# A Framework to Support the Development of Collaborative Components

Hien Le and Surya Bahadur Kathayat

Norwegian University of Science and Technology,
NO 7491 Trondheim, Norway
{hiennam,surya}@item.ntnu.no
www.item.ntnu.no

**Abstract.** In this paper, a framework to support the development of collaborative components is presented. The role of a collaborative component is to support the inter-working among components through their interfaces. By focusing on the behaviors of the collaborative components, the interaction between components can be independently verified and validated. The proposed framework includes: (1) the structural and architecture models of collaborative components; (2) the choreography models to specify the behaviors of the collaborative components; and (3) the orchestration models to specify the behaviors of component types that participate in the inter-working. The orchestration models are created by model transformations from the architecture and choreography models. The created components are placed in the repository so that they can be re-used and composed together.

**Keywords:** Collaborative components, model transformation.

## 1 Introduction

Components are commonly defined as reusable and composable software units having specified functionalities (i.e., local behaviors) and interfaces. Component composition is a process to build complex components from simpler or existing ones [1,7]. On one hand, new components can be rapidly developed by composing existing components together, and thus reducing the development costs. On the other hand, however, there are also many challenges which have not completely dealt with, for example, how to efficiently define and specify component interfaces so that components can be connected and deployed together, or how to specify the interactive and dependent behaviors of composite components [3,4].

We consider components which are collaborative and aiming to establish some desired goals among collaborating entities. Thus, the role of a collaborative component is to support the inter-working among components through their interfaces (also called semantic interfaces of components [2]). By focusing on the behaviors of the collaborative components, the interaction interfaces between components can be independently verified and validated. Once the interfaces of components are completed, the remaining work is to ensure that the local functionalities comply with the designed interfaces.

There works in the literature uses scenario-based approach using formalism such as Use Case Maps (UCM) describe system behavior consisting of large number of collaborating components. However, the problem with such approaches, as identified in [10], is that UCM do not have well-defined semantics for component interface specifications. Some approaches specify interactions and composition of components at the level of programming languages [12], however in this paper, we focus on high-level mechanisms with model-driven approach.

We specify the contracts, similar to conceptual contracts in [11], that specify essential aspects of integrating components. We present a framework to support the development of collaborative components by tackling two challenges: (1) how to model and design collaborative components in such a way they can be easily analyzed and re-used; and (2) how to support the compositions of existing components to create new components taking into account the dependent behavior among components.

The rest of the paper is as following. Section 2 discusses the overview of our collaboration component development framework. Section 3 addresses the issues of composition pattern to support the composition of components. Section 4 concludes the paper.

## 2 Collaborative Component Development Framework

Figure 1 shows the overview of the framework. The proposed framework includes two main layers: design layer and repository layer. The design layer includes: (1) the structural and architecture models of collaborative components; (2) the choreography models to specify the behaviors of the collaborative components; and (3) the orchestration models to specify the behaviors of component types that participate in the inter-working. The orchestration models are created by model transformations from the architecture and choreography models. The created components are placed in the repository layer so that they can be re-used and composed together.

The detail of each model and layer of the framework is discussed in the following sub-sections.

### 2.1 Component Structure and Architecture

We categories two types of components: elementary and composite components. Elementary components are the one which can not be decomposed further and composite components are composed from elementary components. UML collaboration composite structure are used to specify the structural (i.e., component roles and interactions) of elementary components and the architecture of composite components (i.e., how elementary component roles are combined together into composite roles), respectively. For example, Figure 2(a) shows an example of a LocationService component in which an elementary user role interacts with an elementary server role. When existing components (which can be either elementary or composite ones) are composed to form a new composite component, role of these components are bound to composite roles in the composite component. For example, in Figure 1, the roles *roleA* and *roleC* are bound to role *CroleA*.

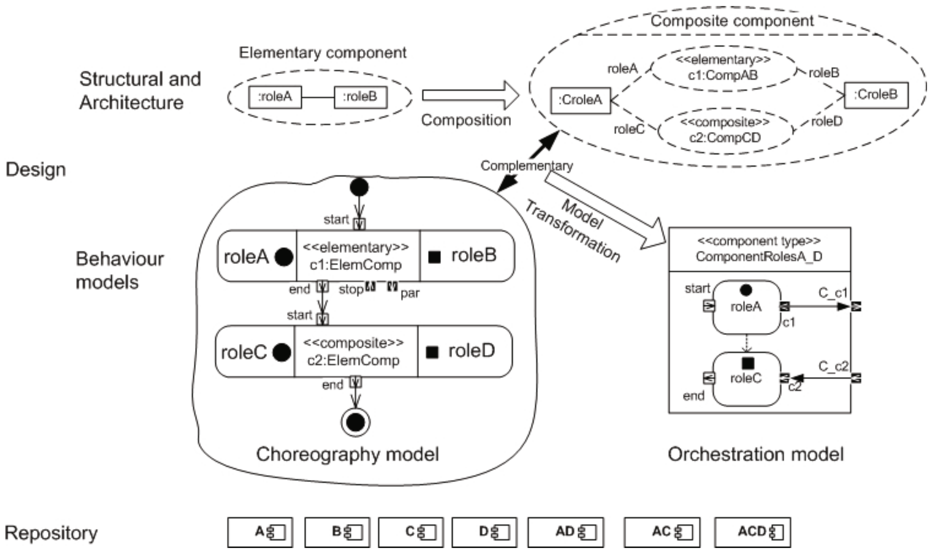

**Fig. 1.** Collaborative component development framework

### 2.2 Behavior Models of Collaborative Components

Choreography model describes the behavior of a collaborative component, i.e., how different component participants interact with each other [9,?]. In the case of elementary components, behavior models specify the interaction among participating roles in a component. Figure 2(b) illustrates the behaviors of the LocationService collaborative component.

An component can have all the different types of pins that UML allows, such as initiating, streaming and terminating pins. An initiating pin will start the component if it is not started yet. Terminating pins will terminate the called component. Streaming pins can pass tokens while the called component is active, i.e. a flow connected by streaming pins allows the called components to interact without being stopped.

In this particular example, the user part will take the initial role (i.e., denoted by a filled-circle) and the server part is the terminating role (i.e., denoted by a filled-square). The user periodically reads the geographic coordinate of the location of the user and sends to the server. This location information then can be forwarded to other components. The activities of LocationService component is ended when the server receives signal (via streaming pin stop). It is important to note that this behavior of the collaborative component is fully specified, however, it may not be composable due to the lack of connection points (explained later).

In the case of a composite component, global behavior is described by connecting (sub-) components together thereby specifying ordering and causality among them. In order to simplify the view of global collaborative behavior, an abstract representation of collaborative component will be used (as shown in

Figure 1). We have used choreography models with two different levels of details: flow-global choreography model and flow-localized model [9].

Flow-global choreography is used to define the intended collaborative behavior of composite collaborations on a high level of abstraction avoiding details of localization and resolution of coordination problems. The behavior is defined by an activity diagram connecting actions by flows (see Figure 4(a) for an example of flow-global in which the LocationService component is composed with the *InforService* component). From the flow-global choreography, by model transformation, a flow-localized choreography will be generated [9]. The flow-localized is used to refine the collaborative behavior in sufficient detail to allow extensive analysis and to automatically synthesize the behavior of component types providing orchestration (explained in Section 2.3).

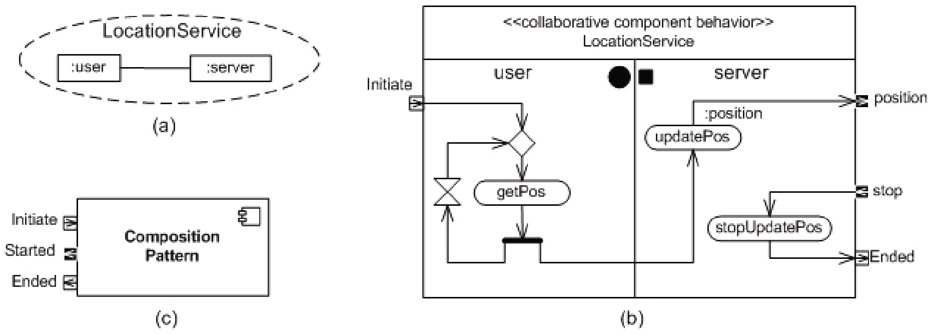

**Fig. 2.** Behavior of LocationService collaborative component

## 2.3 Orchestration Models to Specify the Behavior of Local Functionalities

From the global collaborative behavior models, one can synthesize component models, also called orchestration model [8,9]. Orchestration models describe local behavior (i.e., functionalities) of a component including the information about the collaborative components it participates in and their ordering. An orchestrator may have well defined interfaces and internal behavior. Orchestration models are created by model transformation mechanisms which are based on both collaboration architecture and choreography models. The architecture of components specify which roles (elementary or composite roles) should be placed within a component scope. While the choreography model instructs how should local components (e.g., related to user part) be connected together.

In Figure 1, the architecture specifies that the elementary role *roleA* and the composite role *roleC* will be placed in one component type. Information about the collaborations it participates in and the ordering imposed by external collaborative activities are retained in order to support automatic realization as well as compatibility checks during subsequent composition of components into working systems, detailed algorithm is given in [8,6].

## 3 Composition Pattern of Components

Components can be put into the library and composed later in order to make a complete system. Composition of components together is done by means of pins and collaborations as interfaces. However, in order to deal with the dependent behavior among components (either collaborative or non-collaborative components), collaborative components may need to be modified, for example, telling other components that the component has been initiated, started or ended.

In Figure 4, the InformationService will start when the location of the user is identified by the LocationService. This interactive behavior can be captured and specified by the choreography model. However, at orchestration level, this dependency property between *User_LocationService* and *User_InforService* components (which are composed within the scope of the User component) cannot be implemented. We have identified the following three possibilities:

– If a component is playing a initiating role in a collaboration, then it will retain *Initiate* pin indicating that it will start a component. In this case, a component may not have pins indicating when a component ends, for example user component in the *LocationService* component.
– If a component is playing a terminating role in a collaboration, then it will retain *Ended* pin indicating that component is terminated. In this case, a component may not have pin indicating when a component has been initiated, for example a server component in the *LocationService* component.
– If a component is playing participating role i.e., neither starting nor terminating, then component will not have any pins indicating start and end of a component.

In order to address these issues, we made adjustments by adding additional pins: *Initiate* to indicate that it starts a component, *Started* indicating that component has been started, and *Ended* indicating that component behavior terminated (see Figure 2c). The main novelty of this composition pattern is that, the pre-designed functional behavior of the collaborative component is unchanged with the additional coordination added in an enclosing activity indicated by dashed flows in Figure 3. Adjustments to the original behavior are done as following (and illustrated in Figure 3):

– In the initiating component type, add a fork after the Initiate pin and send signal indicating that all collaborative components have been initiated.
– In the terminating component type, add a fork before the Ended pin and send signal indicating that all collaborative components have been terminated.
– In the participating component type, capture the signal from other roles to indicate when it starts and when it ends.

Once we have the information about a component for example when it starts and when it ends, we can connect the components together by connecting these pins along the implied flow/ordering from choreography models. For example, Figure 4 shows the composition of inner-components in a User component. The

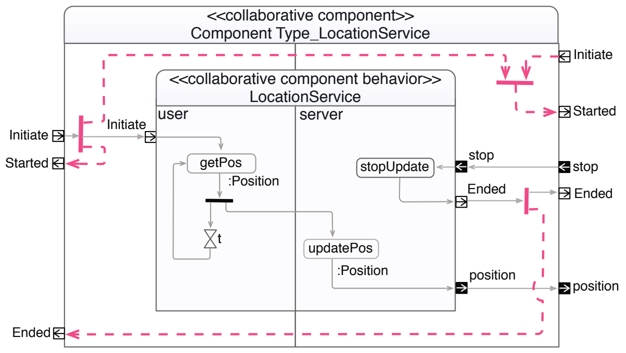

**Fig. 3.** Detailed design of LocationService collaborative component

user component plays roles in *LocationService* and *InforService*. Their ordering is such that once the LocationService component updates a position information then *InforService* component can be started, i.e., to be ready to receive information from the server components. Note that there is some implied concurrency between them due to the streaming pin. This means the user component of location service and user component of the information service may overlap. This is specified by a flow connecting *Started* pin of *User_LocationService* to *Initiate* pin of *User_InfoService* in Figure 4(b).

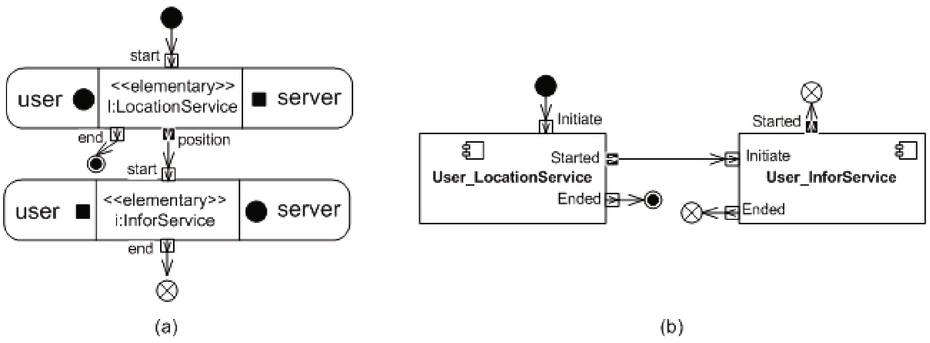

**Fig. 4.** Global component composition view (a) and Local composition of user composite components (b)

## 4 Concluding Remarks

A collaboration based framework to support the development of collaborative components is proposed. Structural and architectural model can be specified using UML collaboration structure and corresponding collaborative behavior, i.e. interaction among the participating components, is specified using UML activity diagrams. Elementary components can automatically be synthesized from choreography and orchestration models and put into a repository. Such components can later be reused and composed in making a complete system. We also discuss a general composition pattern which will be applied to modify the behavior of existing components which might not be composable due to lacking of connectivity points. In addition, the composition pattern can also capture both the sequential composition (i.e., via Ended pin) and parallel composition of components (i.e., via Started streaming pin).

Future work includes: (1) integrating the service-oriented architecture (SOA) paradigm into the structural and architecture of components in the framework so that existing components in the library, which are matched with the user requirements, can be quickly retrieved to be composed; (2) considering additional dependencies of components when they are composed (e.g., suspended or resumed); and (3) verifying and validating the semantic interfaces of composite components.

## References

1. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
2. Bræk, R., Floch, J.: ICT convergence: Modeling issues. In: 4th International SDL and MSC Workshop, System Analysis and Modeling, SAM (2004)
3. Crnkovic, I., Larsson, S., Chaudron, M.R.V.: Component-based Development Process and Component Lifecycle. CIT 13(4), 321–327 (2005)
4. Jisa, D.L.: Component based development methods: comparison. Computer Systems and Technologies, 1–6 (2004)
5. Kraemer, F.A., Kathayat, S.B., Bræk, R.: Unified modeling of service logic with user interfaces. In: Proceeding of the First International Workshop on Model Driven Service Engineering and Data Quality and Security, p. 3744. ACM, New York (2009)
6. Castejon, H.N., Bræk, R., Bochmann, G.V.: Realizability of collaboration-based service specifications. In: Proceedings of the 14th Asia-Pacific Software Engineering Conference, p. 7380. IEEE Computer Society, Los Alamitos (2007)
7. Lau, K., Wang, Z.: Software Component Models. IEEE Trans. Software Eng. 33(10), 709–724 (2007)
8. Kathayat, S.B., Bræk, R., Le, H.N.: Automatic derivation of components from choreographies - a case study. In: International Conference on Software Engineering, Phuket, Thailand (2010)
9. Kathayat, S.B., Bræk, R.: From flow-global choreography to component types. In: Kraemer, F.A. (ed.) SAM 2010. LNCS, vol. 6598, pp. 36–55. Springer, Heidelberg (2011)
10. Bruin, H.D.: Scenario-Based Analysis of Component Compositions. In: Proceedings of the Second Symposium on Generative and Component-Based Software Engineering, pp. 1–18. Springer, Heidelberg (2000)
11. Kiniry, J.R.: Semantic component composition. In: Proceedings of the Third International Workshop on Composition Languages, pp. 1–13. ACM, New York (2003)
12. Kristensen, B.B., May, D.C.: Component Composition and Interaction. In: Proceedings of International Conference on Technology of Object-Oriented Languages and Systems (TOOLS PACIFIC 1996) (1996)