Structured Analysis of Interactions in Collaborative Environments

Elena Troubitsyna
Åbo Akademi University, Dept. of IT
Joukhaisenkatu 3-5A, 20520,
Turku, Finland
e-mail: Elena.Troubitsyna@abo.fi

***Abstract*— Collaborative computing environments are dynamic compositions of communicating components that interact with each other to achieve a common goals. The collaborations continuously reconfigure to achieve the required goals. Ensuring correctness of complex component interactions is cumbersome and requires structuring techniques that allow us to model and analyse component interactions in a systematic way. In this paper, we propose a set of modelling abstractions that allow us to define component interactions in dynamic collaborative environments. We propose a structured approach to analysing possible deviations in the component interactions based on HAZOP – Hazard and Operability Study -- and formally define the impact of deviations in component interactions on achieving the required goals.**

Keywords-*dynamic collaborations; interactions; goals; deviation analysis; formal modelling.*

I. Introduction

Over the recent years, collaboration has became a one of the primarily engines to create new services, achieve higher productivity or enable creating novel applications. Increasing openness of software and advances in networking has led to a proliferation of collaborative computing environments in different domains. Among the most remarkable examples of collaborative environments is the Internet of Things [10]. The term is introduced to stress the growing outreach of connectivity towards sensors, machines and variety of appliances. The wide-spreading use of the collaborative approach amplifies the need for novel communication paradigm that enables dynamic flexible collaboration creation and function.

The inherently dynamic mode of collaborations requires novel approaches that allow the designers systematically analyse the dynamics of collaborative environments and in particular, predict how deviations in the component behaviour and interactions impact objectives that a collaborative environment should achieve.

It has been recognized that it is convenient to formalize objectives that a system should achieve by a notion of *goals* [4]. The collaborations are formed to achieve certain goals. The components forming collaboration provide certain individual functionality that contributes to achieving overall goal. When a component fails or components communicate inappropriately, collaboration might fail to achieve the required goal. Therefore, we should analyse the possible deviations in the component behaviour and formally define the impact of these deviations on achieving overall goals.

To systematically study possible deviations in the component interactions we propose to use Hazard and Operability Studies -- the HAZOP method [1,2]. We define the main types of deviations in the components interactions and define their impact on achieving system goals.

We believe that the main contribution of this paper, i.e., a formal link between goals and possible deviations in component interactions, can potentially facilitate design of complex collaborative environments

The paper is structured as follows. In Section II, we define collaborative environments in terms of the goals that should be achieved. In Section III, we describe generic scenarios of component interactions. Section IV shows how to systematically analyse deviations in the component interactions using HAZOP. Finally, in Section V, we discuss the proposed approach and overview the related work.

II. Goals in Collaborative Environments

In this paper, we define a *collaborative environment* as a set of collaborations, i.e.,

$$ColENV = \{C_1, C_2, \dots, C_N\}$$

where Ci is an id of collaboration.

Collaboration is a dynamic composition of components. The components join and leave collaboration depending on the current goal and the states of the components. In general, any collaboration is formed to fulfil a certain goal [4]. The set of goals, which an entire collaborative environment can achieve, is denoted as *GOALS:*

$$GOALS = \{G_1, G_2, \dots, G_M\}$$

The set consists of the constants defining the names of the goals. We assume that each particular collaboration is formed to perform a certain goal from the set *GOALS.*

A *goal* is an objective that collaboration should achieve. A goal can be decomposed into a set of subgoals, and furthermore, into a set of sub-subgoals that each component in the collaboration should perform. Each component carries a special attribute describing the functionality that it implements. Often these attributes are called roles. Usually a component implements a set of roles chosen according to the tasks that it should perform in each particular collaboration.

 ISBN: 978-1-61208-287-5

The goal that collaboration should implement defines how many components executing each role collaboration should have to achieve a goal. Therefore, a configuration can be defined as follows:

$Config \in CONFIG$, where $CONFIG : ROLES \rightarrow N$

where *ROLES* is a set of roles and *N* is a set of natural numbers.

Often a configuration of collaboration is defined not only by a goal but also non-functional parameters, e.g., performance. We assume that goals are distinct if their non-functional parameters are different. Therefore, we can unambiguously map a set of goals on the set of configurations.

For each goal Gi, $Gi \in GOALS$, we can define the minimal sufficient configuration as a function

$MINCONF : GOALS \rightarrow CONFIG$

The function defines how many components in each role collaboration should have to be able to achieve a certain goal. The function *MINCONF* defines the minimal necessary conditions. Obviously, collaboration can have more components that might be inactive while achieving a certain goal or used as standby to implement fault tolerance in case some components fail.

In practice, at each particular moment of time, a collaborative environment *ColENV* does not try to achieve all the goals defined by the set *GOALS* at once. Therefore, we can distinguish between a set of the active (trigged) goals, i.e., the goals that a collaborative environment tries to achieve at a certain moment of time and the goals that are not trigged. This defines a partitioning of the set of goals into two non-intersecting subsets:

$GOALS = ACT\_G \cup PAS\_G$,
where $ACT\_G \cap PAS\_G = \emptyset$

In our modelling, we assume that the components are not kept idle but rather are getting engaged in different collaborations (as soon as their roles match the roles required in the collaboration). Therefore, when a goal is trigged, it might be the case that the conditions defined by *MINCONF* are not satisfied because the required components are still engaged in some other collaboration. If the required configuration is established then the collaboration executes the required actions to achieve the goal. We introduce a set

$C\_STATE : \{Active, Activated, Dormant\}$

to designate the status of the collaboration and introduce the function C_STATUS that maps the id of the collaboration to its status:

$C\_STATUS : CNAME \rightarrow C\_STATE$

The function *CUR_CONFIG* is defined as follows:

$$CUR\_CONFIG : CNAME \rightarrow CONFIG$$

It designates the current configuration of the collaboration.

Next, we formally define the relationships between the status of the collaboration, goals and configurations.

The collaboration *Ci* is active, i.e.,

$$C\_STATUS\ (Ci) = Active$$

if

$Gj \in GOALS \wedge$
$Gj \in Act\_G \wedge$
$MINCONF(Gi) \leq CUR\_CONFIG(Ci)$

where the ordering relation $\leq$ is defined over the configurations as follows:

For $Conf_k$ and $Conf_l$, such that $Conf_k$, $\mathrm{Conf}_l \in CONFIG$, $Conf_k \leq Conf_k$ if

$$\forall r_n . r_n \in dom(\mathrm{Conf}_k) \Rightarrow r_n \in dom(Conf_l)$$
$$\forall r_n . r_n \in dom(\mathrm{Conf}_k) \Rightarrow Conf_k(r_n) \leq Conf_l(r_n)$$

When a collaboration *Ci* is set to achieve a certain goal but has not established the required configuration or an execution of a scenario required to achieve a goal is suspended due to failures, its status is *Activated, i.e.,*

$C\_STATUS\ (Ci) = Activated$

if

$Gj \in GOALS \wedge$
$Gj \in Act\_G \wedge$
$\neg (MINCONF(Gi) \leq CUR\_CONFIG(Ci))$

Finally, collaboration can be inactive, i.e.,

$C\_STATUS\ (Ci) = Activated$

if

$Gj \in GOALS \wedge$
$Gj \in Pas\_G$

We assume that components are involved in the collaboration with the status *ACTIVE* communicate with each other by exchanging messages. To achieve a certain goal, collaboration should perform a predefined scenario. In the next section, we define generic scenarios performed by the components in collaboration.

III. MODELLING COMPONENT INTERACTION

We can describe a scenario by a UML [5] use case model and define the details of the communication by the sequence diagram. A high-level generic scenario is defined as follows:

Description of use case
Collaboration Ci achieves goal Gj

Precondition *Goal is eligible for execution and trigged*
$Gj \in GOALS \wedge$
$Gj \in Act\_G \wedge$

Postcondition *Collaboration achieves goal or Collaboration reports failure*

Includes: *Recover_Scenario_Ci_Gj*

Normal sequence of events:

1. The coordinator of *Ci* receives a notification that a goal is activated and changes the status of the collaboration, i.e.,

   $C\_STATUS\ (Ci) := Activated$

2. The coordinator broadcasts an invitation to join a collaboration to the components of *ColENV* and monitors that the required configuration is established
3. When a configuration is established, i.e.,

   $MINCONF(Gi) \leq CUR\_CONFIG(Ci)$

   it broadcasts the message engaged to the involved components and changes the status of the collaboration, i.e.,

   $C\_STATUS\ (Ci) := Active$

4. Components communicate to each other to perform the tasks required to achieve goal and the coordinator monitors the status of the components. If it discovers a component failure then go to step 8.
5. When goal is achieved the components report to the coordinator about completion of scenario.
6. Coordinator hands over the control to the collaborative environment manager and changes the status of the collaboration, i.e.,

   $C\_STATUS\ (Ci) := Dormant$

7. The coordinator broadcasts disengage message to all components.
8. The collaboration coordinator re-evaluates the status of the collaboration. If the condition of the sufficient configuration is not satisfied then it changes the status of the collaboration to *Activated* and activates timer.
9. If the components recover within the timeout then the status is changed to *Active* and the normal execution is resumed.

If the components fail to recover within timeout then switch to executing failure recovery scenario *Recover_Scenario_Ci_Gj.*

Description of use case *Recover_Scenario_Ci_Gj*

Precondition
Normal execution of scenario to achieve goal Gj by collaboration Ci failed.
Status of Ci is Activated

Postcondition Reconfiguration and resuming normal execution or permanent failure

Extends: *Collaboration Ci achieves goal Gj*

Sequence of events:

1. The coordination of *Ci* broadcasts a new invitation to join a collaboration and activates a timer

2. If within the timeout the coordinator receives a respond from components whose roles match the roles of failed components then continue. Otherwise the scenario terminates, i.e., go to 4.

3. The coordinator sends engagement message to the newly joining components and changes the status of the collaboration to Active. Normal execution resumes, i.e., the use case *Collaboration Ci achieves goal Gj* resumes.

4. The collaboration sends the failure message to the collaborative environment manager and changes the status of the collaboration *Ci* to *Dormant.*

5. The coordinator broadcasts disengage message to all components.

Let us now depict the described scenario as a sequence diagram. An excerpt from the sequence diagram is shown in Fig. 1. We use the sequence diagram as an input for conducting analysis of deviations in the component interactions. Next, we present our analysis method, HAZOP, adapted for analysis of dynamic behaviour.

IV. GOALS IN COLLABORATIVE ENVIRONMENTS

HAZOP was originally developed in chemical industry [1,2]. Essentially, HAZOP provides a structured basis for brainstorming by a group of experts about possible deviations in the behaviour of the system. As a result of conducting HAZOP, experts indentify hazards and propose means to mitigate them.

HAZOP is conducted by applying the list of guidewords to certain system parameters. The list of the guidewords is presented in Table I.

TABLE I. LIST OF GENERIC HAZOP GUIDE WORDS

Guideword	Interpretation
No/None	Complete negation of the design intention. No part of the intention is achieved and nothing else happens
More	Quantitative increase
Less	Qualitative increase
As Well As	All the design intentions is achieved together with additions
Part of	Only some of the design intention is achieved
Reverse	The logical opposite to design intention is achieved but something quite different happens
Early	Something happens earlier than expected relative to clock time
Late	Something happens later than expected relative to clock time
Before	Something happens before it is expected, relating to order of sequence
After	Something happens after it is expected, relating to order or sequence

In Table I, we present the generic guideword list from the Defence Standard 00-58 [1] and IEC-61882 [2]. The HAZOP methods has been adapted to various domains and received several interpretations that allows the designers to focus on a wide spectrum of aspects – from human errors to software.

For models of dynamic system behaviour, e.g., such as sequence diagrams, many guidewords interpretations can be used for exploring deviations during the component interactions. In this paper, we adopt the reinterpretation of the guidewords for HAZOP proposed in [3] for the UML sequence diagrams. The adopted interpretation of the HAZOP guidewords [3] to sequence diagram is given in Table II.

Let us now demonstrate an application of the guidewords to the basic scenario of component interactions. We consider only a few examples that have resulted in identifying deviations that hinder achieving the required goal.

Messages outgoing from the coordinator:

Invite message:
No: Execution of scenario is not trigged
Before: Message sent when the goal is not trigged
Earlier: Message sent before the goal is trigged
Later: Message sent with the delay

Messages from the components:
Confirm participation
No: Message might block execution of the goal if no other component confirm
After: Message delays execution of scenario

Inter-component Communication Message:
No: No message is sent after completing execution: Deadlocks goal execution
More than: several messages sent after completing execution: scenario is executed in wrong order
Before /Early: message is sent before task completes and trigs earlier than required execution of tasks in another components
Later: execution of the goal is delayed.

TABLE II. INTERPRETATION OF HAZOP GUIDE WORDS

Attribute	**Guideword**	**Interpretation**
Predecessor/ successors during interactions	No	Message is not sent
	Other than	Unexpected message sent
	As well as	Message is sent as well as another message
	More than	Message sent more often than intended
	Less than	Message sent is often as intended
	Before	Message sent before intended
	After	Message sent after intended
	Part of	Only a part of a set of messages is sent
	Reverse	Reverse order of expected messages
Message timing	As well as	Message sent at correct time and also incorrect time
	Early	Message sent earlier than intended time
	Later	Message sent later than intended time
Sender/ receiver objects	No	Message sent but never received by intended object
	Other than	Message sent to wrong object
	As well as	Message sent to correct object and also an incorrect object
	Reverse	Source and destination objects are reversed
	More	Message sent to more objects than intended
	Less	Message sent to fewer objects than intended
	No/none	The conditions is

Message guard conditions		not evaluated and can have any value (omission)
	Other than	The condition is evaluated true whereas it is false, or vice versa (commission)
	As well as	The condition is well evaluated but other unexpected conditions are true
	Part of	Only a part of conditions is correctly evaluated
Message guard conditions (cont.)	Late	The conditions is evaluated later than required (other dependant conditions have been tested before)
		The conditions is evaluated later than correct synchronisation with environment
Message parameters/ return parameters	No/None	Expected parameters are never set/returned
	More	Parameters values are higher than intended
	Less	Parameter values are lower than intended
	As Well As	Parameters are also transmitted with unexpected ones
	Part of	Only some parameters are transmitted Some parameters are missing
	Other than	Parameter type/number are different from those expected by receiver

Our analysis allows us to derive recommendation how to mitigate the impact of deviations. For instance, it clearly demonstrates that a message omission leads to the system deadlock. Therefore, a time out mechanism should be implemented to ensure that the goal execution progresses despite possible message omissions.

If a component sends a confirmation of a task completion then the consequent task might start in an incorrect state. To mitigate this hazard, a coordinator might additionally send a check to ensure that the required task was indeed completed.

V. Conclusion and Related Work

Our analysis is based on formal definition of relations between the goals that collaboration should achieve and states of the components. A formalization of a goal-oriented development was proposed in [6]. In this paper, the focus was not only on formal representation of relationships between the agents and goals but also on the systematic analysis of deviations. An approach to integration with other techniques for safety analysis was proposed in [8]. This work is relevant to a high-level analysis of collaboration. An approach to analysis of collaborative behaviour in the context of mode-rich systems was proposed in [9]. The focus of this work was on reasoning about modes of collaborating components.

A formalization of agent collaboration has been performed in [7]. The focus of this work was on tolerating temporal agent failures, while in our work we focused on systematic analysis of deviations in component interactions.

HAZOP analysis has been adapted to analyse human computer-interactions as well as process deviations. Our use of HAZOP is similar to the former and allows us to reason about interactions of components participating in collaboration.

In this paper, we proposed a systematic approach to analyse component interactions in collaborative environments. We formally defined relationships between the state of components and ability of collaboration to achieve the required goals. We have demonstrated that the HAZOP method allows us systematically study deviations in the component interactions and establish a link between errors in interactions and goal achieving.

As a future work, it would be interesting to apply the proposed approach to complex collaborative environment from the Internet of Things domain.

References

[1] DefStan00-58. HAZOP studies on systems containing programmable electronics. Defence Standard, Ministry of Defence, UK, 2007.

[2] IEC61882. Hazard and operability studies (HAZOP studies). Application guide. International Electrotechnical Comission, 2001.

[3] J. Guiochet, D. Martin-Guillerez, and D. Powell. “Experience with Model-Based User-Centered Risk Assessment for Service Robots”. HASE 2010: pp. 104-113.

[4] A. van Lamsweerde “Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice”. *Proceedings of RE’04*, 12th IEEE Joint International Requirements Engineering Conference, Kyoto, Sept. 2004, pp. 4-8.

[5] OMG-UML2.OMG unified modeling language (UML). Superstructure, v2.1.2. Oject Management Group.

[6] I. Pereverzeva, E. Troubitsyna, and L. Laibinis. “Formal Goal-Oriented Development of Resilient MAS in Event-B”. In Proc. of *Ada-Europe 2012 --17th International Conference on Reliable Software Technologies.* Lecture Notes in Computer Science 7308, pp. 147–161, Springer, June 2012.

[7] L. Laibinis, E. Troubitsyna, A. Iliasov, and A. Romanovsky. “Rigorous Development of Fault-Tolerant Agent Systems”. In M. Butler, C. Jones, A. Romanovsky and E. Troubitsyna (Eds.), *Rigorous Development of Complex Fault-Tolerant Systems*. Lecture Notes in Computer Science, vol. 4157, pp. 241-260, Springer, Berlin, November 2006.

[8] K. Sere and E. Troubitsyna. “Safety Analysis in Formal Specification”. In J.M. Wing, J. Woodcock, J. Davies (Eds.) *Proc. of FM'99 - Formal Methods: World Congress on Formal Methods in the Development of Computing Systems,* Lecture Notes in Computer Science 1709, pp. 1564 – 1583, Springer, France, September 1999.

[9] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala. “Developing Mode-Rich Satellite Software by Refinement in Event-B”. *Science of Computer Programming,* 78(7), pp. 884-905, 2013.

[10] Internet of Thigs. www.**internet-of-things**.eu/. Accessed July, 2013.