tokens on the pre condition places, a transition in an AHL-Net also requires the correct data elements to be available on these places. An AHL-Net transition fires in combination with an assignment of the variables used in its terms and equations. Based on this assignment, the terms on the pre condition edges are evaluated in order to determine the data elements required for this transition to fire. The terms in the post condition edges describe which data elements are produced by the transition. The transition can only fire if its equations are fulfilled under the assignment.

A morphism between two algebraic high level nets consists of mappings between their places and transitions. These two mappings have to be consistent with the functions pre and post, the arc inscriptions, the types of the places and the equations in each transition. An exact definition of AHL-Net morphisms is given in Definition 2.

Definition 2 (algebraic high level net morphism) An algebraic high level net morphism between two AHL-Nets $N_i = (SP, P_i, T_i, pre_i, post_i, cond_i, type_i, A)$, $i \in \{1,2\}$ with $SP = (S, OP, E, X)$ is defined as a tuple of morphisms $f = (f_P : P_1 \to P_2, f_T : T_1 \to T_2) : N_1 \to N_2$ such that the diagram shown in Figure 2 commutes.

Morphisms between AHL-Nets are an important concept for graph transformation on AHL-

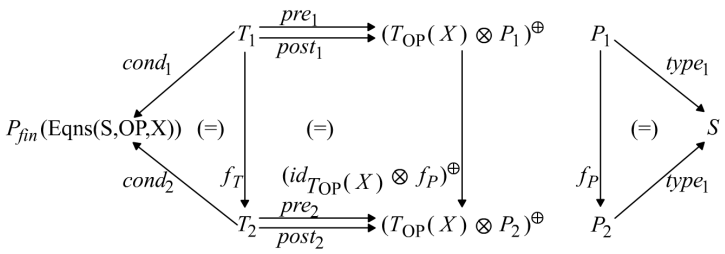


Figure 2: Condition for AHL-Net morphisms.

Nets. An AHL-Net transformation rule is described as a span of injective AHL-Net morphisms as defined in Definition 3. The application of a transformation rule requires the existence of a morphim between the left hand side of the transformation rule and the AHL-Net the rule is applied to. As a foundation for our transformation rules we use graph transformation in the double pushout approach.

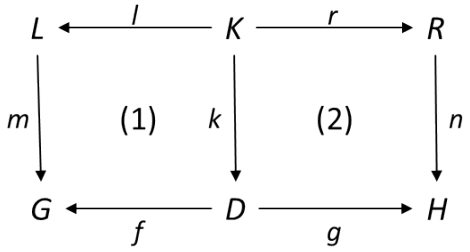


Figure 3: Application of an AHL-Net transformation rule.

Definition 3 (algebraic high level net transformation rule and transformation) An algebraic high level net transformation rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ consists of AHL-Nets L, K and R, called left-hand side, gluing graph and righ-hand side and two injective AHL-Net morphisms l and r.

The application of an AHL-Net transformation rule p to an AHL-Net G requires a morphism $m : L \to G$. The transformation is given by two pushouts (1) and (2) as depicted in Figure 3. The AHL-Net H is the result of the transformation.

An AHL-Net transformation system consists of an initial AHL-Net and a set of AHL-Net transformation rules. Such a transformation system is a way of describing the set of AHL Nets that can be derived from applying the transformation rules to the initial AHL-Net. A formal definition can be found in Definition 4.

Definition 4 (algebraic high level net transformation system) An algebraic high level net transformation system $TS = (N, Rules)$ consists of an algebraic high level net N and a set of AHL-Net transformation rules $Rules$.

These formalisms are used in order to model the case study in the next section.

3 Case Study 1 : Emergency Scenarios

The applicability of transformation rules to AHL-Nets depends on the existence of match morphisms. While modelling an AHL-Net transformation system the designer has to ensure the existence of the correct match morphisms between the transformation rules and all possible runtime structures of the transformation system. The complexity of this task depends on the number of transformation rules and possible runtime structures. In this section we follow a modelling approach that a designer who is not explicitly considering the applicability of transformation rules might take. After modelling the emergency scenario with this approach we concentrate on two occasions that require a transformation of the net and use them to showcase the problems in rule applicability.

As a case study we use pipeline emergency scenarios. As an inspiration and source of information for these scenarios the website *www.pipelineemergencies.com* has been used. In these case studies, the workflow of a set of firefighters during a gas-leak operation is modelled. Such a workflow consists of a set of team members executing tasks in a certain order. During these tasks, additional data can be used or produced. The large size and high number of possible runtime changes make such scenarios challenging and thus a good case study. In fact, pipeline emergencies have already been used as a case study in other publications. In [HEP08] they are used as a case study for modelling with reconfigurable Petri nets. [Tro09] contains our previous work on this case study. In this paper, we modelled this case study using algebraic high level nets. During the modelling process, it became obvious that some kind of general structure or pattern in the AHL-Nets is required in order to keep an overview in the complexity of all possible scenarios and runtime changes. The patterns, used as an example for net patterns in this paper are actually taken from the notions first presented in [Tro09] and slightly altered. However, in [Tro09] the focus is not on the patterns but on how emergency scenarios in general can be modelled with reconfigurable AHL-Nets and how the reconfiguration can be controlled in a higher order net. Higher order nets are AHL-Nets that contain AHL-Nets as tokens and are able to control their firing behavior and apply transformation rules. The purpose of this paper is to explicitly show problems that occurred during the design process for [Tro09] and how net patterns can be used to solve these problems.

For reasons of space we use a simplified version of the case study in our elaborations. We focus on a limited subset of tasks, team members and data. We distinguish between three types of team members: a *Firefighter*, the specially trained *Medical Personal* and the *Team Leader* who leads the operation. Our set of tasks contains four elements: *Repair Gas Leak*, *Treat Injured Person*, *Call Reinforcements* and *Take Gas Reading*. We only use one type of data which is called *Gas Reading* and represents the results of a gas reading. The case study in [Tro09] contains more elements. Especially the set of task types is way larger.

Not all tasks can be executed by every team member type. The task *Treat Injured Person* requires medical knowledge and can therefore only be executed by a team member of the type *Medical Personal*. A table on which of our tasks can be executed by which team member types can be found in Figure 4. Data types are also restricted to be available only in certain tasks. The

	Firefighter	Medical Personal	Team Leader	Gas Reading
Repair Gas Leak	X	-	-	X
Treat Injured Person	-	X	-	-
Call Reinforcements	X	-	X	-
Take Gas Reading	X	-	-	-

Figure 4: Compatibility of tasks with team member and data types

Gas Reading can only be handled by the task *Take Gas Reading*.

Tasks can be adapted to the current situation. For instance the task *Repair Gas Leak* can involve an arbitrary number of firefighters and the task *Take Gas Reading* may take into account a previous gas reading if available.

An intuitive way to model the workflow of these emergency operations is to model each team member and data element as a token and each task as a transition. This is similar to the way this case study is modelled in [HEP08] with the addition that we use AHL-Nets instead of Petri nets. Each place of the AHL-Net is typed over the team member type or data type it may hold. In our example, the set of sorts of the algebraic specification of the AHL-Net contains *firefighter*, *medical personal*, *team leader* and *gas reading*. Tasks may involve different numbers and types of team members and data. The corresponding transition has one input place for each team member and each used data element and one output place for each team member and each produced data element.

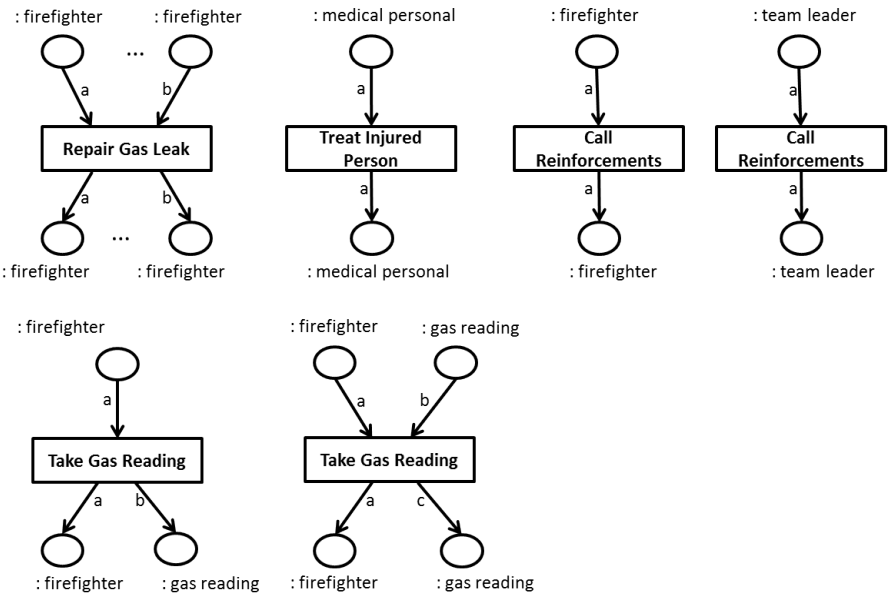


Figure 5: Possible transition structures for the tasks in the running example

The possible structures of transitions for our four example tasks are depicted in Figure 5. The structure of the task *Repair Gas Leak* of a *Firefighter* depends on the number of *Firefighters* that participate in this task. In order to avoid problems that may stem from an infinite number of possible structures, we assume that no more than five firefighters can participate in this task. *Treat Injured Person* is only executed by the team member type *Medical Personal* and thus only has one possible structure. *Call Reinforcements* can be executed by a *Firefighter* or *Team Leader*. For this reason two versions of this task do exist. *Take Gas Reading* is executed by a *Firefighter* and produces a *Gas Reading*. It may or may not take into account a previous reading. For this reason a second possible runtime structure exists where a gas reading is consumed and a new and updated version of the reading is produced. Depending on the newly determined gas values the new reading may differ from the old reading. This is expressed by using different variables in the arcs for the produced and consumed gas reading.

The restricted set of four tasks yields nine different transition structures. Five for the task *Repair Gas Leak* with different numbers of *Firefighters*, one for the task *Treat Injured Person*, one for the task *Call Reinforcements* (the version of this task for a firefighter actually has the same structure as *Repair Gas Leak* for one firefighter) and two possible structures for the task *Take Gas Readings*.

One key feature of emergency operations is their dynamic nature. The plan of an emergency operation changes frequently during its execution. Causes for such changes are reassessments of the situation. Possible changes are changes in the overall task order like the introduction, deletion or relocation of tasks or a change in the structure of an existing task.

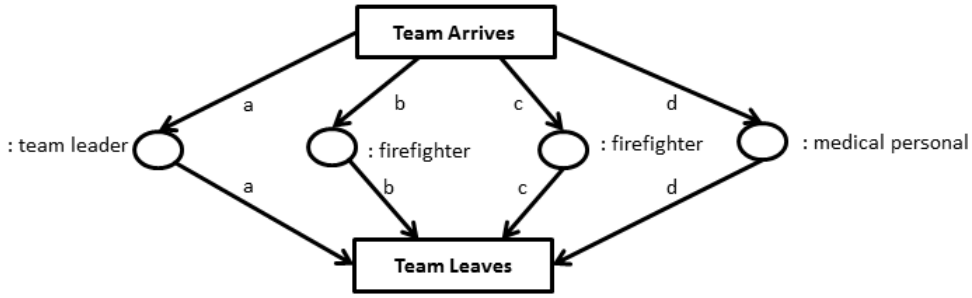


Figure 6: A model of the initial state of the application

For this reason the emergency scenario is modelled as an algebraic high level net transformation system rather than just as an AHL-Net. At startup the initial state of the transformation system is used as a model. This initial structure can be seen in Figure 6. It contains the transitions *Team Arrives* and *Team Leaves*. These transitions represent the team of firefighters arriving at the scene and leaving after the operation is finished. The team consists of one team leader, two firefighters and one medical personal. After the scenario is started the transformation rules are used to adapt the initial net to fit the current situation. This can happen while the net is executed.

Figure 7 shows the net at a later state of the operation. The initial net has been extended by adding several new tasks. The team leader has already set up the command post and assessed the situation. Her next task is to call for reinforcements. One firefighter has taken a gas reading

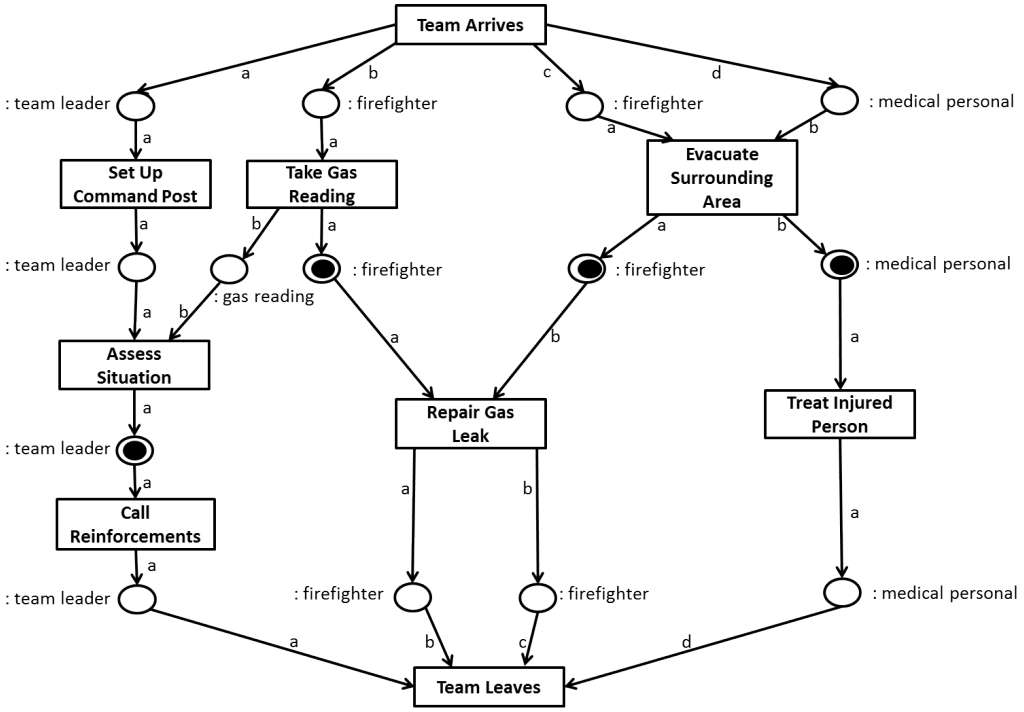


Figure 7: A model of a later state of the operation

while the other one evacuated the surrounding area with a medical personal. Both firefighters still need to repair the pipeline leak while the medical personal has to treat an injured person she encountered during the evacuation. In order to illustrate the complexity of emergency operations this example uses more than our four example tasks. However, this example is still considered one of the smallest emergency operations. Realistic scenarios involve a larger number of team members and tasks. The steps for transforming the initial net into this one and the used transformation rules can be found in Appendix A.

The firing of the AHL-Net and the execution of transformation rules represent different dimensions of runtime dynamics. Firing the AHL-Net represents an execution of the operation. Whenever a task is finished its transition is fired and leads to a new runtime state. The application of transformation rules represents a change in the operation. Whenever the team leader requires the plan of the operation to change she triggers the execution of one or more transformation rules. Their purpose is to introduce the required changes into the net.

Until now the intuitive approach of modelling the AHL-Net has worked. The workflow of the team of firefighters can be represented. The actions of each team member can be executed and tracked. This model is a well-suited way to model a fixed operation. However, it is far from optimal for rule applicability. The remainder of this chapter serves to show the problems that occur in this area. For this task we focus on two occasions that require the transformation of the AHL-Net and analyse how these changes can be achieved in this approach.

The first occasion is the introduction of a new task into the workflow of a firefighter. As

an example, we aim to insert the task *Repair Gas Leak* into the workflow of a firefighter at an arbitrary position (after any other task).

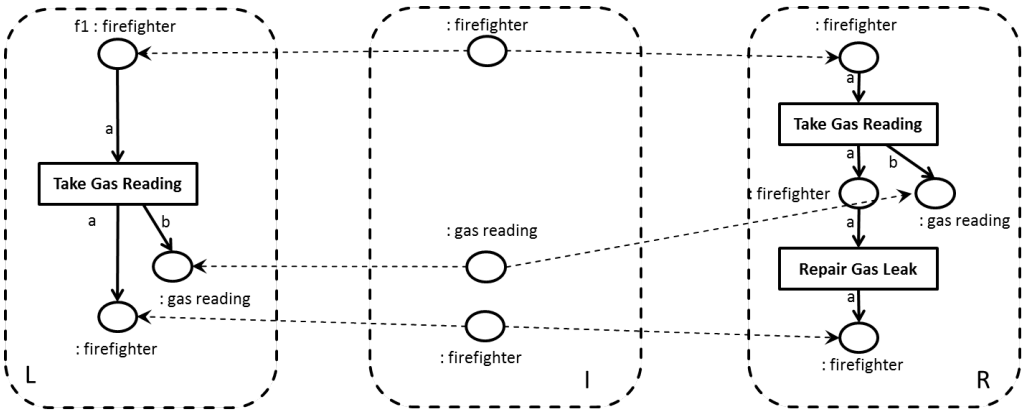


Figure 8: A transformation rule for inserting the task Repair Gas Leak after the task Take Gas Reading

A transformation rule for inserting this task after the task *Take Gas Reading* is depicted in Figure 8. This rule targets the task *Take Gas Reading* on its left hand side, temporarily removes it and reinserts it together with the new task. This rule can only be applied where the structure of *Take Gas Reading* occurs. It cannot be used to insert the task after a differently structured task. For this purpose additional transformation rules are required. In order to be able to insert this task anywhere into the workflow of a firefighter a total number of seven transformation rules is required (one for each possible task structure of a firefighter), even in our limited set of four tasks. The case study in [Tro09] contains a much larger set of tasks. This shows that the designer has to define a large set of transformation rules only for the purpose of inserting one task.

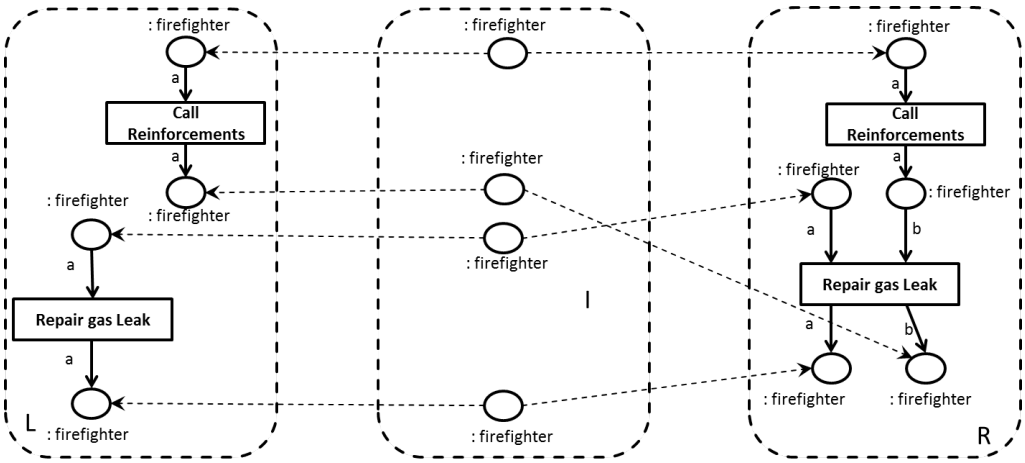


Figure 9: A transformation rule for adding a firefighter to the task Repair Gas Leak

A second occasion for a transformation is the requirement to change the existing task *Repair*

Gas Leak in order to add an additional firefighter. This is required if the leak is dangerous and an additional firefighter is needed for security reasons. This occasion also proves problematic. A rule for inserting a second firefighter into the task done by one firefighter can be seen in Figure 9. This rule can only be applied if currently one firefighter is doing the task and the other firefighter is doing the task *Call Reinforcements* or a similarly structured task before he joins the task *Repair gas Leak*. Again, in order to insert an additional firefighter in any situation several transformation rules are required; one for every combination of number of firefighters already in the task and previous task of the added firefighter.

However, there is another problem in this transformation rule. It is also applicable to the task *Call Reinforcements* of a firefighter. The structures and place types of this task and the task *Repair Gas Leak* for one firefighter are identical. AHL-Net Morphisms do not have to preserve the name of transitions. For this reason this transformation rule is applicable to the wrong task.

These two problems can be avoided if the designer considers rule application while modelling the AHL-Net. However, this also makes the modelling process much more complicated as she has to find a way to enable both correct rule application and correct execution logic of her net. The next section proposes to provide net patterns to the designer in order to ease these problems.

4 Net Patterns

While designing an algebraic high level net transformation system the design of the initial model and the design of the transformation rules are interwoven. If the model has not been designed for easy rule application, designing rules can become a difficult task and may even involve a revision of the original model structure. However, designing an AHL-Net for easy rule application is no trivial task. The fact that AHL-Nets are executable and thus every change in the net also changes its execution behaviour adds to this complexity.

In order to take these problems out of the hands of the designer we propose to provide her with structures she may use during the modelling process. This saves her the effort of having to come up with solutions for enabling easy rule application on her own. We call such structures net patterns. A net pattern is a subnet that has to be inserted by the designer in order to model a certain situation. This subnet may not be complete i.e. there may be parts of the net pattern where the designer is able to fill in a custom structure or define the types of certain places.

This is similar to the idea of design patterns originating in the work of Christopher Alexander [AIS77]. Such patterns are available for programming languages as guidelines for modelling certain reoccurring tasks such as making a class singleton. These patterns consist of a problem specification and a description on how it can be solved in a certain programming language. We can see net patterns as design patterns on AHL-Nets where the problem description is that the designer wants to model a certain situation and the solution is a description of the subnet the designer may use.

Net patterns can be used for several purposes. In this paper we propose to use them to ease the problems in rule application. The basic idea is to use net patterns in the AHL-Net to guarantee the existence of certain structures. These structures can be targeted by the transformation rules. A good design in the patterns can ease the tasks of rule application considerably.

The problems in the first case study can be solved by such patterns. This case study has two

main problems. The first problem stems from the large variety of different structures of tasks in the case study. This variety leads to the problem that targeting an arbitrary task, for example in order to add another task after it, requires an impracticable large set of transformation rules. In general, this problem occurs whenever a transformation rule should be general and applicable in a large variety of places inside the AHL-Net. In order to write such a general transformation rule one has to rely on the structural commonalities of the possible applications. If no such commonalities exist multiple versions of the transformation rule are required.

The second problem in the case study stems from the fact that a transformation rule can be applied for any match morphism that can be found. Some of these possible applications might not have been intended by the designer and thus are possible misapplication. In the first case study, a transformation rule that was intended to change the structure of one task can be applied to a different task it has not been intended for because this task has the same structure. One technique for achieving such a restriction is the use of application conditions for graph transformation rules [EH85, EEHP06, HP09]. Application conditions restrict the applicability of a transformation rule by restricting the surrounding context of the application. By using such conditions, additional structural requirements to the application context of the rule can be made. In normal application conditions the designer can specify which additional structures should exists in order to apply the transformation rule. There are also negative application conditions [HHT95] that can be used to forbid the existence of certain structures. These application conditions have been generalized to the more expressive nested application conditions [HP05]. Such application conditions also rely on the existence or non-existence of morphisms. Although these application conditions considerably increase the possibilities of the designer to constrain where her transformation rule can be applied, there are still some problems that cannot be solved with these techniques. These are the cases where two alternate applications, one correct and one incorrect, are completely identical in their structure. In our example from the case study the structures of the tasks *Call Reinforcements* and *Repair Gas leak* are identical. This means even with additional application conditions they cannot be distinguished from each other on a structural basis. These tasks can only be distinguished by actively modelling them with a different structure.

On the one hand, the different properties of tasks lead to so many different task structures that being able to target an arbitrary task requires an impracticable large set of transformation rules. On the other hand, the representations of some tasks have the same structure. This leads to possible misapplications of transformation rules that where meant to be applicable only for one task.

These problems can be solved by introducing a net pattern for tasks. In this approach each task consists of a subnet of the AHL-Net. The pattern then prescribes parts of the structure of this subnet. It contains parts that are identical for every task and parts that are specific to one task type only. This way transformation rules for generic purposes, such as the insertion, deletion or relocation of an arbitrary task, can target the parts of the pattern that are the same for all tasks while specific rules, e.g. a change that can only be applied to one task type, can target the parts that only occur in this task.

In Section 5 the emergency scenario is modelled with the help of such a net pattern.

5 Case Study 2: Modelling Emergency Scenarios with Net Patterns

As stated in Section 4 the modelling process of the case study can be considerably eased for the designer by the introduction of net patterns. We use a pattern that prescribes a structure for tasks done by certain team members. This pattern has to be applied on the nets algebraic specification and its AHL-Net structure. The specification needs to contain all sorts and operations used in the net structure.

The pattern on the specification level is based on the description of which team member type can execute which task types, defined in the case study. If this information is known the specification can be generated. The generated signature contains:

- **Team Members and Data:** For each team member and data type $\langle tm \rangle$ the sorts $\langle tm \rangle$, $preprocessing\_\langle tm \rangle$, $postprocessing\_\langle tm \rangle$ and constructing operations $preprocessing\_\langle tm \rangle : \langle tm \rangle \rightarrow preprocessing\_\langle tm \rangle$ and $postprocessing\_\langle tm \rangle : \langle tm \rangle \rightarrow postprocessing\_\langle tm \rangle$.
- **Task Types:** For each task type $\langle task \rangle$ and all team members and data types $\langle tm \rangle$ that are allowed to execute this task the sort $doing\_\langle task \rangle\_\langle tm \rangle$ and the constructing operation $doing\_\langle task \rangle\_\langle tm \rangle : \langle tm \rangle \rightarrow doing\_\langle task \rangle\_\langle tm \rangle$.

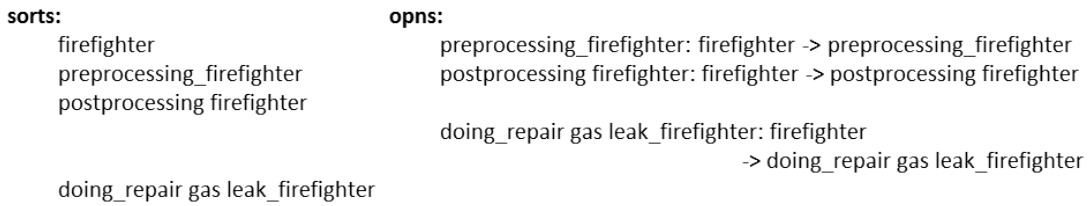


Figure 10: Excerpt from the induced specification of the example.

Part of the induced specification for the team member type *Firefighter* and the task *Repair Gas Leak* is depicted in Figure 10. The specification does not require any equations and variables. Based on the types and operations in this specification the net pattern for a task is defined. In addition, an algebra for the induced specification can be generated. This can be done based on an implementation for the team members and data types. For a data type of the sort *tm*, the implementation of *preprocessing_tm*, *postprocessing_tm* and *doing_task_tm* are exactly the same as *tm* for all tasks *task*. The operations for constructing these sorts out of the normal *tm* simply generate an identical piece of data. The implementations for the sorts for each team members could for example consist of the name of the team member and a set of additional attributes

The left hand side of Figure 11 shows the scheme for the net pattern for a task done by one team member. In order to be able to distinguish a general and a task-specific part of a task, the pattern introduces a preprocessing and postprocessing phase before and after the task. The preprocessing phase consists of the net around the transitions *Start Preprocessing* and *Finish*

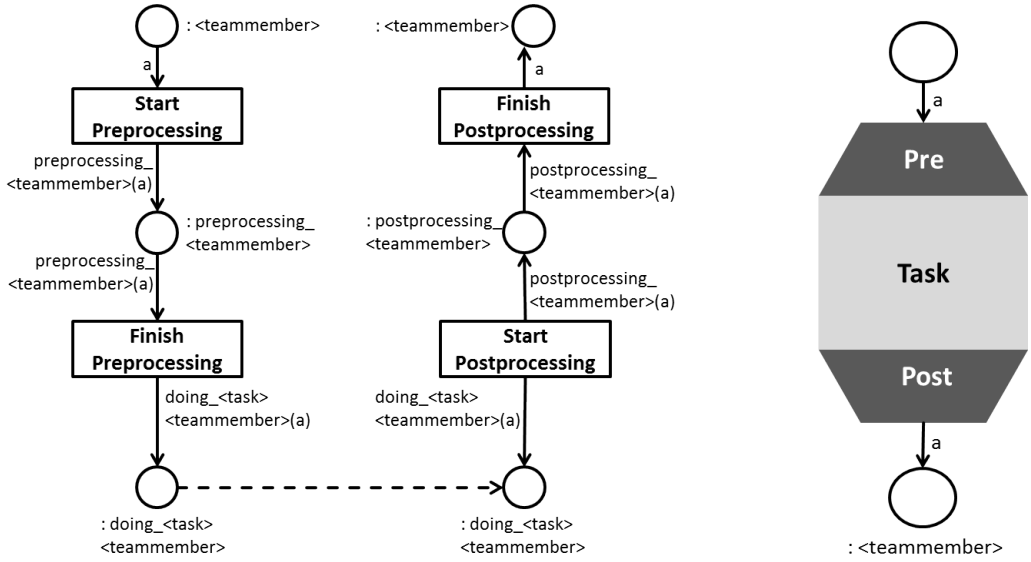


Figure 11: Left: The pattern of Places and Transitions used to represent a task. Right: A visual abbreviation for the pattern.

Preprocessing. This sub-pattern represents one team member preparing and then starting the task. Similarly, the task pattern ends with a postprocessing pattern that represents the ending of the task. This pattern consists of the net around the transitions *Start Postprocessing* and *Finish Postprocessing*. A Task may have multiple preprocessing and postprocessing patterns; one for each team member and piece of data participating in this task. In our general version, the core of the task is indicated by a dashed arrow. This is where the implementation of the task is inserted. Here, the designer is free to model the actual implementation of the task. Since the pattern contains several places and can get quite big in complex nets, Figure 11 also contains a visual abbreviation for the pattern. In this abbreviation *Pre* represents the preprocessing phase, *Post* represents the postprocessing phase and *Task* represents the inner implementation of a task. This abbreviation is be used in several figures to save space.

The types of the places are also part of the pattern. In Figure 11, diamond brackets are used as a placeholder for a certain type. For instance, the type $doing\_\langle task\rangle\_\langle teammember\rangle$ has to be instantiated with a task name and a team member type. For the task *Treat Injured Person* of the team member *Medical Personal*, the type of the places used in the inner implementation of a task is $doing\_treat injured person\_medical personal$. For the net pattern all placeholders of the same name have to be instantiated with the same type. Two dependent types are contained in this pattern. One represents the executing team member and the other one the executed task. The types that are used in the pattern are all part of the induced specification.

These types are used to accomplish the general and task-dependent parts of a task. The general parts of the task pattern are the ones whose place-types do not contain the task name. These are the transitions *Start Preprocessing* and *Finish Postprocessing* and the places connected to them. From the point of view of one team member each task starts with a preprocessing phase and ends

with a postprocessing phase which contains these structures. They can be targeted by general transformation rules.

The types of the places on the inner task structure contain the name of the task. This means that for any two different tasks the types of this part of the net pattern are different. Specific transformation rules can make use of this fact.

The tasks in case study 1 depicted in Figure 5 can now be translated to tasks using the pattern. For each incoming arc in the transition representing the task, the pattern contains one preprocessing pattern typed over the type of the place connected to the incoming arc. For example, if the arc is typed with *firefighter* and the task is *take gas reading* the types of the places in the preprocessing pattern from start to end are *firefighter*, $preprocessing\_firefighter$ and *doing_take gas reading_firefighter*. Analogous, each outgoing arc of the transition in case study 1 is modelled as a postprocessing pattern containing the according types. The core of the task can be implemented by using the task structure from case study 1, substituting the types of the places with the correct type inside of the task (for instance *firefighter* is substituted with *doing_take gas reading_firefighter* inside the task take gas reading) and connecting each place to the corresponding preprocessing pattern. If required, the inner structure of a task can also be modelled in a more sophisticated way by substituting this one transition with a subnet. This allows to model subtasks for each task. An example of a conversion of a task and transformation rule can be found in Appendix B.

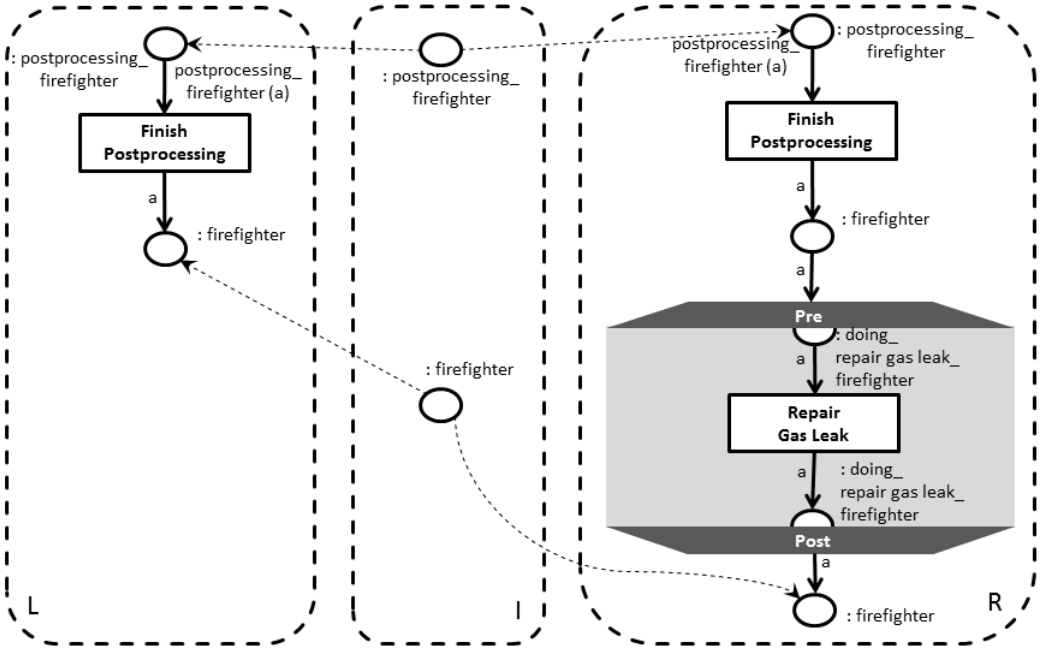


Figure 12: General insertion rule for the task *Repair Gas Leak*.

The net pattern enables a more comfortable formulation of transformation rules. Figure 12

shows how the task *Repair Gas Leak* can be inserted with net patterns. On its left hand side the transformation rule contains the transition *Finish Postprocessing* from our task pattern for the team member type *Firefighter*. On the right hand side of the transformation rule the task *Repair Gas Leak* is inserted after this transition. Since the transition *Finish Postprocessing* is contained in the general part of our net pattern it is contained in every task of the team member firefighter. Thus, this transformation rule can be used to insert the task *Repair Gas Leak* anywhere in the workflow of a firefighter. This is the insertion rule that proved problematic in Section 3. Insertion rules can always be formulated similar to this example. The left hand side contains the transition *Finish Postprocessing* for all participating team members and used data elements. The right hand side inserts the task in its initial structure conforming to the pattern after these transitions.

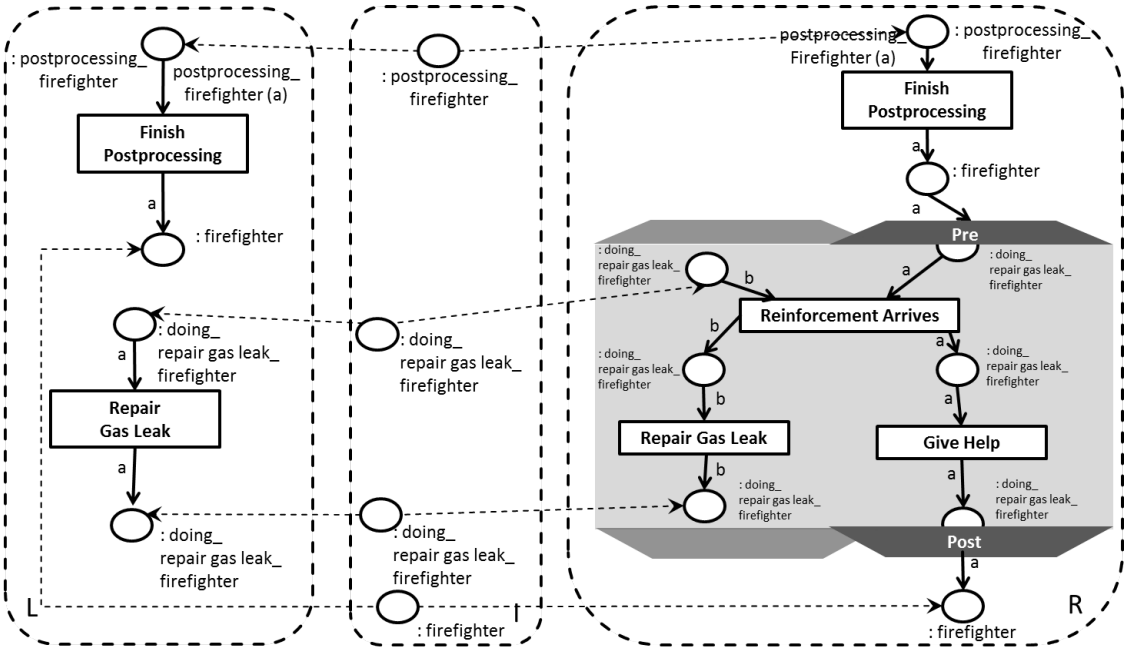


Figure 13: A transformation rule for adding a firefighter to the task *Repair Gas Leak*.

The second problem in the first case study occurs when trying to add a new firefighter to the task *Repair Gas Leak*. In addition to the problem of not being general enough, it is not possible to formulate the rule in a way that forbids misapplications to other tasks in the first case study. Using the net patterns this transformation rule can be formulated as depicted in Figure 13. The transformation rule targets one transition inside of the Task *Repair Gas Leak* and the *Finish Postprocessing* transition of the second firefighter. From the point of view of this second firefighter the rule works like any other insertion rule. For the first firefighter the transformation rule adds the subtask of waiting for the second firefighter to arrive before executing the already existing subtask. The transformation rule uses places of the type *doing_repair gas leak_firefighter*. According to our net pattern this type can only occur inside of the task *Repair Gas Leak* of the team member type *Firefighter*. For this reason the transformation rule cannot be applied to any other

task. This transformation rule solves the problems that occurred in the first case study.

These two transformation rules show that the introduced patterns enable the designer to formulate more efficient and safe transformation rules. The results from both case studies are compared in the next section.

6 Evaluation and Related Work

This paper shows two ways of modelling emergency operations with algebraic high level net transformation systems. The case study in Section 3 shows that an intuitive way of modelling the AHL-Net can lead to problems in rule application.

The second case study in Section 5 uses a net pattern to model the same operations. Although the net pattern leads to a larger AHL-Net structure it enables efficient definitions of transformation rules.

One problem that occurs in the first case study is the number of transformation rules required to insert a task anywhere in the workflow of a team member. The different structures require a specialized insertion rule for each possible task structure. Even the restricted set of four tasks requires seven transformation rules in order to insert one task anywhere in the workflow of its corresponding team member. In a realistic set of tasks this number is much higher. The application of net patterns in the second case study solves this problem. In this case study only one rule is able to insert a task anywhere in the workflow for a team member type. This number is independent of the number of structures of the available tasks as long as they are modelled with the net patterns. This specific problem can be seen as an example for a class of problems where the designer wants to create a general reconfiguration rule that should be applicable in several places throughout the net. In order to do this she can rely on a net pattern in order to restrict the targeted places of application in the net to the same structure. This way this structure can be targeted in a generic rule. In order for this to work the modelled system under study should be decomposable in logical units or parts, like our tasks.

The second problem in the first case study stems from the fact that some tasks have identical structures. This leads to a possible misapplication of transformation rules that should only be applicable within the scope of one specific task. Again, the net patterns are able to eliminate this problem. In these patterns each task contains a unique type for its inner places. If a transformation rule uses this unique type it can only be applied to the according task. Again, this problem is an example for a larger class of similar problems where the same structure in the net occurs multiple times and not all of these occurrences are considered correct applications of the transformation rule. This problem can be eased by using application conditions and restricting the possible places of application. A net pattern can explicitly force the designer to choose different structures in order to distinguish between the right and wrong places of application. In our case study we used the typing of the places in order to enforce this difference.

This shows that the introduced net pattern has successfully eased the problems of the designer. If this net pattern is provided to the designer, either as a part of the modelling environment or in a descriptive way as a design pattern, this can save her a lot of time since she does not have to come up with possible solutions and validate them herself.

Emergency scenarios are a popular case study for showing the capabilities of modelling lan-

guages. Their large complexity and frequent changes make them a good example to show the capabilities or problems of modelling languages. For instance, in [HEP08] the application of reconfigurable systems for modelling the dynamic aspects of such scenarios is proposed and illustrated on Petri nets. This work also is the foundation for [Tro09] from where our net patterns originate.

A different approach to modelling dynamic emergency scenarios is presented in [RM08]. In this approach the process is modelled as a questionnaire that contains decision points. This questionnaire contains all possible courses of actions in the scenario. This is another view on modelling these dynamics. Instead of modelling the workflow of the team only the decisions that change this workflow are modelled. This model could, for example, be used to decide when a transformation rule is applied.

Basically our case study models the workflow of a team of firefighters with the special property that this workflow may change at runtime. [AWW03] focuses on workflow modelling and the required perspectives on the workflow. This publication also introduces several approaches towards workflow modelling and compares them regarding their capability to model these five perspectives. In [Tro09] we show that our modelling of an algebraic high level net transformation system not only covers these perspectives but also is able to cope with the changes in these perspectives that can occur during the execution of the operation.

Net patterns on modelling languages are an obvious extension. Several authors have considered this approach in order to convey solutions to various problems. For example in [JN98] several reusable design patterns for Petri nets are proposed. [MA05] proposes design patterns on colored Petri nets. This work is part of the results from the *Workflow Patterns Initiative* that researches patterns in process aware information systems. A collection of the patterns required in this context can be found in [VTKB03].

These references concentrate on the use of patterns for structural or behavioural properties of certain modelling languages. The use of patterns for graph transformation has also been researched. One promising approach is introduced in [AVK+05]. However, this approach focuses on the use of design patterns for the formulation of graph transformation units to accomplish a certain task while we propose to also use design patterns in the targeted models for controlling the applicability of the transformation rules.

7 Conclusion and Future Work

The two case studies in this paper show the advantage of the use of net patterns. In fact, the patterns in the second case study are the result of multiple cycles of trial and error by the author of [Tro09] while modelling the emergency scenarios. In order to support other designers with similar problems it makes sense to formulate the patterns as design patterns that can be reused in a similar situation.

Algebraic high level nets are not the only modelling language that may encounter certain problems in connection with the applicability of graph transformation rules. Since this applicability is mainly determined by pattern matching it makes sense to use structural patterns to ease these problems. Thus, we think the approach of design patterns for rule applicability may be of use in other modelling languages as well.

One property of morphisms that is of special use to the case study is the requirement to preserve types. The types of the places have proven to be useful in restricting where a transformation rule can be applied. On the other hand, one could argue that our patterns misuse a typing concept that has been introduced for restricting which kinds of data a place may hold. In fact, we use the types mainly for restricting where a graph transformation rule can be applied. For future work it is interesting to look into whether both concepts can be decoupled. A special kind of typing could be introduced that is only relevant for morphisms and thus can be used to restrict the applicability of graph transformation rules without influencing the structure of the model.

Bibliography

[AIS77] C. Alexander, S. Ishikawa, M. Silverstein. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York, 1977.

[AVK+05] A. Agrawal, A. Vizhanyo, Z. Kalmar, F. Shi, A. Narayanan, G. Karsai. Reusable Idioms and Patterns in Graph Transformation Languages. *Electronic Notes in Theoretical Computer Science* 127(1):181–192, March 2005.

[AWW03] W. M. P. van der Aalst, M. Weske, G. Wirtz. Advanced Topics In Workflow Management: Issues, Requirements, And Solutions. *J. Integr. Des. Process Sci.* 7:49–77, August 2003.

[EEHP06] H. Ehrig, K. Ehrig, A. Habel, K.-H. Pennemann. Theory of Constraints and Application Conditions: From Graphs to High-Level Structures . *Fundamenta Informaticae* 74(1):135–166, 2006.
http://fi.mimuw.edu.pl/vol74.html

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theor. Comp. Science. 2006.

[EH85] H. Ehrig, A. Habel. Graph grammars with application conditions. In Rozenberg and Salomaa (eds.), *The Book of L*. Pp. 87–100. 1985.

[HEP08] K. Hoffmann, H. Ehrig, J. Padberg. Flexible Modeling of Emergency Scenarios using Reconfigurable Systems. *Formal Modeling of Adaptive and Mobile Processes. Electronic Communications of the EASST* 12, 2008.

[HHT95] A. Habel, R. Heckel, G. Taentzer. Graph Grammars with Negative Application Conditions. *Fundamenta Informaticae* 26:287–313, 1995.

[HP05] A. Habel, K.-H. Pennemann. Nested Constraints and Application Conditions for High-Level Structures. In *Formal Methods in Software and System Modeling. LNCS 3393*. P. 293308. Internationales Begegnungs- und Forschungszentrum fuer Informatik, 2005.

[HP09] A. Habel, K.-h. Pennemann. Correctness of high-level transformation systems relative to nested conditions†. *Mathematical. Structures in Comp. Sci.* 19:245–296, April 2009.
doi:10.1017/S0960129508007202
http://portal.acm.org/citation.cfm?id=1552068.1552070

[JN98] J. W. Janneck, M. Naedele. Introducing Design Patterns for Petri Nets. 1998. TIK-Report No. 39, February 1998.

[MA05] N. Mulyar, W. M. van der Aalst. Towards a Pattern Language for Colored Petri Nets. 2005.

[PER95] J. Padberg, H. Ehrig, L. Ribeiro. Algebraic High-Level Net Transformation Systems. *Mathematical Structures in Computer Science* 5:217–256, 1995.

[RM08] M. L. Rosa, J. Mendling. Domain-Driven Process Adaptation in Emergency Scenarios. In *Business Process Management Workshops*. Pp. 290–297. 2008.

[Tro09] F. Trollmann. Modeling Emergency Scenarios using Algebraic Higher Order Nets. Master's thesis, Technische Universität Berlin, 2009.
http://tfs.cs.tu-berlin.de/Diplomarbeiten/TFSdipl/09-FrankTrollmann.pdf

[VTKB03] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, A. P. Barros. Workflow Patterns. *Distrib. Parallel Databases* 14:5–51, July 2003.

A Transformation of the Initial Net for Szenario 1

The transformation from the initial net in Figure 6 to the net given in Figure 7 in Section 3 is accomplished by using a set of graph transformation rules. This section exemplifies this transformation by showing these rules.

In order to transform the initial net the three tasks *Set Up Command Post*, *Take Gas Reading* and *Evacuate Surrounding Area* are inserted after the task *Team Arrives*. The respective insertion rules are shown in Figure 14, Figure 15 and Figure 16. Each of these rules uses the task *Team Arrives* on its left hand side and inserts the structure of the respective task on its right hand side. The task *Set Up Command Post* is added to the workflow of the team leader whereas the task *Take Gas Reading* is added to the workflow of a firefighter. The task *Evacuate Surrounding Area* is added to the workflow of the second firefighter and the medical personal. This rule could actually also be applied in a second way, adding this task to the workflow of the other firefighter before the task *Evacuate Surrounding Area*.

The state of the net after the application of these three transformation rules is depicted in Figure 17. Note that this is already a valid operation. The team leader establishes a base while one firefighter has the task of determining whether there is any dangerous substance by taking a gas reading while the second firefighter and a medical personal are evacuating the area as a precaution.

After taking the gas reading by firing the transition *Take Gas Reading* it is determined that there is in fact a critical amount of gas in the air. According to this new information the team

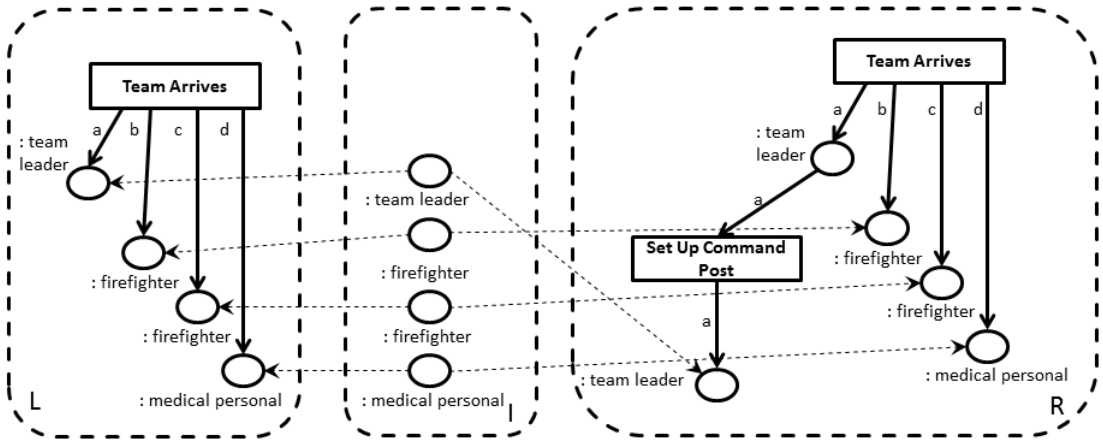


Figure 14: Insertion rule for the task *Set Up Command Post*.

leader has two new tasks. One task is to assess the situation by evaluating the previously taken gas reading. This task is inserted by using the transformation rule in Figure 18. This transformation rule has the structure of the task *Set Up Command Post* on its left hand side and can therefore only be used to insert the new task after this task. In addition, the team leader has to make a call and inform the responsible authorities so they may send reinforcements if required. This task in inserted after the previous task using the transformation rule in Figure 19. During the evacuation an injured person has been found. Thus, the medical personal has the additional task to treat this injured person. This task is added to the operation using the insertion rule in Figure 20. After adding these tasks to the operations plan the net is in a new state. Figure 21 shows this new state. The previously inserted tasks have been executed thus the marking of the net has changed. In accordance with the new information gained by executing these tasks the nets structure has been changed by adding the tasks as described above.

Finally, it is determined that the pipeline is save enough to be repaired by one of the firefighters. However, in case of an emergency another firefighter should stand by during the repair process. This task is inserted in two steps. First, the task *Repair Gas Leak* is inserted into the workflow of one firefighter using the insertion rule already given in Figure 8. A second rule, depicted in Figure 22 is used in order to add the second firefighter. This rule contains the previously inserted task *Repair Gas Leak* and the task *Evacuate Surrounding Area* of the participating firefighter on its left hand side. During the execution both tasks are deleted and reinserted. The task *Evacuate Surrounding Area* is reinserted as it is while the reinserted version of the task *Repair Gas Leak* now involves both firefighters. The result of the transformations is the net given in Figure 7.

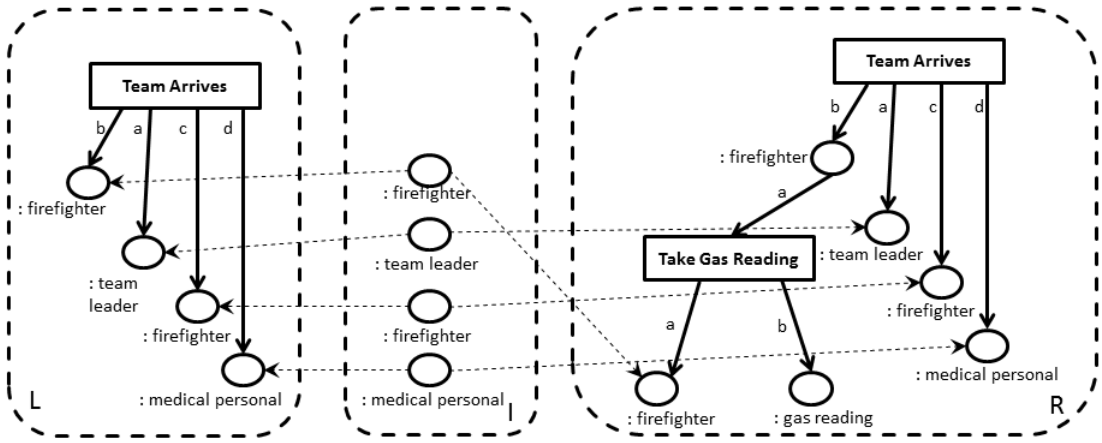


Figure 15: Insertion rule for the task *Take Gas Reading*.

B Conversion between Tasks and Rules from Case Study 1 and Case Study 2

Since the net patterns in this example provide a structure for tasks it is possible to transform the tasks from case study 1 into tasks using these patterns. This has already been explained in Section 5. This appendix serves to give an example of such a conversion and also show how a transformation rule from case study 1 can be converted into a transformation rule for case study 2.

As an example for a task conversion we use the task *take gas reading* in the version that uses an additional gas reading as input. The structure of this task in both case studies can be seen in Figure 23. During the conversion the structure of the transition *Take Gas Reading* is kept. However, its two incoming arcs and connected places are substituted by a preprocessing pattern typed over the respective team member / data and task. For instance the incoming arc, connected to the place typed with *firefighter* is substituted by a preprocessing pattern as indicated by the encircled areas in the figure. This pattern contains the transitions *start preprocessing* and *finish preprocessing* as specified in our pattern. These transitions are connected to places of the types *firefighter* (outer place), $preprocessing\_firefighter$ (place between the transitions) and $doing\_takegasreading\_firefighter$ (the inner place connected to the transition *Take Gas Reading*). Similarly each outgoing edge is substituted by a postprocessing pattern. The inner structure of our task in case study 2 is the same as the task structure of case study one. This inner structure could also be redefined in order to model subtasks. For instance, the task *take gas reading* could be divided into the subtasks of finding the gas leak, evaluating the previous reading and taking the new reading.

Figure 24 shows an example for a conversion from a transformation rule from case study 1 to a transformation rule in case study 2. Both rules insert the task *repair gas leak* in the respective case studies. In order to derive the transformation rule for the second case study from one for the

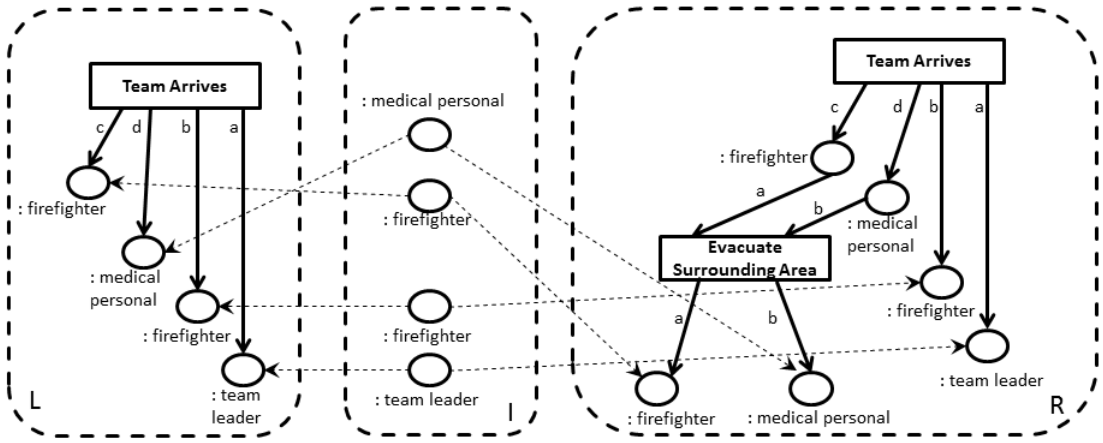


Figure 16: Insertion rule for the task *Evacuate Surrounding Area*.

first case study two things have to be done. On the left hand side the transformation rule from the first case study has the structure of the task *take gas reading* after which this task should be inserted. This structure is substituted in the second case study by a transition *Finish Postprocessig* from the postprocessing pattern as indicated by the encircled areas in the left hand sides of the transformation rules. The required transition is contained in the net pattern and thus is contained in the net at the end of any task of the team member firefighter. Thus, the transformation rule from case study 2 is more general than the example from case study 1 since it is able to insert the task anywhere in the net. If the designer explicitly wants the transformation rule to be applied only after the task *Take Gas Reading* she has to use more context in the transformation rule. For instance, she could also use the transition *Start Preprocessing* from the pattern. This transition is connected to one place typed over *doing_take gas reading_firefighter*. Thus, it ensures that the transformation rule can only target this specific task. In accordance with the substitution of the left hand side the preserved places in the gluing graph and the reinserted structure in the right hand side change.

In addition, the inserted task *Repair Gas Leak* on the right hand side changes. While the transformation rule from the first case study inserts the task as one transition the rule from the second case study uses the task pattern. The first example in this section already showed how the structure of a task in the first case study can be converted into the patterns from the second case study.

C Complete Signature of the Second Case Study

In the second case study the signature has been indicated as generated from the information about the task and team member relations. This appendix serves to give a complete signature for the tasks used in the case study. This signature is depicted in Figure 25.

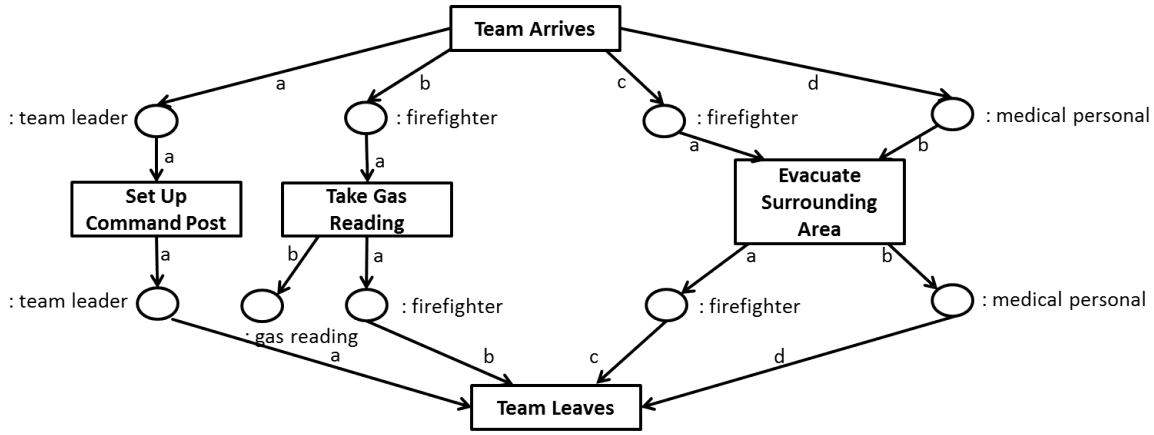


Figure 17: The net after the insertion of the tasks *Set Up Command post*, *Take Gas Reading* and *Evacuate Surrounding Area*.

D Detailed Modelling of the Second Case Study

This section shows how the models from Section 3 are modelled with the patterns introduced in Section 5.

The initial net that is used on startup of the application is depicted in Figure 26. This net corresponds to the initial net in the first case study. The task *Team Arrives* and *Team Leaves* have been modelled using the net pattern. *Team Arrives* has no preprocessing pattern and four postprocessing patterns representing the fact that during this task the four team members arrive and are available for the operation. Analogous, *Team Leaves* has four preprocessing and no postprocessing pattern, reflecting the fact that the four team members leave the operation.

Following the same situation as in the first case study in Appendix A, the three tasks *Set up Command Post*, *Take Gas Reading* and *Evacuate Surrounding Area* are inserted. The corresponding insertion rules are depicted in Figure 27, Figure 28 and Figure 29. Each of these insertion rules uses the postprocessing pattern on its left hand side. This means that they can actually be applied after any task of the correct team member. This is different from the corresponding rules in the first case study. These rules can only be used to insert the three tasks after the tasks they use in their left hand side. This means in order to generate another execution order of tasks, more rules are required in case study 1 whereas the rules in case study 2 can be reused for this purpose. The result of the application of all three rules is depicted in Figure 30.

The transformation rules for the insertion of the tasks *Assess Situation*, *Call Reinforcements* and *Treat Injured Person* are given in Figure 31, Figure 32 and Figure 33. Again, these insertion rules can easily be reused in order to generate other operations where the tasks are in different order while the transformation rules from case study 1 cannot. The net after the insertion of these three tasks is depicted in Figure 34.

The transformation rules for inserting and then altering the task *Repair Gas Leak* are depicted in Section 5 in Figure 12 and Figure 13. The first rule is a normal insertion rule. The second rule

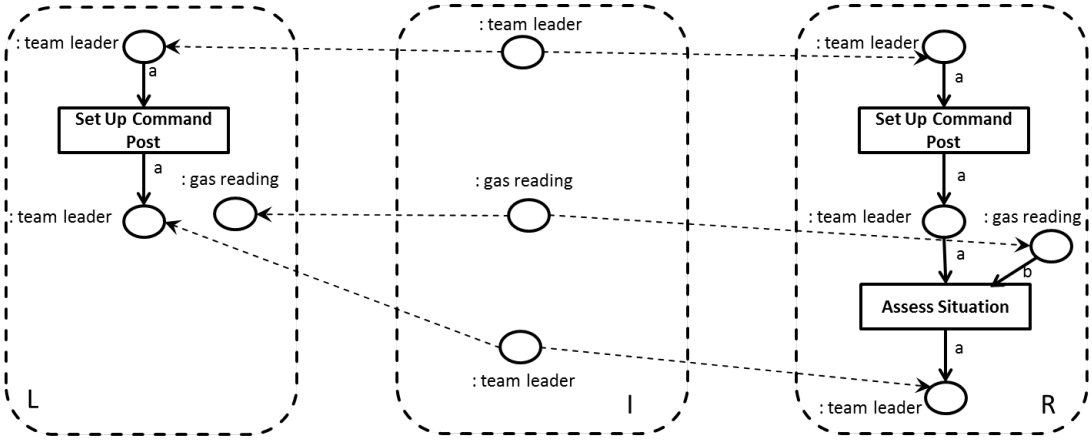


Figure 18: Insertion rule for the task *Assess Situation*.

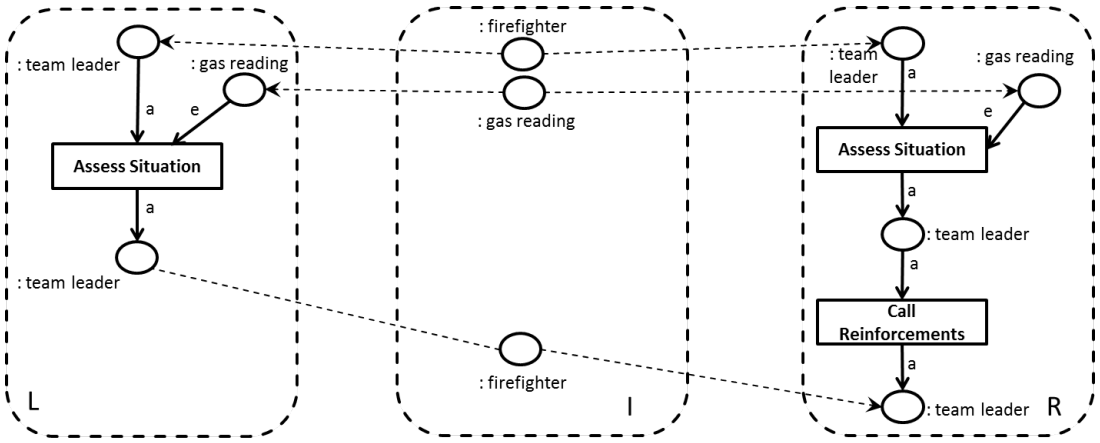


Figure 19: Insertion rule for the task *Call Reinforcements*.

actually targets the inner transition *Repair Pipeline* as well as a *Finish Postprocessing Transition* of the second team member. From the point of view of the already existing team member the task is altered. A new transition that represents the arrival of the reinforcement and a transition that represents the new team member who is giving help are added. From the point of view of the added team member the rule works like a normal insertion rule, targeting the postprocessing pattern and inserting a new task.

Since the transformation rule targets an inner transition of the task *Repair Gas Leak* it cannot be misapplied to change the structure of any other task. This transition is connected to places typed over *doing_repair gas leak_firefighter* which only occur in this task. Thus it cannot be misapplied to another task. Such a misapplication is possible for the corresponding rule in case study one, given in Figure 22 which is also applicable for the task *Call Reinforcements* of a firefighter.

The final net after the insertion and alteration of this task is depicted in Figure 35.

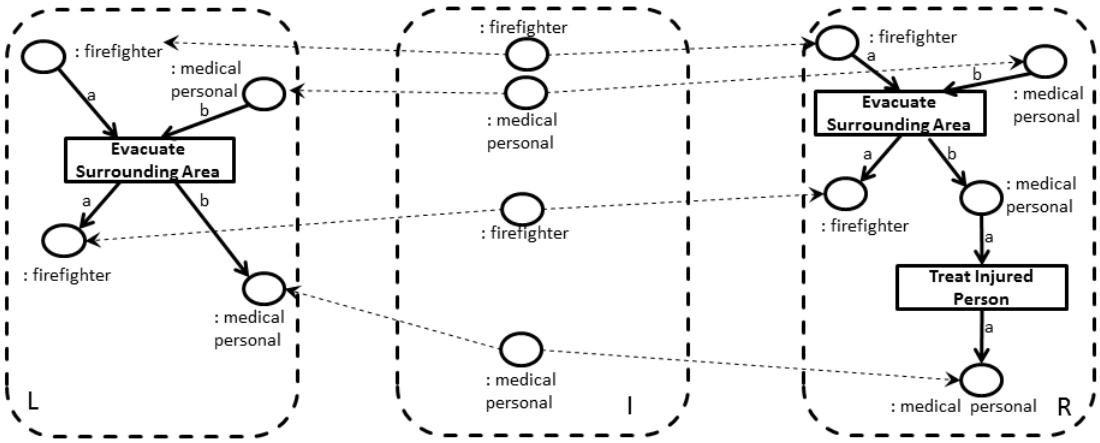


Figure 20: Insertion rule for the task *Treat Injured Person*

The generic nature of the transformation rules using the net patterns enables them to generate the set of all possible scenarios that contain the inserted tasks.

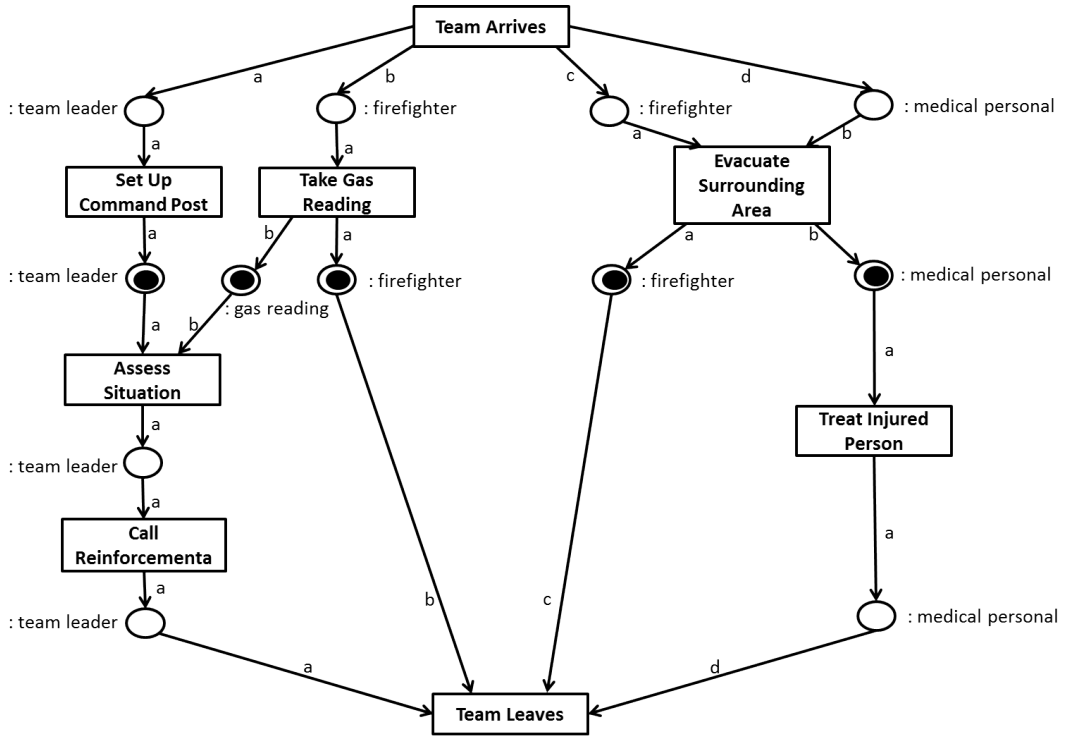


Figure 21: The net after the insertion of the tasks *Assess Situation*, *Call Reinforcements* and *Treat Injured Person*.

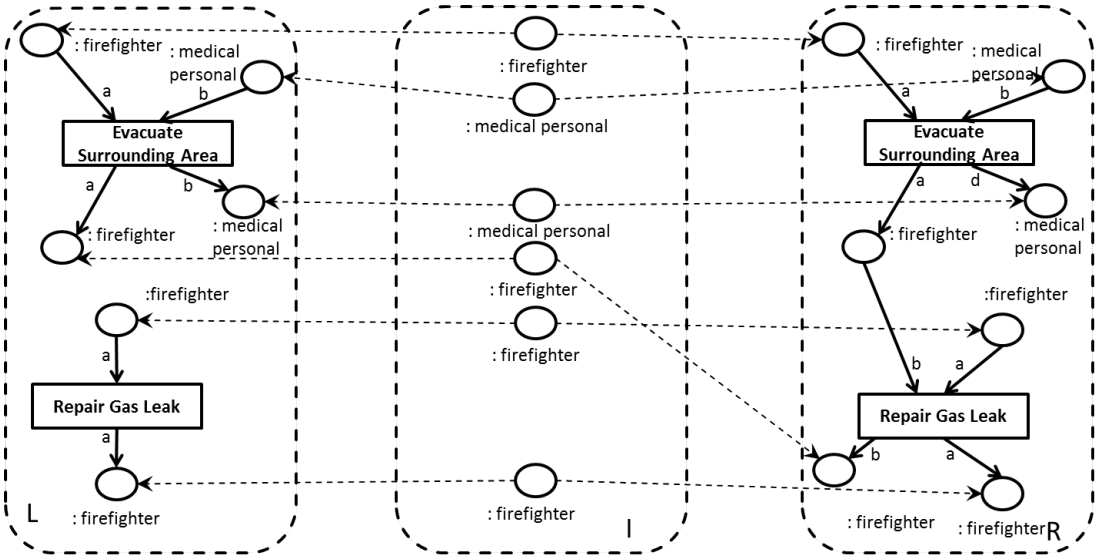


Figure 22: Rule for adding a firefighter to the task *Repair Pipeline*

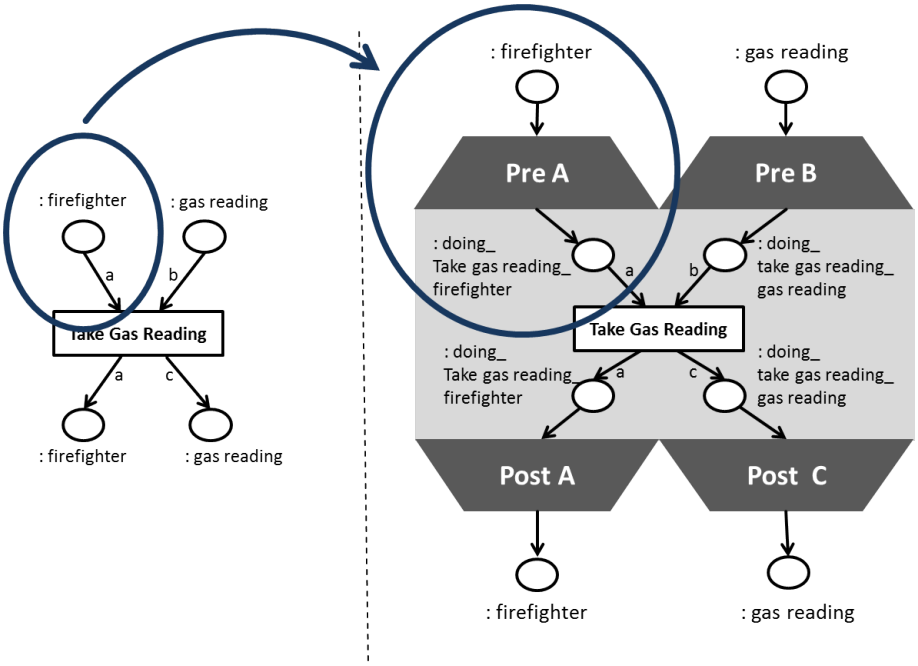


Figure 23: Conversion of a task from case study 1 (left) to a task using the patterns from case study 2 (right).

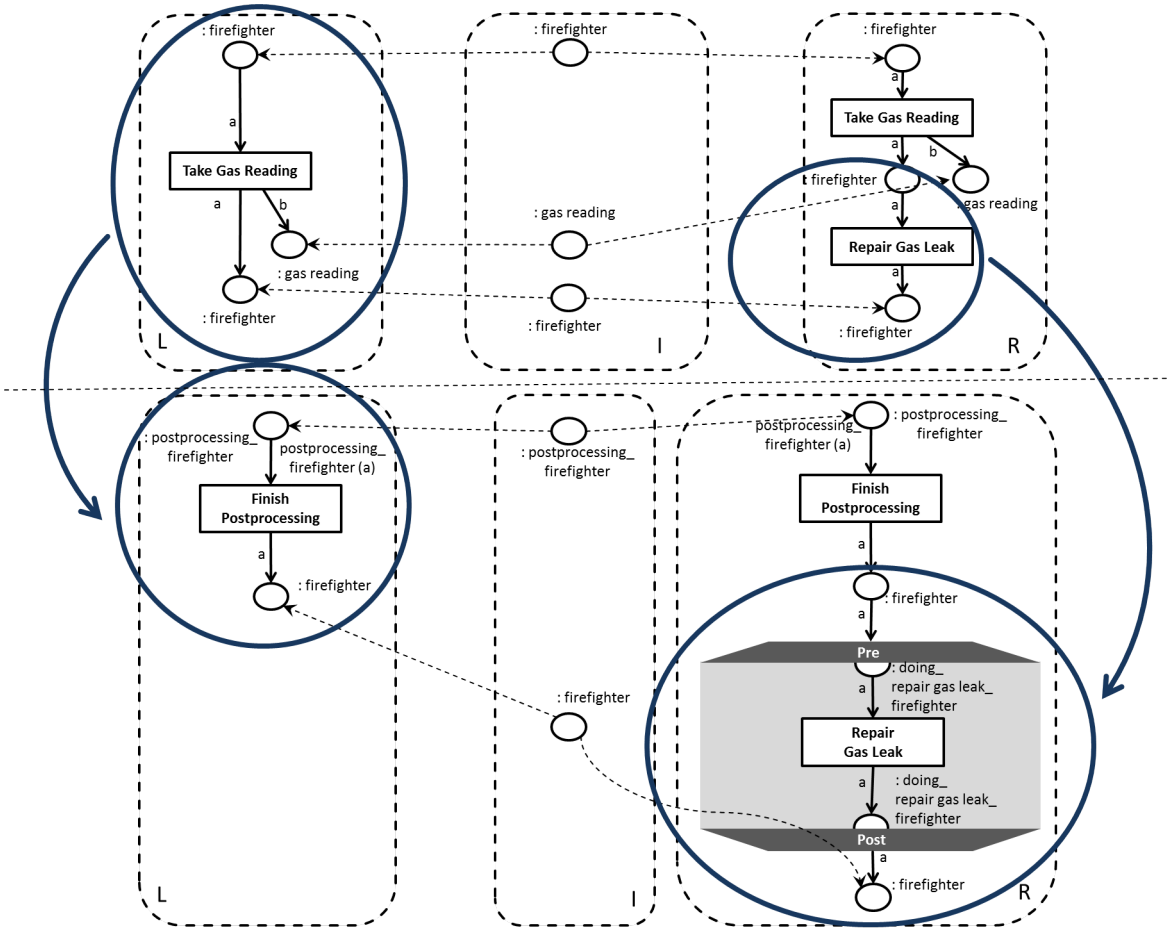


Figure 24: Conversion of a transformation rule from case study 1 (top) to a task using the patterns from case study 2 (bottom).

sorts:

firefighter
preprocessing_firefighter
postprocessing firefighter

team leader
preprocessing_team leader
postprocessing_team leader

medical personal
preprocessing_medical_personal
postprocessing_medical personal

gas reading
preprocessing_gas reading
postprocessing_gas reading

doing_team arrives_firefighter
doing_take gas reading_firefighter
doing_evacuate surrounding area_firefighter
doing_repair gas leak_firefighter
doing_team leaves_firefighter

doing_team arrives_team leader
doing_set up command post_team leader
doing_assess situation_team leader
doing_call reinforcements_team leader
doing_team leavers_team leader

doing_team arrives_medical personal
doing_evacuate surrounding area_medical personal
doing_treat injured person_medical personal
doing_team leaves_medical personal

doing_take gas reading_gas reading
doing_assess situation_gas reading

opns:

preprocessing_firefighter: firefighter -> preprocessing_firefighter
postprocessing firefighter: firefighter -> postprocessing firefighter

preprocessing_team leader: team_leader -> preprocessing_team leader
postprocessing_team leader: team leader -> postprocessing_team leader

preprocessing_medical personal: medical personal -> preprocessing_medical personal
postprocessing_mdeical personal: medical personal -> postprocessing_medical personal

preprocessing_gas reading: gas reading -> preprocessing_gas reading
postprocessing gas_reaidng: gas reading -> postprocessing_gas reaidng

doing_team arrives_firefighter: firefighter
-.> doing_team arrives_firefighter
doing_take gas reading_firefighter: firefighter
-> doing_take gas reading_firefighter
doing_evacuate surrounding area_firefighter: firefighter
-> doing_evacuate surrounding area_firefighter
doing_repair gas leak_firefighter: firefighter
-> doing_repair gas leak_firefighter
doing_team leaves_firefighter: fiefighter
-> doing_team leaves_firefighter

doing_team arrives_team leader: team leader
-> doing_team arrives_team leader
doing_set up command post_team leader: team leader
-> doing_set up_command post_team leader
doing_assess situation_team leader: team leader
-> doing_assess situation_team leader
doing_team leaves_team leader: team leader
-> doing _team leaves_team leader

doing_team arrives_medical personal: medical personal
-> doing_team arrives_medical personal
doing_evacuate surrounding area_medical personal: medical_personal
-> doing_evacuate surrounding area_medical personal
doing_treat injured person_medical personal: medical personal
-> doing_treat injured person_medical personal
doing_team leaves_medical personal: medical personal
-> doing_team leaves_medical personal

doing_take gas reading_gas reading: gas reading
-> doing_take gas reading_gas reading
doing_assess situation_gas reading: gas reading
-> doing_assess situation_gas reading

Figure 25: The complete signature for case study 2

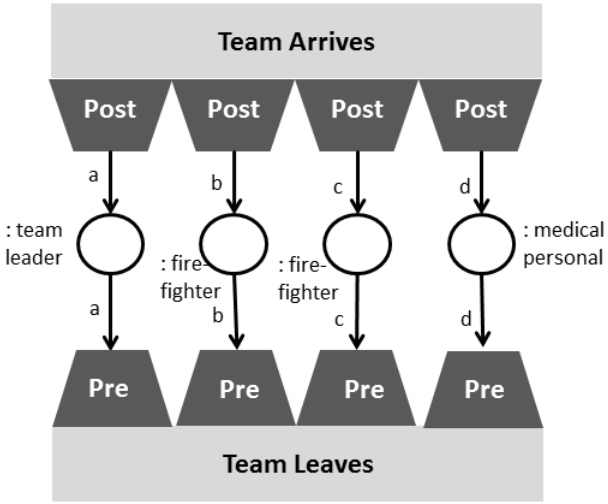


Figure 26: The initial net for the second case study

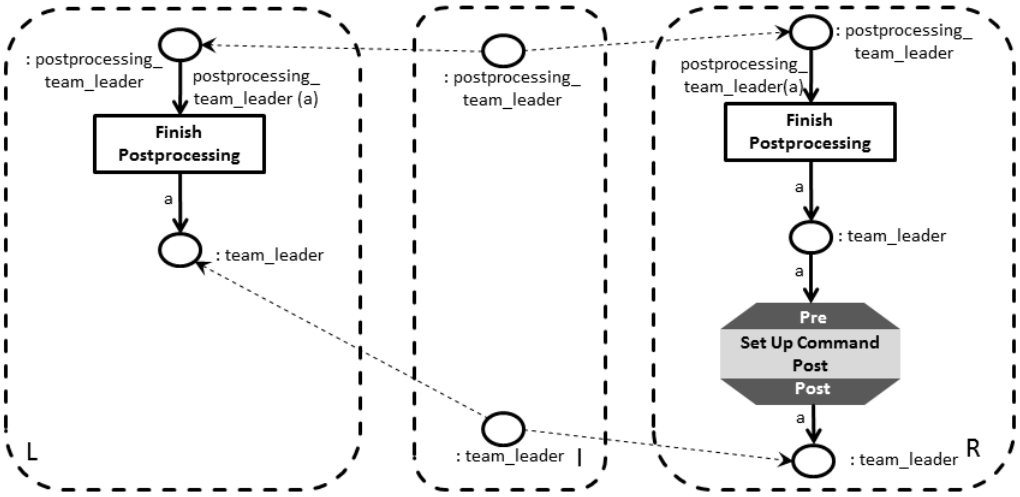


Figure 27: Insertion rule for the task *Set Up Command Post*.

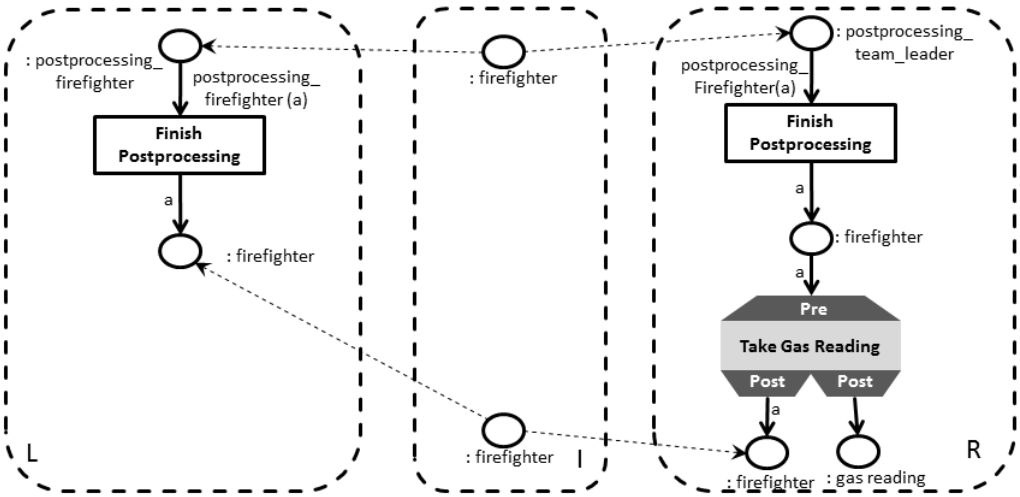


Figure 28: Insertion rule for the task *Take Gas Reading*.

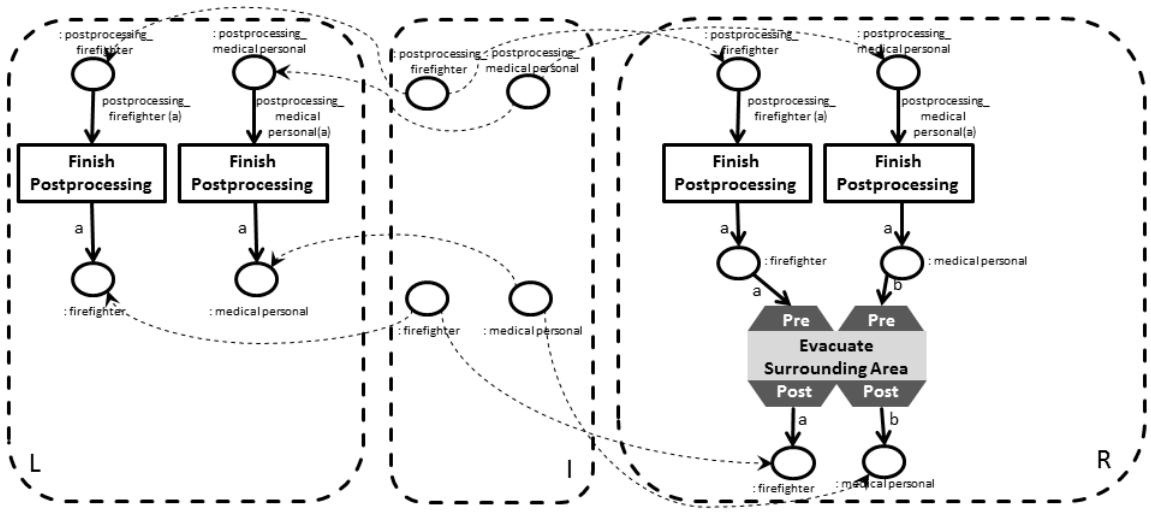


Figure 29: Insertion rule for the task *Evacuate Surrounding Area*.

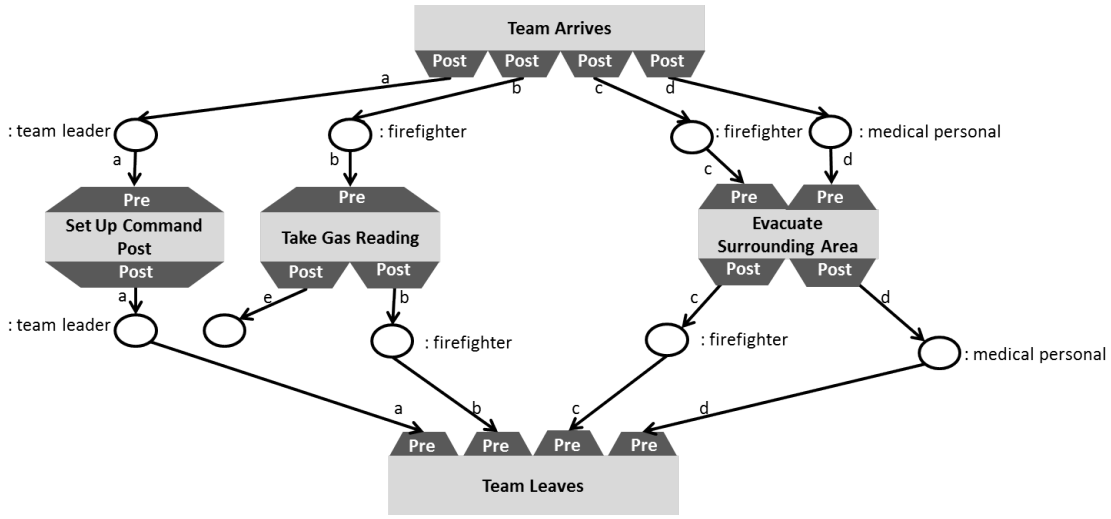


Figure 30: The net after the insertion of the tasks *Set up Command Post*, *Take Gas Reading* and *Evacuate Surrounding Area*.

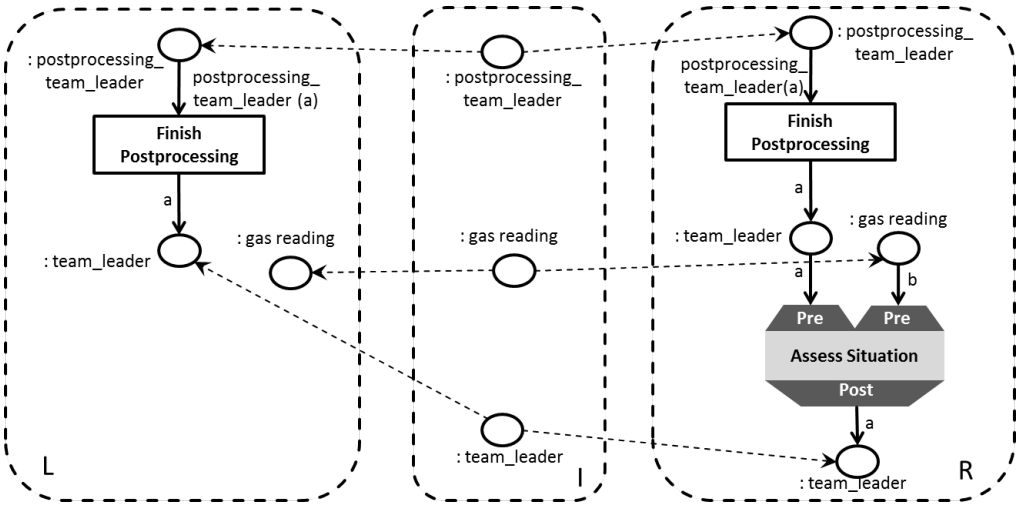


Figure 31: Insertion rule for the task *Assess Situation*.

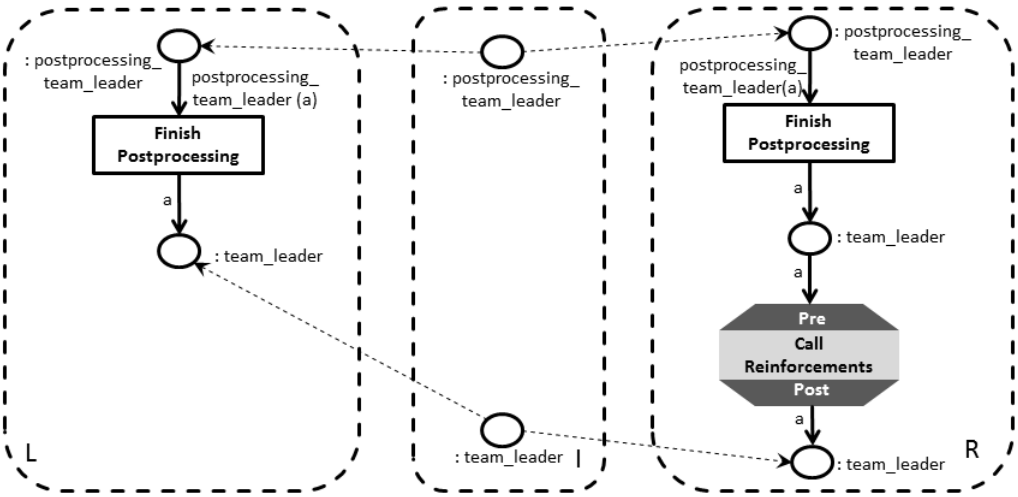


Figure 32: Insertion rule for the task *Call Reinforcements*.

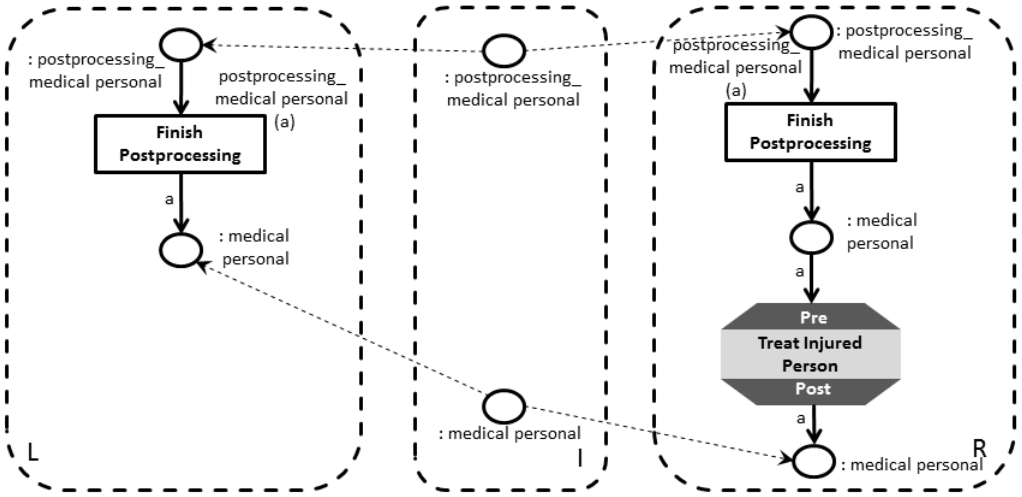


Figure 33: Insertion rule for the task *Treat Injured Person*

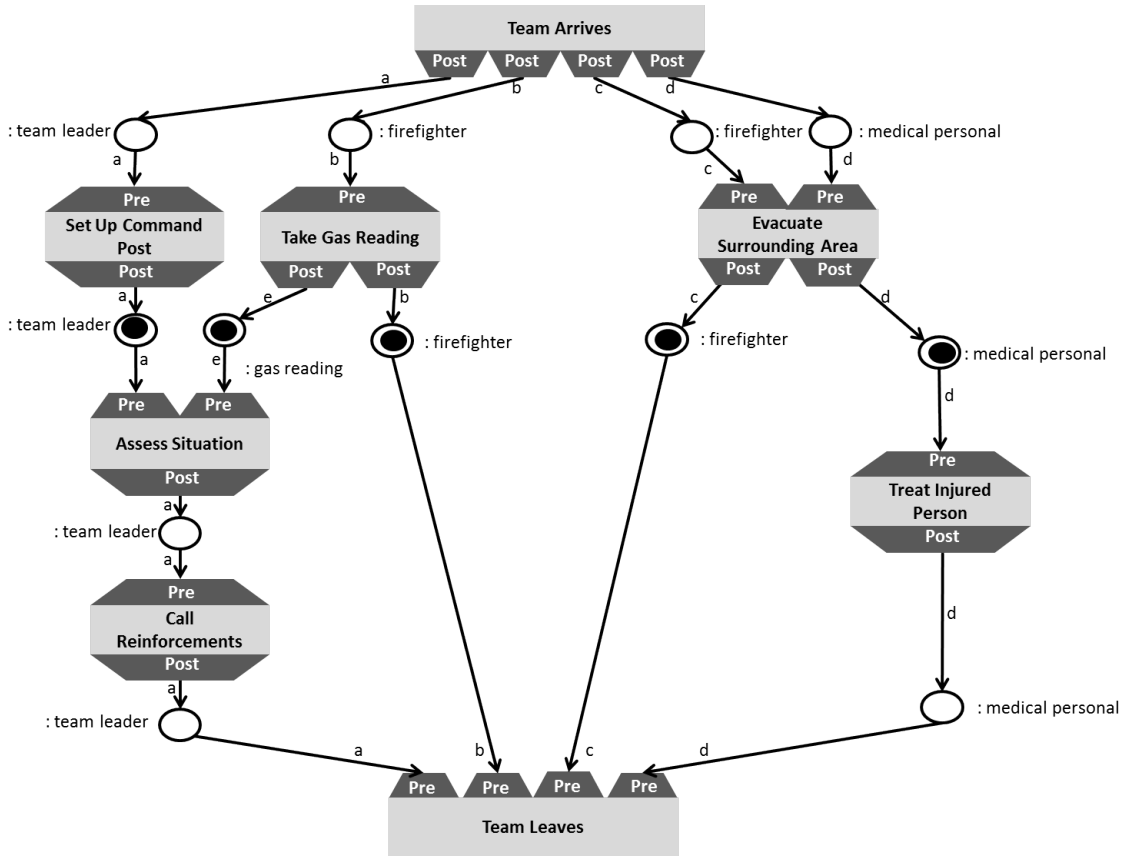


Figure 34: The net after the insertion of the tasks *Assess Situation*, *Call Reinforcements* and *Treat Injured Person*.

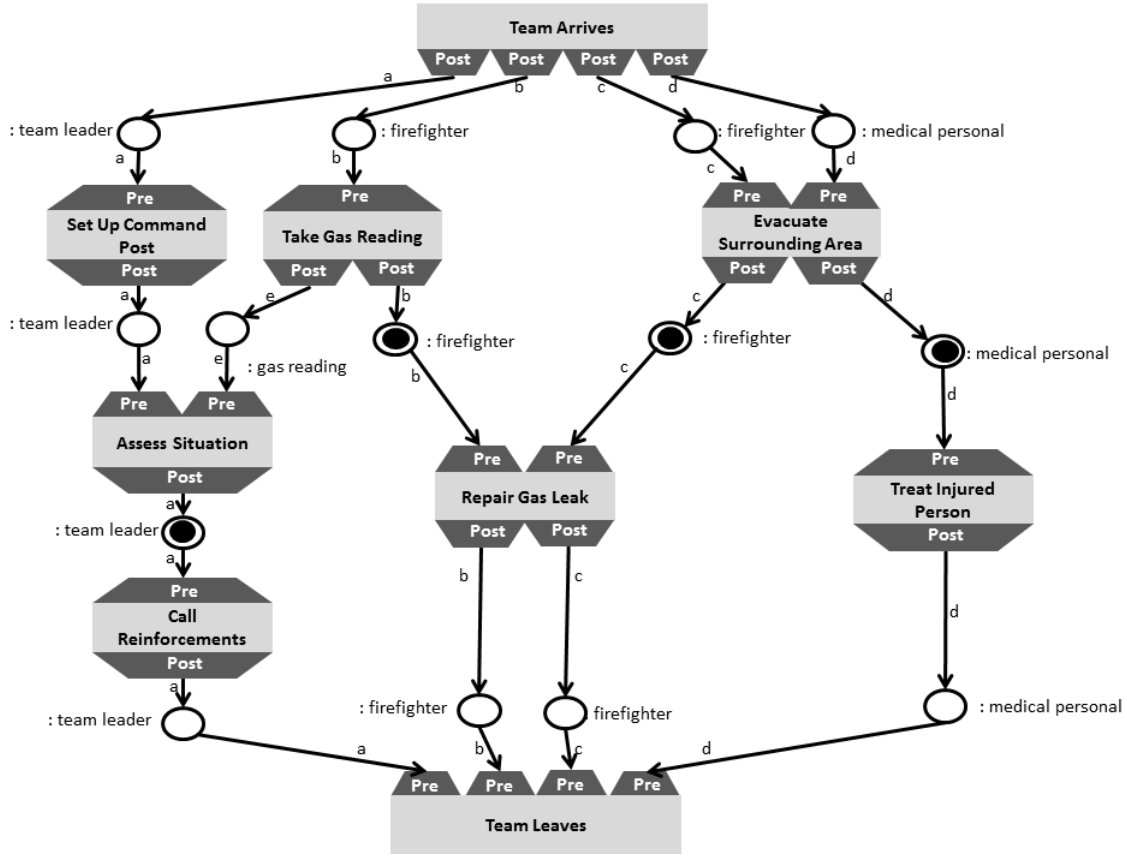


Figure 35: The transformed net in the second case study