A Coordination Model for Coordinating Software Components

Hsin-Hung Lin, h-lin@jaist.ac.jp
Japan Advanced Institute of Science and Technology

1. Introduction

Our research is to introduce a coordination model in which a coordinator coordinates software components which can increase the interoperability and reusability of software components as well as services. The need of such a coordination model can be depicted with a simple Fresh Market Update Service problem showed in fig.1. As the behaviour described in fig.1, in a usual communication model for software components, the Start message could never be received by the investor because the communication is jammed when the research department wants to send the Data message. In this example problem, we may understand that the stock broker provides services of investing stock markets and the research department provides services of analyzing market trends and other information but people may argue that it is so strange to have such incompatible behaviour. To make it not strange any more, we may further assume a situation that the broker and the research department services are possibly provided by different companies so that these services are not designed as a whole big service. The two services thus may not cooperate well for an investor.

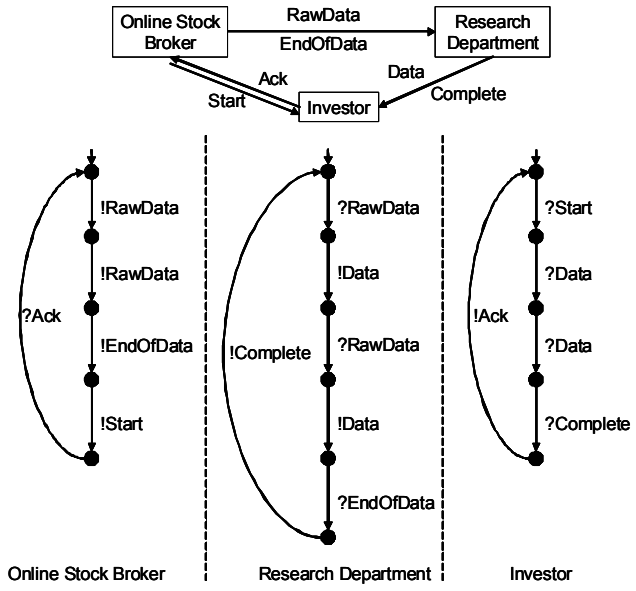

Fig. 1 A Fresh Market Service Problem

To give a solution for the above problem, we introduce our coordination model. The objective of this research is as follows:

1. A new coordination model in which a coordinator coordinates software components.
2. A formal model definition for our coordination model.
3. Simulation and verification of the coordination model, which includes (a) checking whether the software system is perform as designated behaviour and (b) for a given software component system, find a coordinator to orchestrate the components.

To perform the work of verification, we apply model checking technique and use SPIN model checker.

2. The Model

2.1. Overview

The overview of our coordination model is demonstrated in fig.2. The system has two parts: one part is processes (software components) of the system and the other part is the coordinator. Processes of the system communicate with each other by delivering and receiving events. Note that in the viewpoint of processes, they are communicating with each other directly, but actually in the model, events are not directly delivered to its destination process but first be accepted by the coordinator. The coordinator will then store these events in event pool. When an event is needed, coordinator will retrieves the event and deliver it to its destination process.

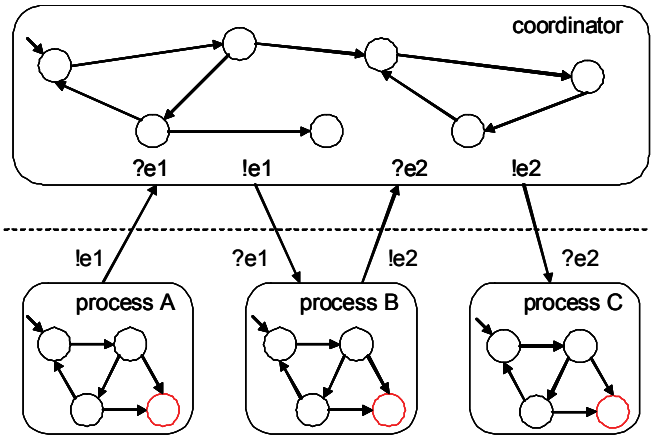

Fig. 2 coordination model overview

2.2. Formal Definition

The formal definition of our coordination model is defined by Buchi automata model. The system is composed of processes with coordinator:

$$T = \{P_1 + \ldots + P_n\} + \text{coordinator}$$

The process P_i has behaviour as Buchi automaton:

$$P_i = (Q_i, q_i^0, A_i^{in}, A_i^{out}, \Delta_i, F_i)$$

where

Q^i, q^{i0} : set of states and initial state

A_i^{in}, A_i^{out} : set of input and output alphabets

Δ^i: set of transition relations including ε -move

The coordinator could be considered also a process, so it is a Buchi automaton having the system's all events its input and output events:

$$P_{co} = (Q_{co}, q_i^0, A_{co}, \Delta_{co}^{in}, \Delta_{co}^{out})$$

where

A_{co} : set of alphabets of the system

Δ_{co}^{in}, Δ_{co}^{out} : transition relations triggered by inputs and outputs of coordinator

The system's total behaviour is the composition of automata of processes and coordinator where the transition is restricted that events have to be first received and then re-send by coordinator. Detailed definition of the system's behaviour is skipped. Note that whether a coordinated system behaves well or not is defined by acceptance definition of Buchi automaton, which means that a coordinated system works well when event sequence can go through each process's final states infinitely often.

2.3. PROMELA model

For simulation and verification, we apply model checking technique and choose SPIN as the model checker. The coordination model has to be transformed to PROMELA model which is the input of SPIN. The transformation includes several parts:

1. Pre-definitions and type definitions.
2. Declarations of global variables: events and channels.
3. Definition of processes including: declaration of states, initialization, and transition executions for sending and receiving events.
4. Definition of coordinator.
5. Definition of init process, including initialization of variables and processes.

Note that we define channels for every event to store events the coordinator received but not send yet. This mechanism makes sure that the coordinator can only send events that have been received before.

3. Progress of 2007

The complete works are as follows:

1. We have reviewed our model and make some adjustment to complete the formal definition.
2. Simulation and verification of coordinator with SPIN model checker. This is performed by applying the technique to the Fresh Market Update Service example.

4. Future Direction

Some more work is in progress including:

- Transformation tool construction: to construct PROMELA model automatically.
- The approach of finding a appropriate coordinator for a given component software system.
- Applying on component based software system framework or service oriented architecture.

5. Publication in 2007

H. Lin and T. Katayama. Coordination and Verfication of Software Components Orchestrated by Coordinator. (in Japanese) *FOSE2007*. November 2007.