# Requirements Analysis and Modeling with Problem Frames and SysML: A Case Study

Pietro Colombo[1], Ferhat Khendek[1], and Luigi Lavazza[2]

[1] Department of Electrical and Computer Engineering, Concordia University
1455, de Maisonneuve W., Montreal, Quebec, Canada H3G 1M8
[2] Dipartimento di Informatica e Comunicazione, Università dell'Insubria
Via Mazzini 5, 21100 Varese – Italy
{colombo,khendek}@encs.concordia.ca,
luigi.lavazza@uninsubria.it

**Abstract.** Requirements analysis based on Problem Frames is getting an increasing attention in the academic community and has the potential to become of relevant interest also for industry. However the approach lacks an adequate notational support and methodological guidelines, and case studies that demonstrate its applicability to problems of realistic complexity are still rare. These weaknesses may hinder its adoption. This paper aims at contributing towards the elimination of these weaknesses. We report on an experience in analyzing and specifying the requirements of a controller for traffic lights of an intersection using Problem Frames in combination with SysML. The analysis was performed by decomposing the problem, addressing the identified sub-problems, and recomposing them while solving the identified interferences. The experience allowed us to identify certain guidelines for decomposition and re-composition patterns.

**Keywords:** Requirements analysis, Problem decomposition, Problem composition, Problem Frames, SysML.

## 1 Introduction

Problem Frames (PFs) [1] are a sound requirement analysis approach that aims at driving the analyst from the phase of problem description, where the characteristics of the problem and its requirements are defined, to the specification of a machine that satisfies the requirements. PFs support both context and structural analysis, i.e., they support both the definition of the characteristics of the problem to be solved, and the decomposition of the original problem into simpler sub-problems.

An important weakness of PFs is the lack of an adequate linguistic support. In fact, PFs are not equipped with a unique and clear way for modeling requirements, the behavioral aspects of problem domains and the specification of the machine. Therefore, analysts have to choose a suitable notation to model both the given domain behavior and the required behavior. In order to address this issue, the authors proposed the integration of the PFs with UML [3, 15] and SysML [2]. The experience illustrated in [3] showed that UML does not support the modeling of requirements at

T. Kühne et al. (Eds.): ECMFA 2010, LNCS 6138, pp. 74–89, 2010.

the correct level of abstraction, while SysML [2] addresses all the weakness of UML. In [2] we discuss the SysML support for the context analysis of basic problems, illustrating how SysML constructs can be used to represent PFs concepts.

In this paper we tackle the structural analysis of a realistic problem, a controller for traffic lights of a four way road intersection, for an early evaluation of an extended approach combining PFs and SysML. Our extended approach is illustrated through a case study involving guidelines and criteria that drive the decomposition of a problem and the re-composition of the resulting sub-problems at requirements and machine specifications level.

This work is a first step towards a systematic and a more formal approach to the decomposition and re-composition mechanisms for handling realistic size problems using PFs and SysML.

The rest of the paper is organized into four sections. Section 2 describes briefly how the SysML notation can support the PFs methodology. Section 3 presents the extended approach through the case study and illustrates its analysis and specification by decomposing the problem into simpler sub-problems according to proposed criteria, analyzing these individually and recomposing their descriptions. Section 4 discusses related work. Section 5 discusses lessons learned, draws some conclusions and presents some ideas for future work.

## 2 Supporting Problem Frames with SysML

PFs can be effectively supported by an external notation like SysML. The complete set of guidelines and motivations that show how the requirements engineering approach could take advantage of the usage of SysML are discussed in [2]. In what follow we briefly summarize some activities of the PFs methodology showing how these can be supported by SysML.

- *Problem analysis*: the problem context is decomposed into domains, and shared phenomena are identified. Domains and phenomena are represented by means of Blocks and defined using Block Definition Diagrams (*bdd*) showing the entities of the problem context.
- *Problem definition*: the domain blocks are instantiated and interconnected using an Internal Block Diagram (*ibd*).
- *Requirements definition*: user requirements and properties associated with domains are defined by means of Requirements diagrams (*req*) and refined by means of Parametric Diagrams (*par*), State Machine Diagrams (*stm*), Sequence Diagrams (*seq*) and Activity Diagrams (*act*).
- *Domain refinement*: domain descriptions are refined using SysML diagrams like *bdd* and *ibd* to support domain decomposition into simpler structures, and *stm*, *act*, *par* and *seq* diagrams to define behaviors.

SysML improves the specification part and the usability of the analysis approach, but does not help in scaling it up as it does not affect aspects such as the decomposition and re-composition of problems.

## 3 Extended Approach Combining Problem Frames and SysML

In this section we introduce the case study and illustrate how its analysis can be effectively supported by an extended approach combining PFs and SysML. Because of lack of space we present only some of the problem diagrams for the static aspects of the problems and some SysML diagrams for the requirements specification.

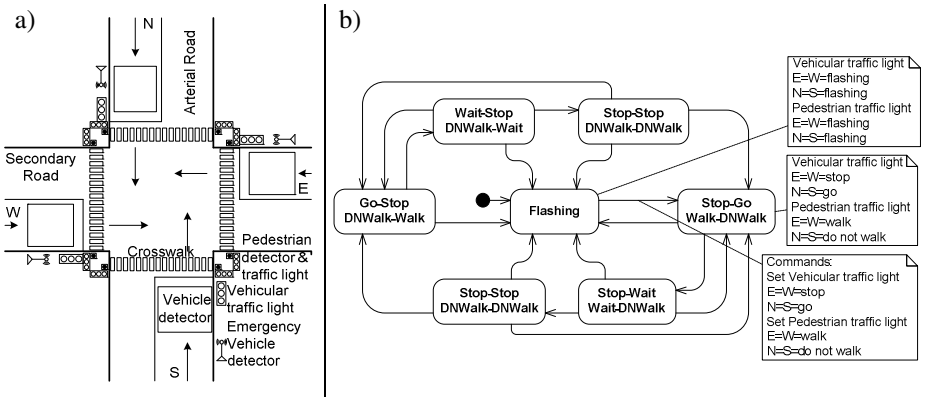


**Fig. 1.** a) The intersection topography, b) The sequence of commands and phases

### 3.1 The Case Study

The intersection controller manages the traffic lights for cars and pedestrian traffic at a four way intersection. The controller reacts to events such as the pressing of a presence button at a crosswalk, vehicles transit, emergency vehicles approaching the intersection, commands issued by an operator via a console, and operates vehicle and pedestrian traffic lights that are positioned next to the intersection.

The controller operates a system composed of two approaches partitioned into distinct semi-approaches: N, S, E, W, each of which is characterized by zebra crossing and is equipped with vehicle and pedestrian traffic lights, vehicle and pedestrian presence detectors, and also detectors that reveal emergency vehicles that are approaching or leaving the intersection (see Fig. 1a). The controller is also provided with a console that allows an operator to configure the operating mode of the controller and to set the states of the traffic lights.

The intersection controller has to operate the states of the traffic lights according to the criteria introduced by the "Semi actuated", "Manual" and "Preempted" operating mode (discussed later). In addition, the intersection controller has to verify that the current state of the traffic lights complies with the last command sent, reacting suitably. The commands sent by the controller cause the change of state of the system. The admissible progress is described (in Fig. 1b) by means of a *stm* as a sequence of commands and "phases" (i.e., combinations of states of the traffic lights). For the sake of simplicity vehicles can only go straight: no lefthand or righthand turns are allowed.

The problem diagram [1] shown in Fig. 2 describes the domain of the problem by presenting 1) the involved machine and problem domains, 2) the phenomena that are

shared by such domains, and 3) the requirements that predicate on the domains' phenomena. The system is composed of the following domains:

- Pedestrian presence detector: A device that forwards to the controller the requests of a pedestrian (i.e., the pressure of the button) to cross a zebra crossing.
- Vehicle presence detector: A device that monitors the presence and passage of vehicles. It generates events indicating that 1) no vehicle is passing for some time, 2) a vehicle is waiting in queue, 3) a vehicle has crossed the sensible area.
- Emergency Vehicle Detector: a device that notifies the intersection controller whenever an emergency vehicle is approaching and when it has crossed the intersection and is moving away.
- Pedestrian Traffic Standard and Vehicular Traffic Standard: a traffic light that receives the *go*, *stop*, *wait* and *flash* events from the controller and sets its lamps. It informs the controller of the current state (i.e., go, stop, wait, flashing), and of configuration errors through continuous flows of data and by means of error codes that specify for instance burn bulbs.
- Manual Override: this device receives requests from an operator to set the phases of the system and to set the operating mode of the controller, and sends such requests to the intersection controller.

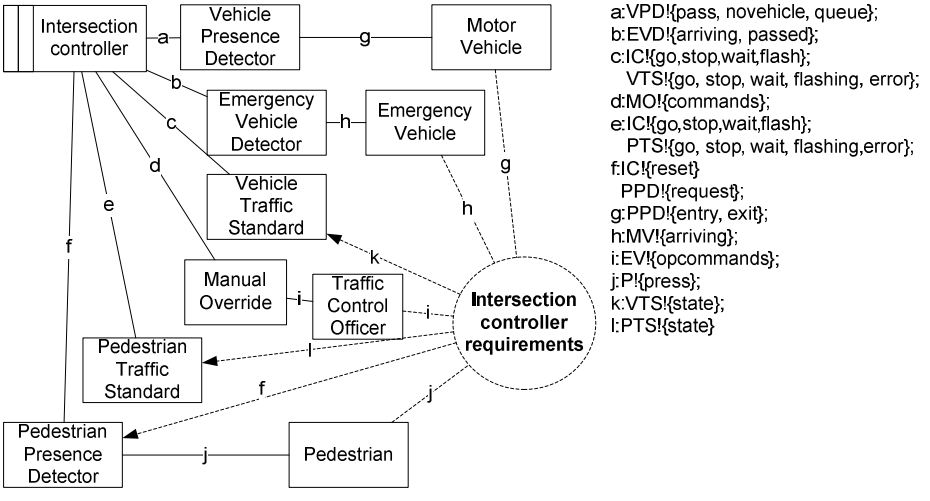


**Fig. 2.** The problem diagram describing the domain of the problem

### 3.2 Decomposition of the Problem

The problem is too complex to be analyzed and modeled as a basic frame. The key to mitigate this complexity is decomposition.

Decomposition is not only an approach to the solution of a problem, but also a process that helps the analyst understanding and analyzing the problem itself [1]. Decomposition aims at projecting original large problem into simpler and smaller

sub-problems in a recursive manner, until sub-problems that fit basic problem frames are identified. The solution of each sub-problem will contribute to the solution of the whole problem.

Given a certain problem, it may be decomposed in different ways. Decomposition criteria usually depend on the characteristics of the problem and on the knowledge and experience of the analyst. The aim is to generate sub-problems whose description and solution is as simple as possible, and that are also simple to re-compose.

We propose to apply a general decomposition criterion based on the identification of the sequential and parallel activities executed in the context of the original problem. At any given time, the controller can operate a single operating mode. The controller operates the traffic lights alternating a sequence of different operating modes depending on specific conditions. Other activities –like monitoring the state of the traffic lights– are performed in parallel with the control.

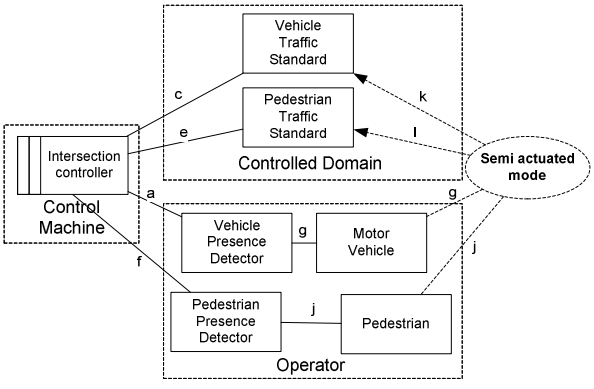


**Fig. 3.** Semi-actuated mode problem diagram

### 3.2.1 Semi Actuated Mode

In this operating mode the approaches at the intersection have different priorities. Vehicle detectors in the minor street (EW) notify the presence of vehicles to the controller. The intersection controller must operate the traffic lights so that the minor approach can receive a green light only when traffic is present. When no traffic is on the minor street or a timeout expires, the green lights of the main road (NS) have to be switched on again. Requests of the pedestrians to anticipate the Walk signal may shorten the duration of the waiting phase.As described in Fig. 3, this sub-problem can be modeled as a Commanded Behavior frame [1] that contains a projection of the domains of the original problem. The requirements constrain the state of the traffic lights according to the requests issued by pedestrians and by the presence of vehicles. Transitions triggered by the operator when exiting (entering) the state *Flashing* specify when the traffic light are turned on (off). Transitions triggered by the "after" clause fire when a minimum timeout is expired. More specifically, *WaitD* and *SafetyD* timeouts are imposed for safety reasons, while *GoD* depends on the priority of the approach.

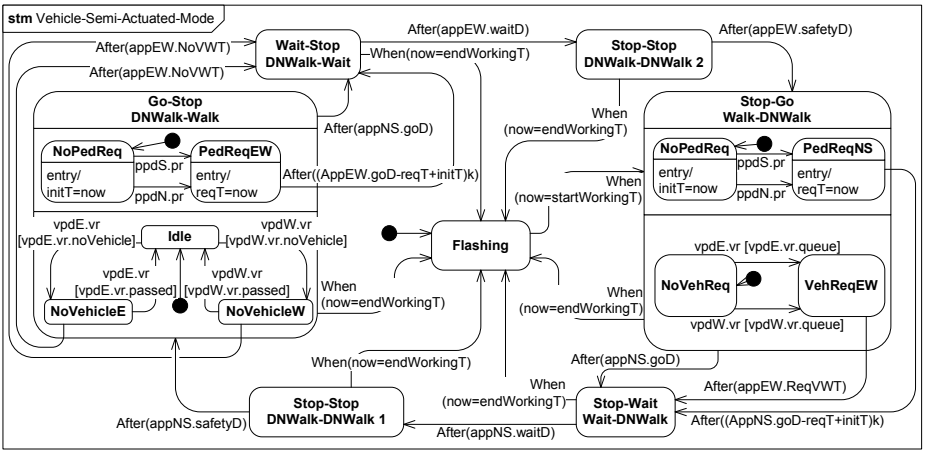


**Fig. 4.** The requirements of the Semi-actuated mode problem

The duration in the states with the "Go" signal (composite states) can be less than *GoD* when a pedestrian with a "Do not walk" signal requests to cross the street, or there are vehicles that are waiting for the "Go" signal on the secondary approach, or no vehicle has passed on the secondary approach for a given time.

The machine specification, i.e. the behavior of the intersection controller that satisfies the requirements of the sub-problem, is described by means of a *stm* diagram that features the same set of states and transitions as the one used to describe the requirements. Transitions have the same triggering conditions, but –unlike in Fig. 4– they generate "commands" to change the state of the traffic lights. The *stm* is not shown here due to lack of space.

### 3.2.2 Manual Mode

In this operating mode an operator controls the evolution of the phases. He/she interacts with a dedicated console determining the length of the phases characterized by the "Go" signal, and establishing which will be the next phase. As in the previous operating mode, the transitions between some phases are constrained by safety timeouts and cannot be shortened or overridden by the commands of the operator.

Also this sub-problem is an instance of the Commanded Behavior Frame (Fig. 5). The requirements of the sub-problem are described by means of the *stm* diagram of Fig. 6. For instance, if the current phase is *Stop-Wait Wait-DNWalk* and an emergency vehicle is approaching on the street that has the "Wait" signal, the operator can impose the "Go" signal setting the *Stop-Go Walk-DNWalk* phase.

The specification of the machine could be provided by using a *stm* (not reported here for space reasons) similar to the one that presents the requirements. Like in the Semi-actuated mode, also in this case the only differences between the *stms* concern the signals that are generated at transition firing time.

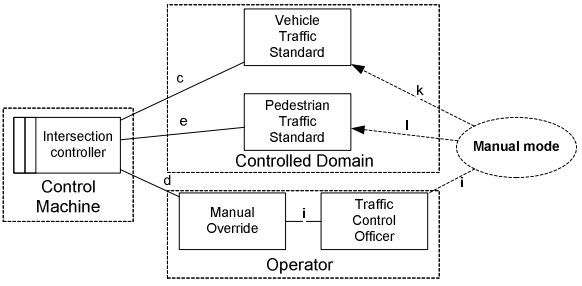


**Fig. 5.** Manual mode problem diagram

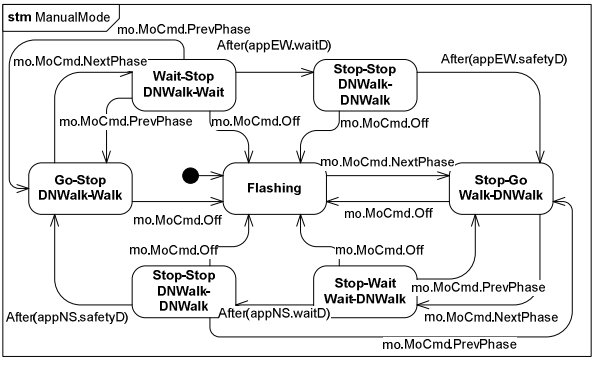


**Fig. 6.** Manual mode problem requirements

### 3.2.3 Preempted Mode

In this operating mode the traffic lights are operated in a way such that the roads where emergency vehicles are approaching get the "Go" signal as soon as possible. Emergency detectors determine the presence (and the direction) of emergency vehicles.

The controller has to switch to "Stop" all the traffic lights with the exception of the ones of the approach where the vehicle that triggered the preemption sequence is arriving.

The preempted sequence terminates when the detectors inform the controller that the emergency vehicle has crossed the intersection. Whenever the preempted sequence is enabled, all the requests from the pedestrians and the signals from the vehicle detectors are ignored. The problem is characterized by a Commanded Behavior Frame (not shown for lack of space). The requirements of the problem, described using two parallel state machines, are shown in Fig. 7. The first machine keeps track of the presence of emergency vehicles, while the second one, depending on the current phase, establishes the preempted sequence.

The evolution of the machines is synchronized by the shared variables *EVNS* and *EVEW*, which keep track of emergency vehicles on a given approach, and by events *startEmergency* and *stopEmergency*, which specify when the preempted sequence starts and stops, respectively. The first *stm* introduces four states that depict the

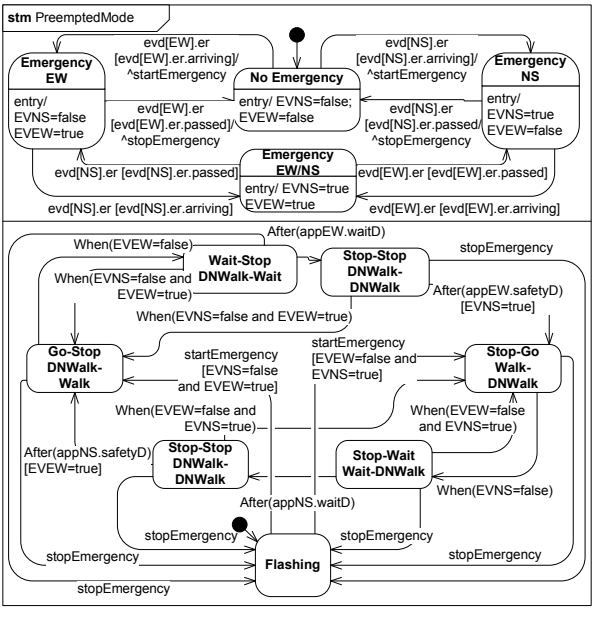


**Fig. 7.** Preempted mode requirements

following situations: no emergency, emergency on the approach EW, on the NW one, or on both the approaches. The previously mentioned shared variables are updated depending on the current state, while the events are generated when the state *NoEmergency* is entered and exited.

The second *stm* describes the preempted sequence. Whenever an emergency starts (notified by the corresponding event), a new phase that is determined according to the position of the emergency vehicle (shared variables *EVEW* and *EVNS*) is enabled.

As soon as all the emergency vehicles have crossed the intersection the preempted sequence is terminated by returning to the initial state.

The machine specification is not significantly different from the requirements and is not shown due to lack of space.

### 3.2.4 Traffic Light State Check

The intersection controller has to check whether the current state of the traffic lights is the one expected as a result of the received commands. The control is mediated by the local controllers of the traffic lights that in case of malfunction inform the intersection controller of the error type. When a malfunction that may compromise the safety of the system is reported, the controller has to impose the phase *Flashing*. When minor errors are detected, a description of the problem must be logged.

The problem is characterized by two distinct requirements with respect to the monitoring of the traffic light conditions and the control of their state, respectively. Hence, the problem is decomposed into two distinct parallel sub-problems: an Information display problem [1], which deals with monitoring the states of the traffic lights and a Controlled behavior problem, which defines the reaction of the controller.

The auditing sub-problem is illustrated in the Problem diagram of Fig. 8.

The requirements are defined using the *act* diagram shown in Fig. 9. It defines an action named *CheckTL* that takes as input, as a continuous flow, the information on the traffic lights state and compares them with the last issued commands. Safety threatening misbehaviors cause *Alarm* events to be generated; minor problems are logged. The machine specification is given as an act diagram equivalent to the one used to define the requirements, but it is not shown here due to lack of space.

The second sub-problem, described by means of the problem diagram shown in Fig. 10, defines the effects of the signal *Alarm*. The requirements for the problem are defined using a state machine diagram composed of states representing the phases of the system, and transitions, triggered by the event *Alarm*, allowing the passage from each phase to the *Flashing* one. The requirements *stm* is similar to the ones that illustrate the operating modes and is not reported for space reasons. Also the machine specification is not shown since substantially equivalent to the requirements.

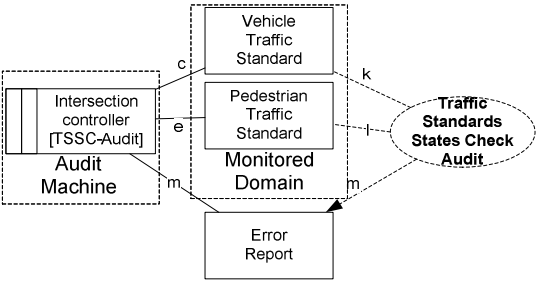


**Fig. 8.** Misbehaviors auditing sub-problem

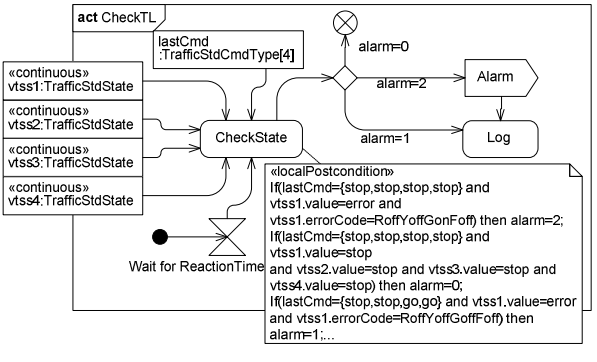


**Fig. 9.** Monitoring the states

### 3.3 Composition of Problems

Once the sub-problems have been identified, analyzed and a machine specification has been provided for each of them, their descriptions have to be recomposed. The goal is to define a unique machine specification that, once connected to the involved problem domains, satisfies both the requirements of each sub-problem and additional

constraints that aim at addressing possible inconsistencies among the sub-problems. The analyst has to take into account the relationships between the sub-problems and how they overlap and interact. Such interactions occur whenever multiple sub-problems share domains and phenomena.

The correctness of the composition is the subject of composition concerns, which deal with conflicts and interferences that may affect both *indicative* and *optative* descriptions [1]. The concerns aim at addressing conflicts and interferences by introducing priority criteria, mutual exclusion and scheduling mechanisms.

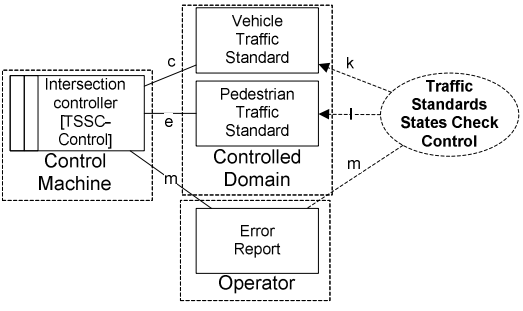


**Fig. 10.** Reacting to misbehaviors

Composition strategies are subject to the choices operated at decomposition time, hence, in what follow we propose two patterns that guide the composition of sequential and parallel problems, respectively.

### 3.3.1 Composition of Sequential Problems

The intersection controller operates in a single operating mode at a time. In the previous sections we have identified and described the single sub-problems representing the operating modes of the controller. We have also provided a machine specification for each sub-problem. The composition of these sub-problems requires coordinating the transition between operating modes by synchronizing their machines.

We propose to tackle the problem by means of a Composition Frame [4]. We define the composition by means of a new problem, named Operating Mode Coordinator (OMC), illustrated in the Problem diagram of Fig. 11.

The problem is characterized by the union of problem and machine domains of the sub-problems describing the operating modes. According to the Composition Frame, the machines of the sub-problems, labeled with the initial letter of the sub-problems' name ([SA] for Semi Actuated Mode, [MM] for Manual Mode, [PM] for Preempted mode) are considered given domains [1]. No phenomenon is shared by the machines of the sub-problems and by any other problem domains. According to the Composition frame, we introduce also a new machine, labeled OMC, which filters the connections between the machines of the sub-problems and the problem domains, specifying an interface composed of the union of all the existing shared phenomena. The OMC machine has to behave in a way such that it satisfies the coordination requirements by interacting with the other domains and machines.

The application of the Composition frame suggests a way to organize the composition of the problem, but it does not drive the resolution of the interferences between sub-problems. Interferences depend on the characteristics of the specific sub-problems to be composed, and must be addressed according to priority, mutual exclusion and scheduling principles defined by new requirements. In our case, such requirements can be formulated by considering the pre-conditions that enable an operating mode. Both the requirements and machine specifications of the sub-problems describing the operating modes define *Flashing* as initial phase.

However, the system can be in a different state when a mode change occurs, and we have to assure that the change of mode does not cause losing the current phase. In fact, the new mode has to start by resuming the current phase of the previous operating mode. This requirement is particularly relevant in case of emergency. Suppose that the emergency vehicle is arriving on an approach that has already the "Go" signal: the change of mode has to cause no change of the current phase. The new requirements, reported in Fig. 12, also constrain how and when it is possible to change the operating mode: in case the system is operating in semi-actuated mode and an emergency vehicle is approaching the intersection, it is required to automatically enable the Preempted mode.

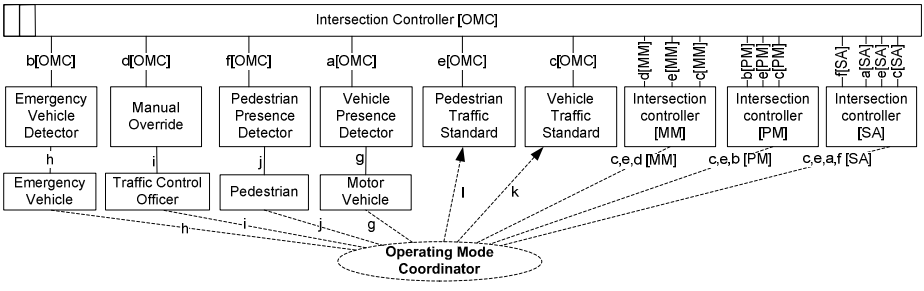


**Fig. 11.** OMC Problem diagram

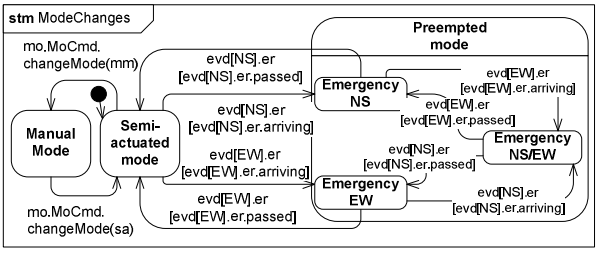


**Fig. 12.** Mode changes requirements

The last phase before enabling the preempted mode becomes the first phase of the preempted sequence. When the emergency is over, the controller has to return to the previous operating mode. The evolution of the stm is determined by events controlled by domains and observed by the machine of the OMC problem. These events are

forwarded to the machines of the different sub-problems, which, in turn, generate commands that trigger the transitions between phases. Fig. 13 shows a small portion of the resulting stm, which considers the transition from the phase Stop-Go Walk-DNWalk to Stop-Wait Wait-DNWalk. Notice that the events "NextPhase", "Pr", and "Vr" generated by the console, pedestrian or vehicle detectors are forwarded to the machines of the sub-problems, which react generating signals that may enable a new phase. The machine specification differs from the requirements only for the commands to set the state of the traffic lights and it is not illustrated here because of lack of space.

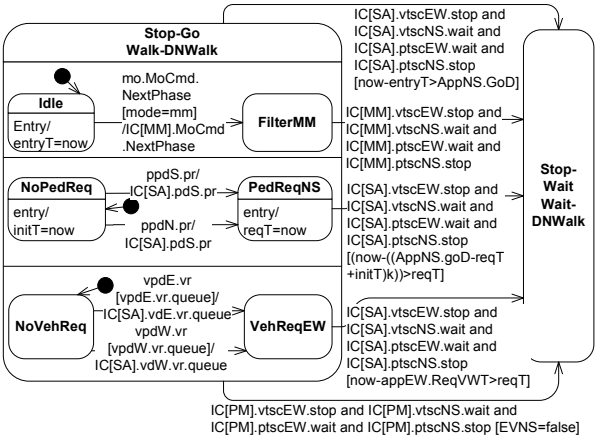


**Fig. 13.** A view on the *stm* of the requirements

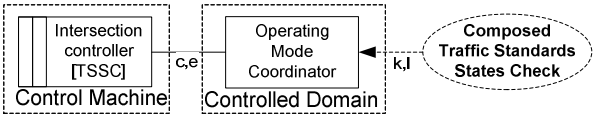


**Fig. 14.** TSSC Problem diagram

### 3.3.2 Composition of Parallel Problems

Once the sequential sub-problems have been re-composed, we can consider the interaction with the parallel sub-problems. We need to define the nature of the interaction, establishing whether parallel sub-problems may trigger behaviors described in other sub-problems and addressing possible interferences and conflicts. The State check problem is active in every operating mode.

A preliminary description of the problem –considered as "standalone"– has been given in Section 3.2.4. Now we have to consider its integration with the OMC one.

Hence, we introduce a new problem (illustrated in Fig. 14) characterized by a new integration machine -named Intersection Controller [TSSC]- and a new domain -named Operating Mode Coordinator- that contains the problem domains of the standalone TSSC sub-problems, and all the domains of the OMC (i.e., those that appear in

Fig. 11). This new integrated problem is modeled as a Required Behavior Frame (Fig. 14). The machine of the new problem constrains the connections of the OMC sub-machine with the traffic lights domains, by setting the phase *Flashing* whenever an event *Alarm* is generated. The new machine is characterized by the same interface as the machines of the "standalone" sub-problems TSSC-Audit and TSSC-Control (see Fig. 8 and Fig. 10).

The requirements of the composed problem specify the union of the required properties illustrated in the sub-problems OMC, TSSC audit and TSSC Control, and also additional constraints that address interferences among these sub-problems. Interferences may exist whenever different domains control the same phenomena, for instance, between the sub-problems TSSC Control, which describes the reaction of the controller in case of a malfunction, and OMC, which describes the coordination of the operating modes. For instance, an interference occurs when a serious error concerning the current state of the traffic lights is detected by the controller, which would react by setting the phase *Flashing*, and at the same time an emergency vehicle approaches the intersection, thus causing the intersection controller to switch to the Preempted mode and to force the transition to the phase that favors the crossing of the emergency vehicle: the two target phases are different!

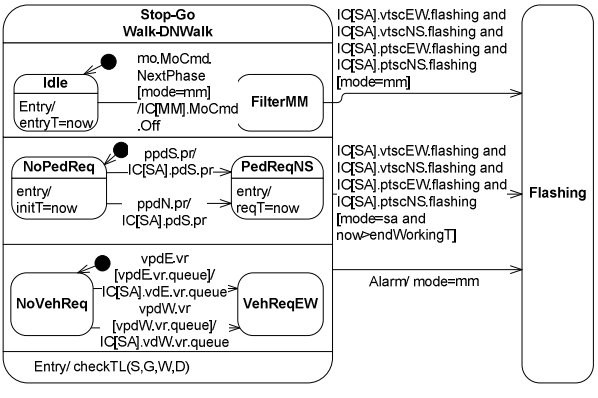


**Fig. 15.** A view on the stm of the requirements

Hence, whenever an event *Alarm* is generated, it is required to set the Manual operating mode, thus preventing the automatic change of mode and phase.

The requirements of the problem are described by means of the *act* diagram that illustrates the requirements of the TSSC Auditing problem (see Fig. 9), and a *stm* that extends the one used for describing the requirements of the OMC problem. Such a *stm* integrates the transitions of the *stm* of the TSSC Control problem, and also the management of the monitoring activity and the change of mode. Fig. 15 presents a portion of the resulting *stm* which shows that whenever a phase is enabled (and during the whole phase), the current state of the traffic lights is monitored by the action *CheckTL*. The phase *Flashing* can be reached from each phase through transitions that set the Manual mode once triggered by the event *Alarm*. Since the machine specification is not substantially different from the requirements it is not shown here.

## 4 Related Work

In the literature, requirements analysis and specification approaches based on PFs are usually applied to simple case studies. Only a few papers tackle their applicability to more or less realistic case studies. As a consequence, there is no evidence of the scalability of the analysis approaches. Among these few works, [11] shows the application of PFs to the modeling of the requirements of an online financial system. The same case study has been revisited in [12], describing different decomposition criteria that brought to the identification of different sub-problems and to the proposal of new basic frames. In [10] PFs are used for the analysis of a geographic application. [6] describes the usage of PFs to model the requirements of a system monitoring the transportation of dangerous goods. All the aforementioned works propose decomposition criteria that depend on the characteristics of the analyzed problems, and that cannot be easily abstracted to derive general criteria. On the contrary, the work reported in this paper introduces patterns that can be applied to different problems and domains.

Only a few research works propose (de)composition techniques at problem level. KAOS [7] and the NFR framework [8], together with PFs, are the techniques focusing on problems instead of solutions. However, both KAOS and NFR are goal based approaches and do not explicitly focus on domain properties.

Composition issues have been addressed by several aspect-based approaches [9], but they essentially focus on design and implementation issues. [13] is one among the few aspect oriented papers that address inconsistencies and conflicts among non functional requirements, but it does not address the decomposition of requirements.

Most of the research works found in the literature focus on formal techniques to merge behaviors or to deal with inconsistencies at model level [13]. Although these works focus on the solution space, some of the proposed techniques could be combined with PFs by redefining their scope at problem level. For instance, in [5] a formal approach for composing system behaviors defined as finite state automata is proposed. Such a technique could be adapted to support a (semi-) automatic composition of machine specifications defined using state machines.

## 5 Discussion and Lessons Learned

The work reported in this paper contributes to the evaluation of the applicability of PFs to case studies of realistic complexity. The experience has shown that PFs can help achieving a complete and clear comprehension of the requirements of the system. It has also confirmed that PFs can benefit from the modeling support of SysML. Although some of the benefits may not be evident from the few diagrams shown in the paper because of the lack of space, SysML modeled well both the structural and behavioral characteristics of the problem domains. It effectively supported the modeling of domains that represent non-pure SW components and the modeling of continuous behaviors (e.g., the monitoring of Fig. 9). SysML is also very promising with respect to scalability. It provides a full support for the decomposition and the projection of domains and phenomena using *bdd* and *ibd* diagrams, views and allocation mechanisms.

The modeling effort would range reasonable to small: PFs mitigated the effort by favoring the application of an incremental process. For instance, the modeling of the recomposed problem machines reused several parts of the specifications of the starting sub-problems. In order to further ease the re-composition we are studying techniques that allow automating the composition of activities like the merging of behaviors described with *stm* diagrams both at requirements and machine specification level.

All the identified frames were easily modeled; diagrams that expressed requirements, specifications and problem domains characteristics at the right level of abstraction and with an intuitive and expressive notation were easily created.

The case study also showed that the decomposition driven by the identification of parallel and sequential sub-problems mitigated the complexity of the original problem. However, such criteria do not help addressing a typical PFs based analysis: the same decomposition choices applied to the requested functionalities or to the machine responsibilities may results in different problems. The patterns facing the re-composition helped in addressing inconsistencies between the sub-problems. However, they represent one possible solution to the (de)composition. We need to identify multiple guidelines to drive the analysis and to refine and generalize them through a thorough analysis of several case studies.

We retain that both the analysis and the modeling could be further facilitated by general analysis guidelines to use PFs, and also by ready to use modeling elements representing frames and other basic concepts. Such elements are the base of a framework for the analysis of complex problems. This experience helped us identifying research directions for the framework definition. In order to better support the modeling we are working towards the definition of a SysML profile for PFs. We have also defined a meta-model for PFs and a tool, built on such a meta-model. This tool supports the analysis of problems using PFs [15]. We are planning to enrich this tool with a transformation engine capable of generating skeletons models of the SysML profile for PFs. The predefined guidelines will ease and speed up the learning of the approach by practioners in requirements analysis and UML modeling. The proposed framework may ease and speed up the adoption of PFs by practioners.

## References

1. Jackson, M.: Problem Frames - Analysing and Structuring Software Development Problems. Addison-Wesley/ACM Press (2001)
2. Colombo, P., Del Bianco, V., Lavazza, L., Coen-Porisini, A.: A methodological framework for SysML: a Problem Frames-based approach. In: 14th Asia-Pacific Software Engineering Conference, APSEC 2007 (2007)
3. Lavazza, L., Del Bianco, V.: Combining Problem Frames and UML in the Description of Software Requirements. In: Baresi, L., Heckel, R. (eds.) FASE 2006. LNCS, vol. 3922, pp. 199–213. Springer, Heidelberg (2006)
4. Laney, R., Barroca, R., Jackson, M., Nuseibeh, B.: Composing Requirements Using Problem Frames. In: Int. Conf. on Requirements Engineering, RE 2004 (2004)
5. Mizouni, R., Salah, A., Kolahi, S., Dssouli, R.: Merging partial system behaviours: composition of use-case automata. Software, IET 1(4) (2007)

6. Colombo, P., del Bianco, V., Lavazza, L.: Using Problem Frames to Model the Requirements of a System for Monitoring Dangerous Goods Transportation. In: 3rd Int. Work. on Advances and Applications of Problem Frames, IWAAPF 2008 (2008)
7. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: 5th International Symposium on Requirements Engineering, RE 2001 (2001)
8. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Dordrecht (2000)
9. Elrad, T., Filman, R., Bader, A. (Guest eds.): Special Issue on Aspect Oriented Programming. Communications of the ACM 44(10) (2001)
10. Nelson, M., Alencar, P., Cowan, P.: Informal description and analysis of geographic requirements: an approach based on problems. SoSyM 6(3) (2007)
11. Cox, K., Phalp, K.: From process model to problem frame - a position paper. In: Int. Work. on Requirements Engineering: Foundation for Software Quality, REFSQ 2003 (2003)
12. Cox, K., Phalp, K., Bleistein, S., Verner, J.: Deriving requirements from process models via the Problem Frames approach. Information and Software Technology 47(5), 319–337 (2005)
13. Rashid, A., Moreira, A.M.D., Araujo, J.: Modularisation and Composition of Aspectual Requirements. In: AOSD 2003 (2003)
14. Colombo, P., del Bianco, V., Lavazza, L., Coen-Porisini, A.: Towards a Meta-model for Problem Frames: Conceptual Issues and Tool Building Support. In: 4th Int. Conf. on Software Engineering Advances, ICSEA 2009 (2009)
15. Lavazza, L., Del Bianco, V.: A UML-based Approach for Representing Problem Frames. In: 1st International Workshop on Advances and Applications of Problem Frames (IWAAPF), an ICSE 2004 Workshop, Edinburgh, May 24 (2004)