Surya Bahadur Kathayat

On the Development of Situated Collaborative Services

Thesis for the degree of Philosophiae Doctor

Trondheim, February 2012

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics
and Electrical Engineering
Department of Telematics

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Telematics

ISBN 978-82-471-3357-6 (printed ver.)
ISBN 978-82-471-3359-0 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2012:46

Printed by NTNU-trykk

Dedicated To
Lal Bahadur (My Father)
Motikala (My Mother)
Sutthasinee (My Wife)

Abstract

In this thesis, we consider distributed, reactive and collaborative services. In such services, two or more entities (components) interact in order to achieve their objectives. The primary goal of this thesis is to provide a methodology comprising abstract notations and tools that support the domain and enable rapid development of situated collaborative services. The following challenges have been identified.

- Finding generic architectural concepts and services.
- Finding suitable methods for specification and design of services and components.
- Defining mechanisms for composing smaller unit of specifications into composite units.
- Defining mechanisms for transforming abstract service specifications into more detailed specifications until application code can be automatically generated.

The proposed methodology consists of three types of models:

- Structural models that specify the structure of collaborating entities in a service. UML collaboration and collaboration uses are used for this purpose.
- Service behavioral models, also called choreography, that specify the global behavior among collaborating entities in a service. UML activity diagram are used for that purpose.
- Orchestration models, also called component behaviors, that specify the local behavior of each collaborating entity in a service. Orchestration models are also specified using UML activity diagrams.

We distinguish between flow-global choreography and flow-localized choreography. The flow-global choreography is intended to be used together with domain experts to define and validate the desired global behavior of services. Mechanisms are proposed to automatically transform flow-global specifications into the more detailed flow-localized choreography that considers coordination details needed for distributed realization and also helps to analyze the realizability of choreography models.

Methods to synthesize reusable components and their orchestrations from choreography models are proposed. The proposed methodology also consists of the methods to compose these reusable components together via their local interfaces or external interfaces (also called semantic interfaces) in order to design composite components and systems. The concepts of semantic interfaces facilitates dynamic lookup and binding to heterogeneous components.

From a service or a system model, one can automatically generate application code using existing tools and techniques.

The evaluation of the proposed methodology is performed by developing proof of concept applications, by analyzing use cases, and by using logical reasonings.

Acknowledgements

My deepest gratitude goes to my supervisor Professor Rolv Bræk. He showed me a road to the target, guided me through the obstacles along the way, and has been so instrumental and continuous source of inspiration through out my PhD work. This work would never become a reality without his hard work, depth knowledge in the domain, and invaluable problem analysis skills. I salute you, Professor Bræk.

I would like to express my due respect to my co-supervisor Professor Monica Divitini for her great co-operation and help during my PhD work and FABULA project. Many thanks goes to Research Council of Norway (NFR) and Department of Telematics at NTNU for partially funding the research work presented in this thesis.

Loads of thanks to my colleagues at NTNU. Specially Hien Nam Le, Frank Alexander Kraemer, Fenglin Han, and Professor Peter Herrmann for cooperating and writing papers together. This is just the start and I hope that we will continue cooperating for many more years to come. Many thanks to Linda Ariani Gunawan, Sobah Abbas Petersen and Humberto Nicolás Castejón for sharing useful words together when it matters. My project colleagues Mamdouh Eljueidi, Ilaria Canova Calori and Basit Ahmed Khan deserve many thanks for participating in useful discussions in a project and for sharing the office during the PhD study.

Many thanks goes to my dear staffs at Department of Telematics, Randi Schrøder Flønes, Mona Nordaune and Pål Sturla Sæther. Your assistance and positive attitude always made me feel home. Many thanks to Geir Melby, Knut Eilif Husa, Lars Thomas Boye and others at TellU, Asker. I appreciate your assistance especially with ActorFrame tool. Many thanks to Jacqueline Floch and Richard Sanders. It was a nice pleasure to participate in some meetings of UbiCompforAll project.

Many many thanks to my nepalese friends, NeST members in Trondheim. You guys make my stay in Trondheim enjoyable and fruitful. Special thanks to Kamal Raj Chapagain, Hari Neupane, Rishi Sharma, Tara Nath

Subedi, Netra Timalsina, Kamal Acharya, Bibek Thapa, Pranita Shrestha, Hari Shanker Shrestha, Pawan Bhattarai, Chudamani Lamichhane and Krishna Panthi. Many thanks to Nani Raut not only for helping during my travels to Oslo but also for sharing useful words time to time. My due respect goes to all dear bhaujus (Kamala, Radhika, Manju, Sajani, Anita, Sharmila, Munu, Laxmi, Erica) for their kindness, hospitality and delicious food time to time.

At last, not the least, I take an opportunity to express my greatest thanks to all my family members. If I achieved some academic things in my life, all credit goes to one of the most genius person I have ever known, non other than my father. Special thanks also to my wife Sutthasinee Kathayat (Fon) for being so patience when it matters, thank you for your care and love.

Surya Bahadur Kathayat,
October 2011.
Trondheim, Norway

Contents

List of Figures

Part I

Overview

Chapter 1

Introduction

The work presented in this thesis is performed within the context of FABULA project which is part of a multidisciplinary effort among the Department of Telematics, the Department of Information and Computer Science, and the Program for Learning with ICT at the Norwegian University of Science and Technology. The principal objective of the FABULA project was to develop novel principles and solutions for situated collaborative learning services [Fab]. The work presented here contributed to the project by providing a set of methods and mechanisms to design and develop situated collaborative learning services rapidly and flexibly.

A well known principle in system development and in the design science (c.f. Section 1.2) research method is to start by analyzing a problem domain rather than directly creating an artifact for a particular purpose. Analysis of the domain (c.f. Section 1.1) revealed that it is hard to pinpoint any characteristics that sets situated collaborative learning services (SCLS) apart from situated collaborative services (SCS) in general from a development and delivery point of view. The main distinction is the content and the application context of the service, not the fundamental nature. Although motivated by the situated collaborative learning services (SCLS) domain, the problems and solutions discussed in this thesis apply to SCS in general. Therefore the learning aspect itself will not be elaborated in this thesis.

In the following, we start by introducing the domain. Research problems are then formulated. Thereafter the research method is explained including the design and development of artifacts and their evaluation. Finally a summary of scientific publications is given.

1.1 Motivating Domain Example

To illustrate the notion of SCLS, we consider city-wide collaborative learning [CD09, KB09]. In city-wide collaborative learning, users learn by engaging in informal collaborative activities that are enabled by mobile devices and location technologies, unlike in traditional learning where learning implies sitting in a classroom or reading a book. Users work together on collaborative tasks outside the classroom, contributing their fair share to the learning activities. They perform activities and access learning objects depending on their situation in terms of location and state.

We foresee that situatedness enabled by location technologies and collaborative activities is playing an important role in changing the way that learners learn, leading to a paradigm shift in learning processes. The following are some indicators [Fab, KB09]:

- Users are being more mobile and at the same time connected, enabled by mobile technologies.
- Learning materials are being digitized and supported by computers and a wide range of handheld devices. Therefore learning material is becoming available anytime and anywhere.
- Networking has enabled distance education and collaboration among geographically distributed people.
- Social networking has enabled communities and awareness.
- Mobile and location technologies allow location and context to be considered in learning activities.

The technological challenge is to develop appropriate means allowing users to interact with other people and learning resources, supporting different social structures and collaborative knowledge creation, supporting the informal nature of learning rather than traditional class-room based settings, and engaging in exploratory activities and serendipitous incidents. Services and systems should also allow the users to be able to ask, argue, identify and solve problems; search for and utilize resources; share expertise and learning experiences. On one hand domain users such as learners need to be able to dynamically access situation specific services and learning objects. On the other hand domain experts such as teachers need mechanisms that allows them to easily and rapidly develop services that support situated collaborative learning activities. This thesis addresses in particular challenges from the perspective of service design and development.

The domain consists of passive as well as active objects. Passive objects respond to requests while active objects can take independent initiatives on their own to interact in a peer-to-peer fashion. This means that service providers not only provide services that respond to request from requestors but also services that are able to take situation specific initiatives providing for instance a recommendation service. The domain objects may be logically and physically separated, autonomous and heterogeneous, and may concurrently interact with each other and with their environments.

Services provided for this domain should allow learners to interact not only with other learners and groups of learners, but also with situation specific learning objects and other resources such as maps. Due to the dynamic nature of learning situations in city-wide collaborative learning, accessible learning resources are continuously changing. Learning activities are not static but changing depending on the topic at hand, the chosen pedagogical approach and available learning resources. This means that the need for supporting services will vary. Therefore it is important to have mechanisms to compose and develop services easily and flexibly. It is desirable that service composition can be performed by end-users such as teachers or learners themself. Therefore it is important that service composition can be performed in terms close to the domain, rather than in technical terms. This thesis has not addressed end-user aspects fully however it only provides a sound basis that can enable this in the future.

The services are distributed and reactive in nature and the development of such distributed reactive services is complex and error-prone process. Therefore being able to flexibly and rapidly develop such services is certainly a big challenge.

1.1.1 Domain Entities

The core domain entities in the case of city-wide collaborative learning are illustrated in Figure 1.1 and considered to be the following:

- *Users* are those who use the system. They can be *learners* who dynamically access and use services to achieve their learning goals. *Users* are mobile and at the same time connected to each other, and playing active roles in learning activities. Users can be *composers* such as teachers and administrators who compose learning services for learners. There is a challenge to provide proper notations and tools for service *composers* in order to enable rapid and flexible composition of services.

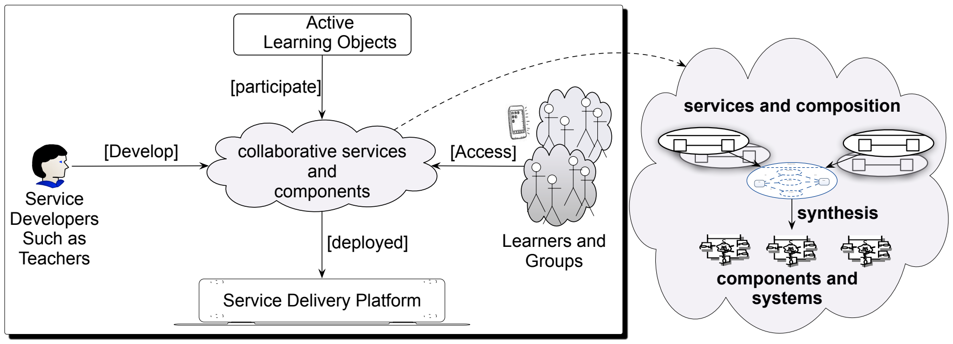


Figure 1.1: Overview of the domain

- *Social configurations* represent sets of users such as groups and meeting places, c.f. [KB09]. The social configurations enable users to participate in collaborative activities and contribute their fair share.

- *Learning objects (LO)* represent any digital or non-digital entity that can be used for learning. Learning objects can be classified as active and passive, and also situated and general. *Active learning objects* represent learning objects that can take a proactive role (i.e. take initiatives on their own) towards *users* in learning activities. Such learning objects are different from traditional passive objects such as web pages. Situatedness means that learning objects are accessible depending on and/or adapting to user situation.

- *Enablers* such as mobile devices that enable users to access basic services (such as SMS, chat, etc) and situation specific application services.

- *Platforms* where services and components are deployed and executed. The purpose of a platform is normally to provide runtime support for service execution, typically communication and scheduling. The platform should allow flexible and dynamic composition of services based on existing ones.

1.1.2 Services and Characteristics

A service is considered as *"an identified functionality aiming to establish some desired goals/effects among collaborating entities"* [BCLR10, KLB12]. In the case of city-wide collaborative learning, services support learning

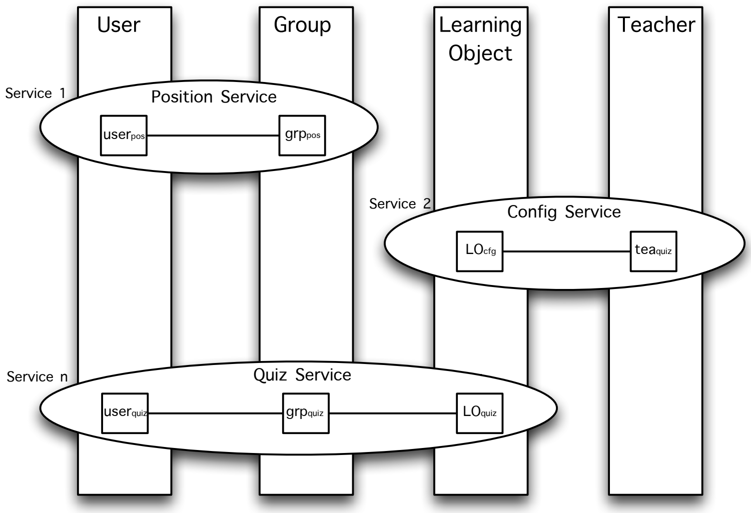


Figure 1.2: Cross-cutting nature of collaborative services

activities that typically cross-cut several domain entities such as *users*, *social configurations*, and *learning objects*. Services can be basic functionalities such as chat, SMS, and document sharing that can be reused in different situation specific applications. Services can also be specific applications involving other smaller services and activities.

Regardless of the particular learning objects and learning objectives, the city-wide collaborative learning domain is about collaboration, mobility and localization. Two or more entities in the domain collaborate and interact with each other in order to achieve some learning objectives. In general, a service to support this can be classified as a *situated collaborative service*.

The cross-cutting nature of collaborative services is illustrated in Figure 1.2. On one hand a service cross-cuts a set of components and on the other hand a component may participate in more than one service. For example, the *Position Service* is spanning over the *User* and *Group* components; and the *User* component contains two inner components (also called *roles* in the following) $user_{pos}$ and $user_{quiz}$ thus providing behaviors and interfaces for the *Position Service* and the *Quiz Service* respectively.

As the city-wide collaborative learning domain is dynamic, the comprising SCS services may come and go, appear and disappear, or be instantiated and destroyed. However, relatively stable domain entities such as *users* and *social configurations* are always there. Use of stable domain entities provides an architectural basis that remains stable and can accommodate a wide range of services and situations adding flexibility and extensibility to a system or platform. Therefore finding generic and stable components is a very important step in system engineering. At the same time it is necessary and challenging to incrementally add cross-cutting service behaviors to the stable entities while avoiding undesired interactions with already existing services.

1.1.3 Service Development Paradigm

Traditionally, software developers or programmers develop systems (consisting of software components) as per the specific user requirements. Now the software development paradigm is gradually changing allowing the end users and domain experts to participate more actively in development.

Domain experts such as teachers may need to compose services (or applications) on their own as per their requirements. Proper notations, mechanisms and tool support is then needed to support the process from the specification and composition of services to application code generation. It is desirable to provide reusable pieces of functionalities encapsulated in abstract building blocks so that non-technical service composers can compose them together by drag-drop. The whole process needs to be as automatic as possible.

The result of this thesis provides a step in this direction by providing abstract notations and tools that support the domain and hide technical details as much as possible.

In this thesis we have taken as a hypothesis that previous work on collaboration based development [SCKB05, Cas08, Kra08] provides a suitable foundation for the specification of collaborative services. This was confirmed by the early demonstrators, and thus we have set out to contribute to its improvement.

1.2 Research Method

The research method is based on the guidelines from design science research [HMPR04]. Design science is a problem solving process [HMPR04]. It requires the creation of an innovative, purposeful artifact for a special problem domain. In IT systems, such artifacts may include models, methods, and instantiations. The artifact must be rigorously defined, formally represented, coherent, and internally consistent [HMPR04]. The artifact must be evaluated in order to ensure its utility for the specified problem. In order to form a novel research contribution, the artifact must either solve an unsolved problem, or provide a more effective or efficient solution to a known problem. Both the construction and evaluation of the artifact must be done rigorously, and the results of the research must be communicated effectively to appropriate audiences.

Considering insights from Peffers et al., 2007 [PTRC08], the research presented in this thesis is organized around five major activities as illustrated in Figure 1.3. These activities are briefly discussed in the following:

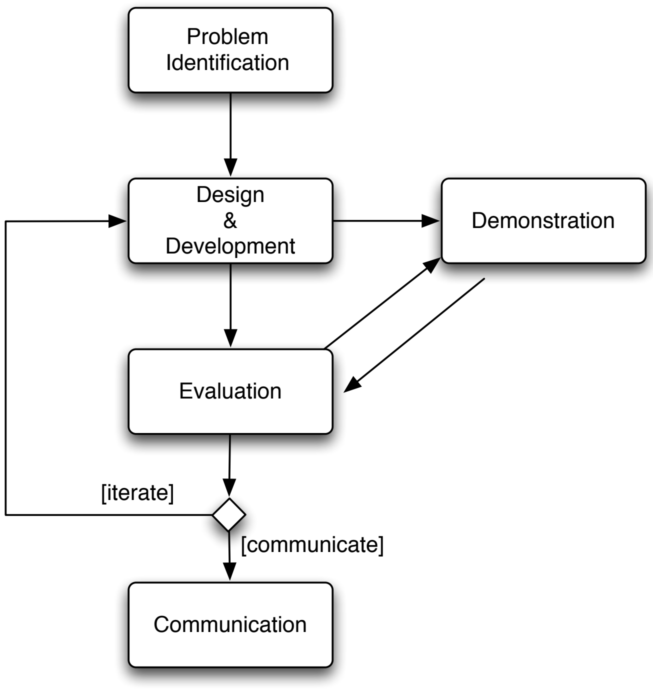


Figure 1.3: Research method based on design science

Problem identification and motivation: This is the beginning where the domain is analyzed and problems are identified. This gives an indication of the type of the artifact to be developed for example services, components and platform in order to effectively provide a solution. Problem specification is given in Section 1.3.

Design and development: This activity is about the design and development of artifacts for providing effective solution to a problem. The artifacts may include services and component models, methods to develop them, and their instantiations. We determine and identify the desired functionalities of artifacts, their architecture, and we discuss a method creating the artifact itself. The method to develop services and components is the main focus of this thesis and is briefly outlined in Section 1.4 and explained in Chapter 2.

Demonstration: This activity demonstrates the uses of artifacts to solve a given problem. We have used experiments and case studies that are briefly outlined in Section 1.5 and explained in Chapter 3.

Evaluation: This activity measures how well the artifact provides a solution to the problem. A comparison can be done with existing solutions or artifacts in terms of functionalities, performance metrics, budgets, result of satisfaction surveys, empirical evidences, logical proofs, etc. After evaluation one can decide whether to iterate back to improve the effectiveness of an artifact or to communicate the results. In our case, evaluation is based on proof of concept demonstrators, evaluation of case studies and using logical reasonings. This is briefly introduced in Section 1.5 and explained in Chapter 3.

Communication: This activity comprises communication in terms of scientific publications. In each publication one needs to communicate to other researchers and audiences the problem and its importance; the artifact, its utility and its novelty, rigor of its design, and its effectiveness. The list of scientific publications included in this thesis is briefly outlined in Section 1.6 and explained in Part II.

1.3 Research Questions

The main problem addressed is formulated as the following main research question (RQ): *how to rapidly and flexibly design and develop situated, collaborative services and components?*. This research question is further divided into following sub-questions:

- **RQ #1**: What basic architectural components (elements) are needed to provide situated collaborative services?
- **RQ #2**: How can the specification and design of situated collaborative services be simplified[1] from the perspective of a higher-level service composer?
- **RQ #3**: How can services be flexibly composed from basic services to satisfy specific needs?

[1]In terms of flexibility, heterogeneity, dynamicity, intuitivity and automatic synthesis.

- **RQ #4**: How can abstract composite service specifications (also called choreography specifications) be mapped to distributed realizations?
- **RQ #5**: How can composite components and systems be composed from smaller components?

RQ 1 is about analyzing the domain. Design and development of services and components from a *composers* perspective is addressed in RQ 2 and RQ 5. Composition of services and components from smaller units are addressed in with RQ 3 and RQ 5. Synthesis of components from choreography models is dealt with in RQ 4. Note that service developers design and realize services and components. Users such as learners use services, and situated learning objects (a kind of components). Services and components are deployed on a delivery platform.

1.4 Design and Development Methodology

In order to answer the research questions mentioned above, a methodology, i.e. a set of methods to develop services and components is investigated assuming that services are reactive in nature and components participate and adapt in situation specific services and may take initiatives on their own.

The methodology is based on the principles of model-driven development (MDD). MDD supports the software development process by creating models on different levels of abstraction and platform independence. Developers first develop more abstract models specifying the pure functionality of a particular solution but hiding aspects of the later realization. These models are then transformed into more detailed and implementation specific models. Based on the refined models, application code can be generated for different platforms. Using MDD, one can thus support a higher degree of automation allowing the service developer to focus on the service behavior rather than the implementation detail.

The term *choreography* is used to denote global behavior involving two or more participants, and the term *orchestration* is used to denote the local behavior of each participant. Choreography is needed to define and analyze the global service behavior, whereas the orchestration is needed to completely define local component behaviors for implementation. Ideally the choreography should be sufficiently precise and complete that component behaviors can be automatically synthesized.

Choreography is provided at two levels of abstraction: *flow-global choreography* and *flow-localized choreography.* The flow-global choreography defines the intended global behavior on a high level of abstraction avoiding details of localization and resolution of coordination problems needed in distributed realization. Flow-localized choreography defines a more detailed global behavior that allows extensive analysis and to automatically synthesize orchestrations. Using code generation techniques, implementation code can then be generated.

UML collaborations and collaboration uses are used to specify the structure of collaborating entities in a service. UML activity diagrams are used for choreography and orchestration models. The service and component models can be put into a repository for later reuse when composing larger components and systems. Details of the design and development methodology is explained in Chapter 2.

1.5 Evaluations and Demonstration

The following aspects are considered important results:

- Platform architecture concepts, their identification and implementation.
- Notations and mechanisms for the specification and design of services.
- Service composition mechanisms from the perspective of domain experts.
- Model transformation techniques from flow-global choreography to flow-localized models.
- Component synthesis techniques from choreography models
- Composition of components to make composite components and system design models

These aspects are evaluated using: (1) Proof of concept demonstrators in [KB09, KKB11, HKLB11], (2) Case studies for the analysis and design in [KB10, KB11b, KLB11, KLB10], and (3) Logical justifications in [KB10] and [KB11a]. Some existing tools, such as Ramses [Kra07] and Arctis [KSH09] are used for modeling, model-checking, composition and code generation purposes. ActorFrame [BHM02] is used as a middleware platform on top of Java SE, J2ME and Android for execution and testing purposes. The details of the evaluation is given in Chapter 3

1.6 Scientific Contributions

The main contributions from this research are the following.

- **C1**: Identification of a set of basic components and services for selected cases: city-wide collaborative learning in [KB09, KKB11, KB11b] and European Rail Traffic Management System (ERTMS) in [KLB10, KLB11, HKLB11].
- **C2**: Architectural platform concepts for situated collaborative learning in [KB09].
- **C3**: Methodology for *flow-global* choreography specification and design in [KB10, KB11b, KLB10].
- **C4**: Principles and rules for analyzing the realizability of *flow-global* choreography specifications in [KB11a].
- **C5**: Principles and rules for mapping *flow-global* specification to *flow-localized* specification in [KB10, HKLB11].
- **C6**: Mechanism for enabling component synthesis from choreography specifications in [KLB10, KB10].
- **C7**: Mechanisms for composing components from smaller components, and to perform system design by composing components together in [KLB11, KLB12].

The papers included in this thesis are the following:

P1: Surya Bahadur Kathayat and Rolv Bræk: *Platform support for situated collaborative learning.* In 1st International Conference on E-Learning, Mobile Learning and Hybrid Learning (eLmL), Cancun, Mexico, 2009, 53-60. IEEE Computer Society.
Relevance to this thesis: This paper identifies some of the platform components and services and the variability needed for the situated collaborative learning. The paper partially answers research question RQ 1 and provides foundation for C1 and C2.

P2: Frank Alexander Kraemer, Surya Bahadur Kathayat and Rolv Bræk: *Unified Modeling of Services Logic with User Interfaces.* In International Journal of Cooperative Information Systems, 20(2), 2011. World Scientific Publishing.

Relevance to this thesis: This paper presents our initial findings for modeling and then composing user interfaces and service logic in a unified way. The service logic is platform independent, while, user interface issues are more dependent on the platform and device being used by the end users. The paper partially answers research question RQ 2 and provides the input to contribution C2.

P3: Surya Bahadur Kathayat and Rolv Bræk: *From flow-global choreography to component types.* In System Analysis and Modeling (SAM), Oslo, Norway, 2011. Lecture Notes in Computer Science, vol 6586, pp. 36-55. Springer.

Relevance to this thesis: This paper identifies two levels of choreography models. A set of transformation rules are proposed to transform from the more abstract choreography models to the more detailed design models, and to do the realizability check of abstract choreography models. Such choreography models provides possibilities to derive reusable component models. This paper partly answers research questions RQ 2 and RQ 4, and provides input to contributions C3, C4 and C5.

P4: Surya Bahadur Kathayat, Hien Nam Le and Rolv Bræk: *Automatic Derivation of Components Using Choreographies - A Case Study.* In International Conference on Software Engineering (SE), Phuket, Thailand, 2010. GSTF Digital Library.

Relevance to this thesis: This paper focuses on the detailed algorithm of deriving component models from choreography models, in particular from flow-localized choreography models. This paper partly answers research question RQ 4, and provides input to contribution C6.

P5: Surya Bahadur Kathayat and Rolv Bræk: *Modeling Collaborative Learning Services - A Case Study.* In International Conference in Collaboration Technologies and Systems (CTS), Philadelphia, USA, 2011, 326-333. IEEE Computer Society

Relevance to this thesis: This paper presents an approach to model collaborative learning services using UML collaboration and activity diagrams. A work-flow of learning activities is specified by ordering learning services. Using such activity-flow models, one can design a learning activity by applying various collaborative learning patterns such as the Jigsaw and Pyramid pattern [ADH$^+$04]. This paper partly answers research question RQ 3, and provides input to contribution C3.

P6: Surya Bahadur Kathayat, Hien Nam Le and Rolv Bræk: *A Model-driven Framework for Component-based Development.* In 15th International Conference on System Design Languages of the SDL Forum Society - Integrating system and software modeling, Toulouse, France, 2011. Lecture Notes in Computer Science, vol 7083, pp. 152-165. Springer.
Relevance to this thesis: This paper presents issues of the component composition and system design. Two types of component interfaces are considered: local interface and semantic interface. Mechanisms to design a composite component by smaller reusable components are discussed ensuring their compatibility via *local interfaces.* A system design is considered as a composition of components where components interact via *semantic interfaces.* This paper partly answers research question RQ 5, and provides input to C7.

P7: Fenglin Han, Surya Bahadur Kathayat, Hien Nam Le, Rolv Bræk, and Peter Herrmann: *Towards Choreography Model Transformation via Graph Transformation.* In 2nd IEEE International Conference on Software Engineering and Service Sciences (ICSESS), Beijing, China, 2011, 508-515. IEEE Computer Society.
Relevance to this thesis: This paper presents possibilities of using graph transformation techniques to implement proposed model transformation rules presented in this thesis. The model transformation rules transform flow-global specification to flow-localized specification. This paper partly answers research question RQ 4, and provides input to contribution C5.

P8: Surya Bahadur Kathayat, Hien Nam Le and Rolv Bræk: *A Collaboration Based Model-Driven Approach for Business Service Composition.* In Handbook on Research of E-Business Standards and Protocols: Documents, Data and Advanced Web Technologies, chapter 27, pages 594-617. Published in January 2012 by IGI Global.
Relevance to this thesis: This book chapter presents how the proposed service engineering method can be used in the business service domain, as a framework for composing business services. It discusses two complementing dimension of compositions: *service composition* where a system is designed as a composition of services, and *component composition* where a system is composed from reusable components. Reusable components can be synthesized from service models. Service composition models can also provide useful information during compatibility check among composing components in component composition. This paper partly answers research question RQ 3

and RQ 5, and provides input to contributions C3 and C7.

P9: Surya Bahadur Kathayat and Rolv Bræk: *Analyzing Realizability of Choreographies Using Initiating and Responding Flows.* In Model-Driven Engineering, Verification, and Validation (MoDeVVA), Wellington, New Zealand, 2011. ACM Digital Library.
Relevance to this thesis: This paper presents details of the realizability and composition issues that needs to be solved when realizing the flow-global choreography models. Mechanisms to detect problems and their resolutions are proposed in terms of propositions and logical reasoning, and explained using a case study. This paper partly answers research question RQ 4, and provides input to contribution C4.

In a nutshell, the relationship between the contributions, research questions and published papers is shown in the Table 1.1.

Research Question	Contribution	Papers
RQ 1	C1, C2	P1
RQ 2	C3	P2, P3
RQ 3	C3, C7	P5, P8
RQ 4	C4, C5, C6	P4, P7, P9
RQ 5	C7	P6, P8

Table 1.1: Relationship between research questions and contributions

1.7 Outline of the Thesis

The thesis is organized into two main parts. The main body of the thesis is presented in *Part II* as a collection of papers. *Part I* presents an overview of the work in *Part II*, and is further structured as following:

- Chapter 1 (this chapter) presents the domain as motivation for the service engineering method investigated in this thesis. Research questions are identified, and the research method is explained. An outline of the proposed service engineering methodology, evaluation aspects, scientific contributions and a summary of each selected papers is given.
- Chapter 2 provides a more detailed introduction to the proposed service engineering method.
- Chapter 3 discusses evaluation of the various aspects of the service engineering method presented in this thesis.

- Chapter 4 gives a summary of related work.
- Chapter 5 provides a discussion and concluding remarks.

Chapter 2

Service Engineering

The proposed model-driven service engineering approach is illustrated in Figure 2.1. It consists of mainly three types of models - *structural* models, *choreography* models and *orchestration* models.

- Structural models specify the structure of roles played by components collaborating in a service.
- Choreography models specify the global service behavior in terms of the ordering of sub-services and local actions.
- Orchestration models specify the local behavior of each role and component.

The approach aims to provide support for abstract choreography models that are sufficiently precise and complete that component behaviors can be automatically synthesized. A service composer (for example a domain expert rather than a software developer or programmer) specifies service behavior using high-level choreography models, called *flow-global choreography* models. Flow-global choreography models can be transformed into more detailed choreography models called *flow-localized choreography* models. The flow-localized choreography models allow extensive analysis and to automatically synthesize component models, called *orchestrators* that correspond to the roles. Orchestrators can then be composed together in different ways in order to compose larger components and systems.

In the following sections, a detailed description of the overall approach illustrated in Figure 2.1 is given.

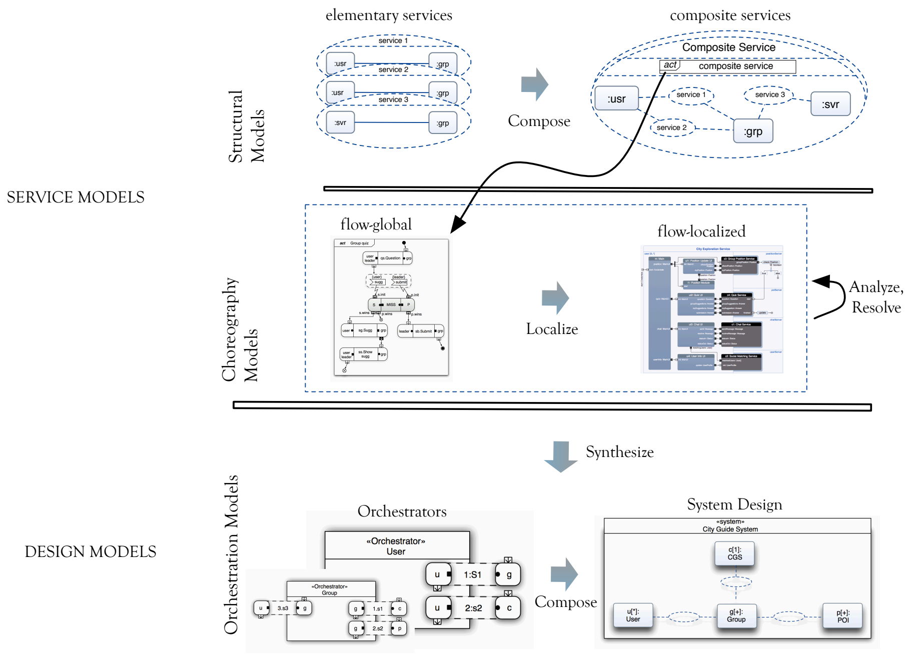


Figure 2.1: Overall approach - service design and development

2.1 Structural Models

UML collaborations are well suited to specify structural service models [OMG09, Cas08, Kra08]. Collaborating entities in a service are represented as collaboration *roles*, and lines between collaboration roles specify that the roles interact with each other. When a service is composed from smaller services, the sub-services are specified using the concept of *collaboration use* where the *roles* of a collaboration use are bound to the roles of a composite collaboration.

Services are classified as either *elementary* or *composite* services. Elementary services are not decomposed further into sub-services whereas a composite service is composed from pre-defined services that are either elementary or composite. Such pre-defined services are re-usable pieces of functionality that may be used in different application settings or contexts. Each service can be used as a *building block* that has associated structure and behavior (c.f. Section 2.2).

A *City Guide* scenario is used in the following to illustrate a composite service specification [KLB12]. Tourists in a city make use of their mobile devices to connect to a city guide server, download the *City Guide* service, and learn about different places of interest around the city. The city guide service provides different guided tours to tourists. During a guided tour, a tourist may communicate with other tourist or groups of tourists who are co-located or having similar interests. Based on the situation, user preferences and requests from the tourists, the system may also provide different recommendation services for example recommending nearby restaurants and museums.

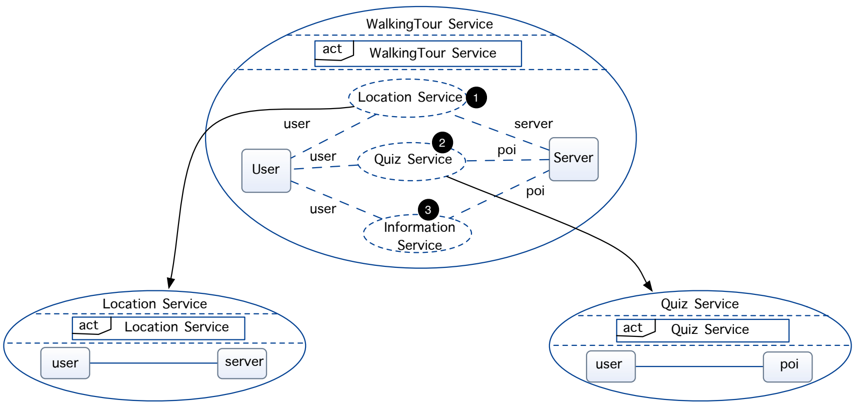


Figure 2.2: Structural model of a walking tour service

The city guide service uses several pre-defined services, for example a location service and a quiz service, that has been identified in a Treasure Hunt Game [KB09] developed previously. The structural model of a *Walking Tour* service (a part of the *City Guide* service described above) is shown in Figure 2.2 where the numeric digits associated with collaboration uses are just for simple referencing in this thesis.

- The *Location Service* allows users to continuously update their position to the server.
- The *Quiz Service* establishes and manages a quiz service concerning point-of-interests in a city.
- The *Info Service* allows users to retrieve or add additional information about different points-of-interests.

One can see in Figure 2.2 that the roles of a collaboration use are bound to the roles of an enclosing collaboration for instance that the *user* role and *server* role of the *Location Service* are bound to the *User* and the *Server* of the *Walking Tour* service respectively. Similarly the roles *user* and *poi* of the *Quiz Service* are bound to the roles *User* and *Server* of the *Walking Tour* service.

A structural model also contains reference to the activity diagram defining the choreography given as (*act* "service name") in a compartment below the title. For example the choreography of the *Walking Tour* service is provided in the diagram referenced as *act* "Walking Tour Service" in Figure 2.2.

2.2 Choreography Models

The global service behavior is specified using UML activity diagrams. It is assumed that the behavior of collaboration uses representing sub-services is defined by an activity diagram having the same name as the collaboration. The choreography is specified by connecting *actions* together by *flows* and *control elements* such as join, fork, decision and merge. The actions call the activity associated with the corresponding sub-services. This means that sub-service behavior is invoked by *Call Behavior Actions* in the choreography of a composite service [OMG09].

As mentioned in Section 1.4, two-levels of choreography models are provided. The flows and control elements in *flow-global* specification are not localized to any service roles, whereas the flows and control elements in *flow-localized* specification are localized to particular service roles. The details are discussed in the following sections.

2.2.1 Flow-global Choreography Models

Flow-global choreography specifies the intended global ordering of composite collaborations on a high level of abstraction avoiding details of localization and resolution of coordination problems that may occur at the level of orchestration. We assume that this is the right level for discussing the intended behavior with end-users and other stakeholders. It is also a useful early step in the formalization of requirements [KB10].

The flow-global choreography of a composite service is specified using normal UML activity diagrams and their semantics [KB10, KLB12]. The intended global behavior is defined by connecting actions i.e. *Call Behavior Actions* by flows that are not localized to any particular role. For example in the flow-global choreography of a *Walking Tour* service in Figure 2.3, the flow from the *end* pin of the *Quiz Service* to the *start* pin of the *Information Service* is global i.e. not localized to any of the roles *user* and *poi*.

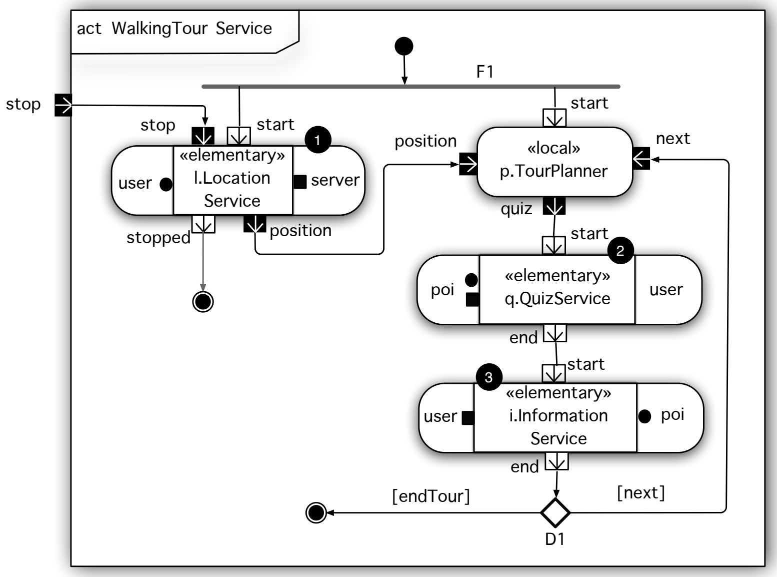


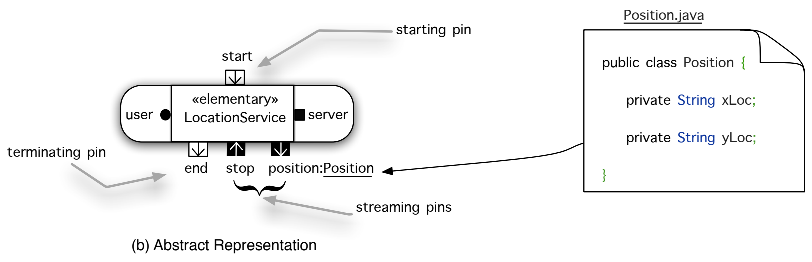
Figure 2.3: Flow-global choreography model of a walking tour service

The actions of a choreography may either represent the behavior of a collaboration or a local activity. The behavior of such actions is defined separately and may be available in a repository in the form of building blocks as mentioned previously.

The notation for the actions symbols as shown in Figure 2.3 originates from [CBvB07]. Compared to those we also allow streaming and interruption. The action symbols, illustrated in Figure 2.4 for the location service, provides the following information:

- Roles participating in a collaboration are indicated by partitions of the action symbol defined by solid lines. Note that UML does not specify a particular notation for partitions and therefore this form is considered to be compliant with UML. For example the *user* and *server* are the participating roles in the *Location Service* illustrated in Figure 2.4.

- Action pins are owned by the actions itself rather than specific service roles. An action may contain different types of pins: initiating pins, terminating pins, and streaming pins. *Initiating pins* initiate



```
public class Position {

    private String xLoc;

    private String yLoc;

}
```

Figure 2.4: Action notation for the location service

the action and is represented by unfilled incoming pins with incoming arrows, for example the *start* pin in the *Location Service* in Figure 2.4. *Terminating pins* indicate the termination of the action and is represented by unfilled outgoing pins with outgoing arrows, for example the *end* pin in the *Location Service* in Figure 2.4. *Streaming pins* can pass tokens while the action is being active and is represented by filled incoming and outgoing pins, for example the *stop* and *position* pin in the *Location Service* in Figure 2.4.

- Initiating and terminating roles of an activity is indicated by filled circles and square boxes respectively. For example *user* is the initiating role and the *server* is the terminating role in the *Location Service* in Figure 2.4. This is not part of standard UML, but may be provided by additional profiling [KLB12].

To define and use data in flow-global models, the normal way defined for UML activity diagrams is used:

- A pin can carry certain types of data. The *position* streaming pin of the *Location Service* as illustrated in Figure 2.4 carries values of the *Position* data type consisting of GPS coordinates *xLoc* and *yLoc*. Similarly the *quiz* pin of the *Tour Planner* carries values of *Quiz* datatype consisting of questions and sets of answers pertaining to a particular point-of-interest. In Figure 2.3, data types associated with pins and flows are omitted. A city exploration service (i.e. variant of the city guide service) is illustrated with complete data types in [KKB11].

- Global variables may be used within the scope of a given flow-global choreography model. Such variables can be used for example in de-

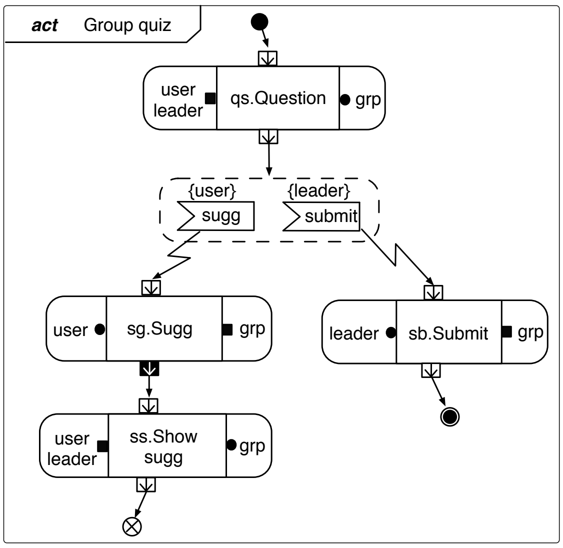


Figure 2.5: Group quiz service

cision nodes in the flow-global specification, and be modified by call operation actions.

Participating roles in a choreography may take independent initiatives to start a collaboration. The situation where there is a choice between collaborations that are initiated independently by different roles is called *initiative choices* [CBvB07, KB10, KSH09]. The case of initiative choices can be modeled in flow-global choreography specification using the notions of interruptible regions and interrupting flows in UML activity diagram. The *Group Quiz* service (a variant of *Quiz* service) from [KB10] serves as an illustration as shown in Figure 2.5. In the group quiz service, the group *grp* first forwards a question to the group leader and all group members. The group members may independently suggest answers or the leader may submit an answer. The intension in the flow-global specification may be to choose either one or the other option, but in a distributed realization there may arise a situation where a user is suggesting an answer at the same time the leader is submitting. This means that independent initiatives may collide and require additional coordination to resolve which initiative to select. Resolution mechanisms are discussed in Section 2.3.4.

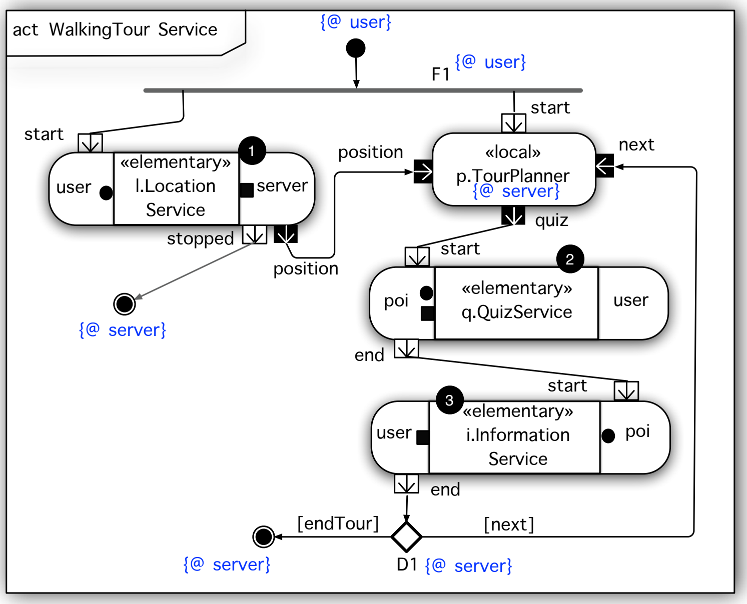


Figure 2.6: Flow-localized choreography model of a walking tour service

2.2.2 Flow-localized Choreography Models

In a distributed realization the global flows of flow-global choreography must be enforced by local flows. The purpose of the flow-localized choreography is to define the local flows and to specify coordination details needed in a distributed realization such as resolution of mixed initiatives. Flow-localized choreography is intended to allow extensive analysis and to support automatic synthesis of component types or orchestrators. The flow-localized choreography also uses UML activity diagrams to specify global behavior. The behavior is defined by connecting actions, that either represent the behavior of a collaboration or a local activity, by flows that are localized to the roles. This means that flow-localized choreography defines the ordering provided by local flows. Figure 2.6 shows the flow-localized choreography of the walking tour service. It is different from the flow-global choreography in the following respects:

- In the flow-localized model, every pin must be assigned to a particular role. This means that at this level of modeling, one can precisely specify how a service is started, terminated or being triggered in a distributed system.

- In the flow-localized model, non-collaborative (local) activities and control nodes such as fork, join, merge or decision must also be localized, i.e., assigned to one role. This may involve mapping global choices onto one or more local choices and additional interactions.

- Control flows in the flow-localized model are derived from the flow-global model. However, the distributed localized flows have semantic differences compared to the global control flows that reflects the nature of distributed systems. Flow-localized models consider: (1) the possibility of message reordering by a communication medium, (2) sharing of communication medium and sharing input buffer at the receiving end. The analysis at this level helps to detect the potential realizability problems, sometimes referred to as *implied scenarios* [CBvB07], that may occur when a flow-global choreography is mapped to distributed orchestrations. This is discussed in Section 2.3, and papers [KB10, KB11a].

Details of transformation rules to transform flow-global choreography into flow-localized choreography are discussed in Section 2.4.

The Arctis tool [KSH09] supports flow-localized choreography models where partitions representing roles are indicated by “swin-lanes” in UML activity diagrams. Arctis also uses *Call Behavior Actions* to represent collaborative building blocks involving two or more participants. The collaborative building blocks spans several partitions. In contrast, local building blocks specify *Call Operation Actions* which involve only one participant and are therefore local to one partition. Arctis can further be used to find several types of errors using model checking techniques, and to automatically generate application code for different platforms for example Java SE, Android, etc.

2.2.3 Elementary Service Choreography

Collaborative behavior of elementary services can be specified using UML activity diagrams. A workflow among activities performed by service participants is specified by connecting actions by flows (that are either local to a partition or crossing partition borders) and control nodes (that are localized to roles) such as decision, fork, merge and join [OMG09]. Other possibilities for specifying elementary service behavior is to use sequence diagrams or message sequence charts (MSC) [CBvB07].

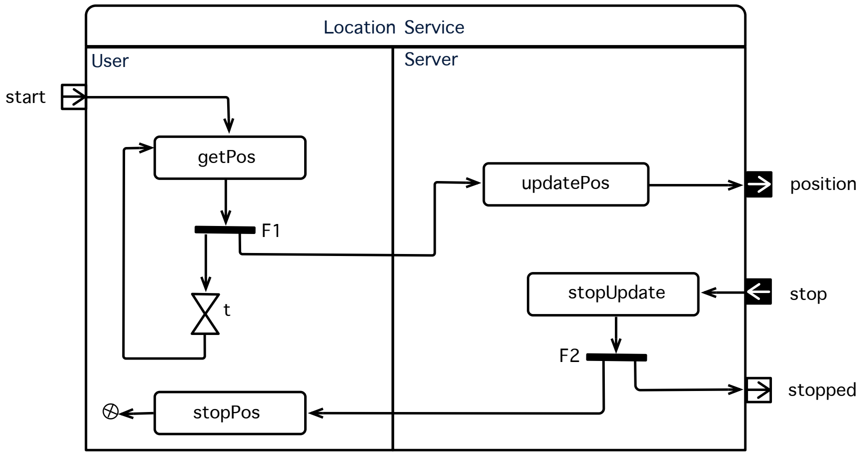


Figure 2.7: Location service behavior model

An example of the *Location Service* choreography using UML activity diagram in the swim-lane form is illustrated in Figure 2.7. Collaborating roles in the service are represented as *partitions* in the UML activity diagram. Activities performed locally by service roles are represented as *local actions* i.e. assigned to a particular partition.

2.3 Realizability of Choreography Models

In a distributed realization, a flow-global choreography will be mapped to possibly distributed components or orchestrators. Mapping of a choreography model aC with roles $P_{j(j=1,\ldots n)}$ to a corresponding set of orchestrators $O(P_j)$ is discussed in Section 2.5.

Ideally the composition of orchestrators shall lead to the same behavior that has been specified in the global choreography. If this is not the case, we consider that there is a *realizability problem.* In other words, if any role orchestrator $O(P_j)$ needs to handle behavior that has not been specified for the corresponding role P_j in aC, this is considered a realizability problem.

In a choreography, the activity flows that cross partition boundaries does not model the possibility that messages may be reordered by the medium, nor that all flows between a pair of components may share a common input buffer at the receiving end, which normally is the case in distributed realizations [KB11a]. In orchestration models we assume an arbitrary transfer delay, and a shared buffer for each component. This models the delays that are inevitable in a distributed realization using an underlying communication medium such as an asynchronous message bus. If this buffer is a FIFO queue, messages will be taken from the queue in strictly the same order as they arrive. If this arrival order is same as the reception order specified by the activity flows, everything is fine. But it may happen that the arrival order differs from the specified reception order either due to different sending orders or due to reordering by the communication medium, which means that the distributed realization will generate reception orders that differ from the order specified in the choreography. This is the fundamental cause of many realization problems.

In choreography we make a distinction between initiating roles and responding roles. In orchestration, it is necessary that responding roles are locally enabled when the roles should be ready to respond. This can be done by mapping flows external to a component to "responding flows" within the component. Thus in orchestration, we distinguish between *initiating flows* and *responding flows* which are defined as follows:

- An *initiating flow* is a local flow and represents the initiation of the next action. In case of an elementary service, an initiating flow initiates the next action that may either be *Call Operation Action* (which is a local action) or *Call Behavior Action.* For example, the initiating flow corresponding to the global flow from aC_1 to aC_2 is the flow that is localized to the initiating role of aC_2. In Figure 2.9(b), the flow that is localized to role A is the initiating flow corresponding to the global flow in Figure 2.9(a).

- A *responding flow* represents that a responding role in the next action should be ready to respond to interactions from other roles in a collaboration. Responding flows are represented as dotted lines with an arrow. In case of an elementary service the next action is a *Call Operation Action.* The responding flow from the send event action *m1* to the receive event action *m2* in Figure 2.8 represents that the *User* component should be ready to respond to *m2* after *m1* is sent. In case of a composite service the next action is a *Call Behavior Action.* The responding flow corresponding to the global flow from action aC_1 to aC_2 in Figure 2.9(a) is the flow which is localized to the responding role of aC_2.

In the following, first the notion of *direct realization* is defined and then the realizability of flow-global choreography is discussed using flow-analysis [KB11a, KB10]. The flow-analysis is found to be useful to detect potential realization problems and resolve them early in the design phase.

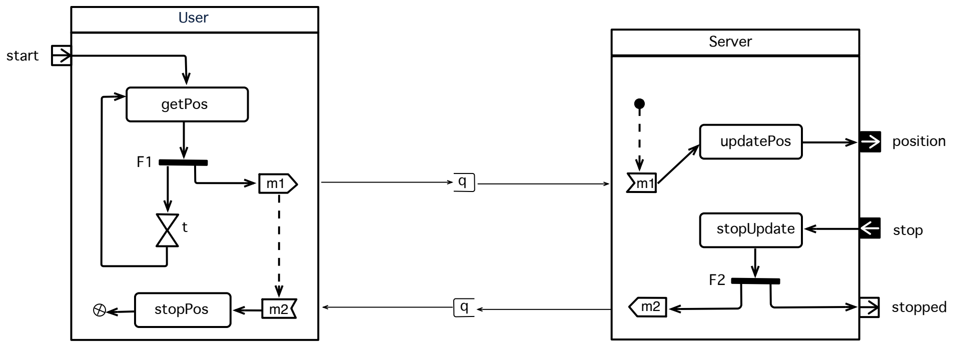


Figure 2.8: Direct realization of location service choreography

2.3.1 Direct Realization

A direct realization of an elementary choreography aC_i with partitions $r_{i,1}, r_{i,2}, \ldots r_{i,n}$ is obtained by directly projecting aC_i to local orchestrations $Od(r_{i,j})$ for each partition. For example the direct realization of the *Location Service* (given in Figure 2.7) is illustrated in Figure 2.8. The projection mechanism involves the following steps [KB11a]:

- Make of copy of the global flow.
- Insert a send event action on all outgoing flows towards a partition boundary.
- Insert a receive event action on all incoming flows from a partition boundary.
- Replace all actions not performed by partition $r_{i,j}$ by no-operation.
- Mark all the control nodes (decision, merge, fork, join) and flows that are not local to $r_{i,j}$ as responding nodes and flows
- Simplify the responding nodes, flows, and no-operation nodes [KLB10]
- Make the simplified external flows internal to $r_{i,j}$ as responding flows, for example the flow from m_1 to m_2 in the *User* component in Figure 2.8.

The given projection mechanism not only elaborates the notion of *direct realization* but also gives a clue about the synthesis of components from elementary service models. The details of component synthesis from composite services is explained in Section 2.5.

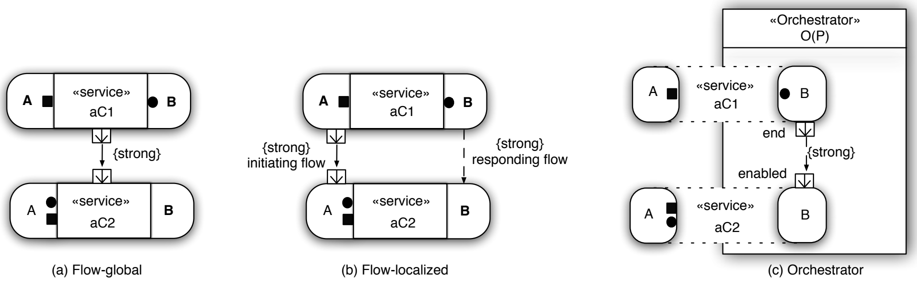


Figure 2.9: Realizability of strong sequencing flow

If the *direct realization* of the choreography aC results in the set of orchestrators $O(P_j)$ then the possible temporal ordering of actions in $O(P_j)$ is exactly same as the temporal ordering of actions in the corresponding role P_j in aC and without any of the following design errors [KB11a]:

- *deadlocks* - a component waits for the message in non-final state that will never be sent,
- *unspecified receptions* - a component receives a message for which there is no transition to consume it in its current state, and
- *orphan messages* - a special kind of unspecified reception where the execution context for which this message was intended does not exist anymore.

Direct realizability means that $O(P_j)$ can be synthesized without adding any additional interactions. This means that it must be possible to localize pins, control nodes and flows without adding interactions.

In the following the realizability of a composite choreography is discussed assuming that each elementary collaboration used as a *Call Behavior Action* in the global choreography is directly realizable.

2.3.2 Realizability of Direct Object and Control Flows

A flow in a flow-global choreography from action aC_1 to action aC_2 is called a direct flow if the flow contains no intermediate control nodes between aC_1 and aC_2. A direct flow from aC_1 to aC_2 is represented as aC_1;aC_2. The semantics of aC_1;aC_2 in the flow-global choreography is that aC_1 is completely finished before aC_2 is started. In a distributed realization the ordering between aC_1 and aC_2 will be determined by the causality enforced by local *initiating* flows. A direct initiating flow in a flow-localized choreography can provide the following causalities [CBvB07]:

- Localized causality - when the terminating role of aC_1 and the initiating role of aC_2 are localized to the same component. In this case all the roles in aC_1 will be completely finished before aC_2 is started, and so it corresponds to the strong sequencing of the global flow aC_1;aC_2. A strong sequencing flow is indicated with the {strong} property.
- Weak causality - when the terminating role of aC_1 and the initiating role of aC_2 are bound to different components, but the initiating role of aC_2 is participating in aC_1. In this case the collaborations are ordered on a per role basis and there is a possible overlapping between aC_1 and aC_2. A weak flow is indicated with the {weak} property. This causality enforces so called weak sequencing. Weak sequencing can be used instead of strong sequencing under the circumstances explained in Section 2.4, and papers [KB10, KB11a].
- Non casual - when the initiating role in aC_2 is not participating in aC_1. This means that no ordering between aC_1 and aC_2 will be enforced by the local flow. This is normally not acceptable and needs to be resolved as discussed in Section 2.4. Such flows are indicated with the {non-causal} property.

In orchestration, there will be no overlap between roles linked by {strong} and {weak} initiating flows. Hence such initiating flows are always directly realizable locally, while {non-causal} initiating flows never are. Considering the responding flows we have the following cases.

- For {strong} responding flows there will be no overlapping of the service roles in aC_i and aC_j i.e. {strong} responding flows are directly realizable. This case is shown in Figure 2.9.
- For {weak} responding flows, there is a possibility of message overlapping between aC_i and aC_j. This is shown in Figure 2.10(c) and explained detail in paper [KB10]. Such flows are directly realizable if: *(1)* all non-initiating roles in aC_j and the terminating role in aC_i are localized to different components i.e. X and B in Figure 2.10 are localized to different components, thus avoiding message overlapping; or *(2)* the participating roles in aC_i and aC_j are communicating over

a FIFO medium. For instance if the communication medium between A and B in Figure 2.10(c) is FIFO then there will be no reordering of message receptions. In general there is a possibility of realization problems in the case of {weak} responding flows.

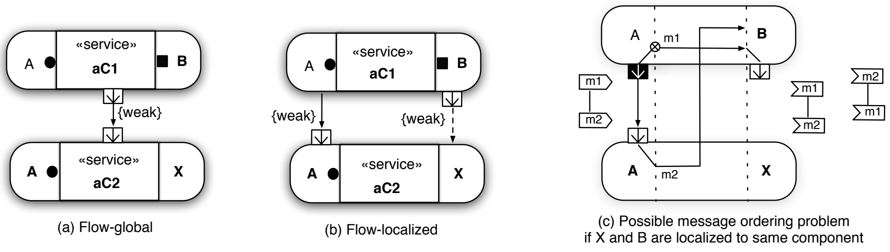


Figure 2.10: Realizability of weak sequencing flow

- {non-causal} responding flows are not directly realizable as they signify that the specified ordering between aC_i and aC_j is not possible. The initiating roles are linked by {non-causal} initiating flows and the participating roles in aC_i and aC_j are linked by {non-causal} responding flows and may overlap arbitrarily.

  If sequential ordering is the intention, one needs *(1)* to add at least one interaction to ensure at least {weak} causality, as for instance in Figure 2.15(b) and 2.15(c), or *(2)* to change either aC_i or aC_j or both so that the flow at least ensures {weak} causality.

There is a possibility of message overlapping in the case of both {weak} and {non-causal} flows. The amount of overlap is determined by how much of a role $r_{i,k}$ (represents the role in the i^{th} collaboration and played by the k^{th} component) in aC_i may remain when a role $r_{j,k}$ in aC_j is started, called *remainder* of $r_{i,k}$ and how much of $r_{j,k}$ that may execute before $r_{i,k}$ is finished, called *beginning* of $r_{j,k}$. Weak responding flows are directly realizable if the ordering of events in the remainder of $r_{i,k}$ and beginning of $r_{j,k}$ can be enforced by the communication medium [KB11a].

When sequential flows consisting of several sequential composition steps $aC_1; aC_2; \ldots; aC_n$ are considered, there is a possibility of indirect responding flows occurring if there is one or more collaborations or local actions in which a component does not participate (represented as *no-op* actions) between collaborations where it participates. This case is shown in Figure 2.11.

If we generalize responding flows to include no-op actions, one can say that an indirect responding flow is a responding flow with one or more no-op

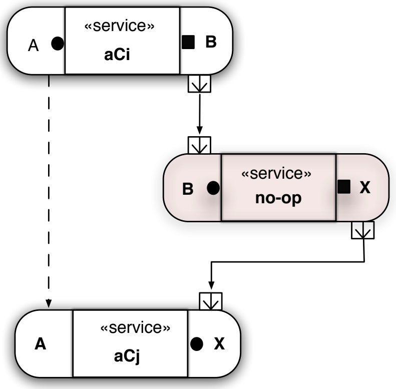


Figure 2.11: General responding flow

actions, a *general responding flow* is a responding flow with zero or more no-op actions.

The first step of a general responding flow determines whether the flow as a whole has property {*non-causal*}, {*weak*} or {*strong*}. A general responding flow that starts with a {*non-causal*} step remains {*non-causal*}. A general responding flow that starts with a {*strong*} step remains strong regardless how many {*weak*} or {*strong*} (no-op) steps that follows. A general responding flow that starts with a {*weak*} step remains {*weak*}, regardless how many {*strong*} or {*weak*} (no-op) steps that follows.

It is found that {*non-causal*} ordering can never be directly realized, and that {*strong*} ordering can always be directly realized. When using {*weak*} ordering there may be problems. Note that problems in sequential composition may only occur between role orchestrations linked by {*weak*} responding flows. Note also that a sequence of responding flows, as shown in Figure 2.12, may extend the overlaps so that A_1, A_2, and A_3 in Figure 2.12 may overlap.

In a nutshell, the following propositions hold [KB11a].

Proposition 1 (realizability of general weak responding flow): A general {*weak*} responding flow between roles $r_{i,j}$ and $r_{k,j}$ is directly realizable only if the remainder of $r_{i,j}$ and beginning of $r_{k,j}$ can be ordered by the underlying communication medium.

Proposition 2 (realizability of sequential composition): A sequential composition aC_1; aC_2; ...; aC_n is directly realizable iff no initiating flows are {*non-causal*} and all general {*weak*} responding flows resulting from a

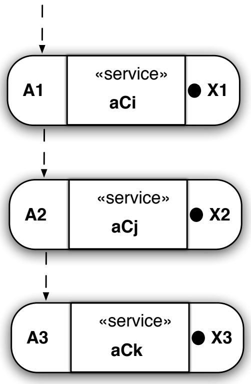


Figure 2.12: Sequence of responding flows

direct realization are directly realizable.

2.3.3 Realizability of Paths and Control Nodes

In a direct realization all the flow-steps and intermediate control nodes should be localized to one component [KB10]. However there may be cases where intermediate flow-steps and control nodes may be part of several flow-paths that are localized to different components. This is an indication of a realizability problem [CBvB07, KB10] and needs to be resolved. The following propositions hold [KB11a].

Proposition 3 (flow-path localization): An initiating flow-path from collaboration action aC_1 to collaboration action aC_2 is directly realizable iff the entire path with all intermediate control nodes can be localized to the role initiating aC_2 and the path is either $\{strong\}$ or is $\{weak\}$ with all its responding flow-paths realizable according to *Proposition 1.*

Proposition 4 (control node localization): If all initiating flow-paths through a control node is directly realizable and the node is localized to one role then the node is directly realizable.

For instance all the flow-paths through the fork node in Figure 2.13 are directly realizable according *Proposition 3.* All the flow-paths through the fork node are not localized to a same role, therefore the fork node in Figure 2.13 is not directly realizable according *Proposition 4.* The resolution strategy is: *(1)* change/replace aC_i or aC_j or aC_k so that the fork node is directly realizable and can be localized to a single component; or *(2)* localize the fork node to the initiating role of the strongest path and resolve other paths according to the rules in Section 2.4 i.e. this case is not directly realizable.

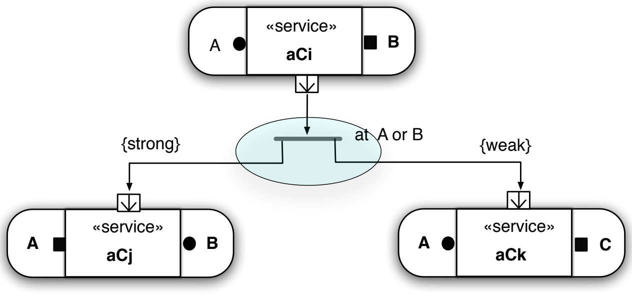


Figure 2.13: Realizability of fork node

If the paths through a choice node are local to different roles, we have a case of *non-local* choice [Kra08, BF04, Cas08]. In the cases of non-local choices, autonomous components may take choice independently and possibly simultaneously, resulting in a conflict and possibly a collision that needs to be resolved. Non-local choices have been extensively studied in the literature, and are normally not directly realizable. It is only when a non-local decision can be performed locally by each component based on local information, in a way that guarantee that the components choose corresponding branches that non-local choice is directly realizable. Some resolution techniques are presented in the paper [KB10].

2.3.4 Realizability of Interruptible Activity Regions

Interruptible activity regions involve interrupting events. If the interrupting events are from two or more different components, as shown in Figure 2.14, there is a possibility of having *mixed initiatives* which are not directly realizable. In such mixed initiatives cases, different components may take independent initiatives to generate interrupting events nearly simultaneously. Hence the events may collide. An additional coordination is then needed to resolve which initiative to select. The resolution strategy depends on the problem at hand and is difficult to solve in a general way. In stead one may develop a library of alternative solutions to select from. One way to handle such situations is to assign primary and secondary priorities to the conflicting partners and let an initiative from the primary side be accepted in all cases. The secondary side must be prepared to receive a primary initiative even after it has issued an initiative itself, and obey the primary one [KSH09, KB10].

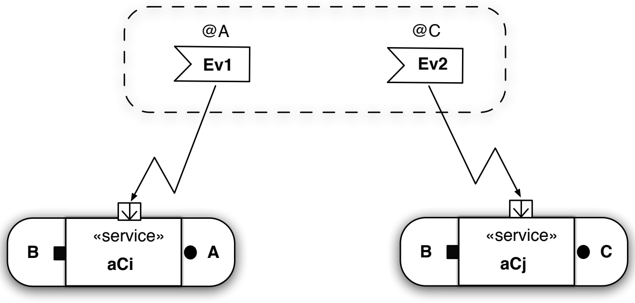


Figure 2.14: Realizability of interruptible activity regions

2.4 Flow-global to Flow-localized Choreography

A Flow-global choreography can be transformed to a flow-localized choreography using a set of transformation rules [KB10], called localization rules. These localization rules can be classified into the following three categories: 1) Pin localization, 2) Direct flow localization, and 3) Indirect flow localization with intermediate control nodes.

Note that we here are talking about localization in flow-localized choreography, not orchestration. Therefore responding flows are not discussed here.

2.4.1 Pins Localization

In a top-down approach (flow-global to flow-localized) pins are localized from the flow-global choreography actions. The pins in a flow-global choreography model are localized such that *starting* pins are localized to *initiating* roles, *terminating* pins are localized to *terminating* roles. Streaming pins must be assigned to roles such that output pins are assigned to the role that generates the output token, and input pins assigned to the role that receives the input token.

In a bottom-up approach, pins are already given in flow-localized choreography of elementary collaborations and only reflected in the flow-global choreography.

2.4.2 Direct Flow Localization

One assumes that causality has been determined on the global-flow (c.f. Section 2.3.2).

In the case of direct global flow $aC_1;aC_2$, the general rule is to localize the flow to the initiating role of aC_2.

If the direct-flow has the {strong} causality property, localize the flow to the initiating role of aC_2 and the terminating role of aC_1 (both roles are local to the same component). A {strong} flow from the *q.Quiz Service* to the *i.Information Service* in the walking tour service in Figure 2.6 will therefore be localized to *poi* role which is the initiating role of *q.Quiz Service* and the terminating role of *i.Information Service.*

If the direct-flow has the {non-causal} property, it can be localized: *(1)* using *interaction flows* as shown in Figure 2.15(b); or *(2)* using coordinating collaborations, as shown in Figure 2.15(c), so that flows can be assured to have at least weak causality; or *(3)* by modifying the flow-global specification so that the flow can be localized.

If the direct-flow has the {weak} causality property, there are two possibilities for localization: (*1*) if the flow is directly realizable according to *Propositions 1* and *2* then the flow can be localized to the initiating role of aC_2 and the non-terminating role of aC_1; or (*2*) if the flow is not directly realizable according to *Propositions 1* and *2* then insert interactions for enforced strong sequencing [KB10].

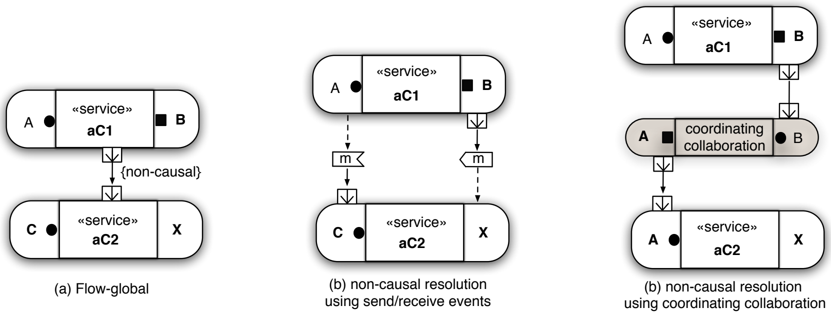

A
«service»
aC1
B
{non-causal}
C
«service»
aC2
X
(a) Flow-global
A
«service»
aC1
B
m
m
C
«service»
aC2
X
(b) non-causal resolution using send/receive events
A
«service»
aC1
B
A
coordinating collaboration
B
A
«service»
aC2
X
(b) non-causal resolution using coordinating collaboration

Figure 2.15: Non-causal sequencing flow and localization

2.4.3 Control Nodes and Path Localization

In the case of indirect flows, one needs to localize all the intermediate control nodes between action nodes aC_i and aC_j. We say that two action nodes

aC_i and aC_j are linked by a *flow-path* from aC_i to aC_j through a number of intermediate control nodes each linked by direct *flow-steps.*

In the following, we represent a *flow-path* as follows:

path ::= $\{$*** | *srcId*$\}$$\rightarrow$*causalityClass*$\rightarrow$$\{$*** | *tarId*$\}$,

where *** represents a pseudo node such as an initial node, or activity final node; *causalityClass*:=$\{$*strong* | *weak* | *non-causal* | *non-applicable(na)*$\}$; and *srcId, tarId::= collaborationUseId.roleUseId.*

In the walking tour service, shown in Figure 2.6, there are three flow-paths through control nodes F_1 and D_1:

P4:= i.user$\rightarrow$*weak*$\rightarrow$p.server; (through D_1)

P5:= i.user$\rightarrow$*na*$\rightarrow$*; (through D_1)

P6:= *$\rightarrow$*na*$\rightarrow$l.user; (through F_1)

P7:= *$\rightarrow$*na*$\rightarrow$p.server. (through F_1)

Localization of intermediate control nodes and indirect flow-paths is performed as follows:

1. *Each flow-path from aC_i to aC_j is first localized to the initiating role of aC_j.*

   For example P_4 and P_7 may be localized to the *server* and similarly, P_6 may be localized to the *user.* This rule does not apply to the flow-paths ending on $*$, such as P_5.

2. *Find all the flow-paths through each intermediate control node.*

   In the given example, there are two flow-paths at F_1 i.e., paths P_6 and P_7; and there is one flow-path at D_1 i.e., paths P_4.

3. *If all paths through a node are local to the same component, localize the node to that component.*

   In the given example, D_1 can be localized to *server* as the only path through it i.e. P_4 is local to *server.*

4. *If the paths through a node are local to different component, the control node may not be directly realized in either component. This is the indication of potential realization problems, and is discussed in Section 2.3 and papers [KB10, HKLB11, KB11a]. The composer may be asked to re-check and modify the choreography model if necessary. If the choreography model is as intended, chose a $\{$strong$\}$ path for the node localization if possible.*

   As a consequence, one needs to resolve associated {weak} or {non-causal} paths or flow-steps using interactions as explained in Section 2.4.2.

In the cases where all the paths through a node have the same causality properties, the node can be localized to either path (i.e. target role of the path) and the remaining paths or flow-steps are resolved using interactions explained in Section 2.4.2. For example, in the given scenario, one can localize F_1 to the *user* and use an interaction flow from F_1 to *p.Tour Planner* or the other way around.

5. *If the node is a choice node, it must be localized to the target component of the path and it must be possible to make the choice based on information local to that component.*

   In the given example, the decision node D_1 is localized to the *server.* If there are two or more flow-paths through a decision node that should be localized to different components, then that becomes a *non-local choice* and a proper resolution strategy should be applied (some are discussed in Section 2.3).

6. *If some flow-steps are {non-causal}, the flow-step can be made {strong} or {weak} using interaction flows, as discussed in Section 2.4.2.*

   In the given example, D_1 is localized to the *server* and the flow-step between the *i.Information Service* and the decision node D_1 can be resolved using {weak} sequence localization as this will lead to no message overlapping.

7. *If localization according to steps 3, 4, 5 and 6 above is impossible, then there is a realization problem to resolve.*

In any case, a control node should be localized to only one role, otherwise this means an indication of realizability problem, according to the *Proposition 1...4* presented in Section 2.3.

2.4.4 Localization of Other Elements

A choreography model also contains other model elements such as local actions, initial nodes, final nodes, interruptible regions and interruptible events. These elements are localized as following:

- *Local actions should be assigned (by composer) to a particular role in a flow-localized model.* In the example in Figure 2.6, the local action *p.Tour Planner* is assigned to the *server.*

- *Initial nodes will be localized to the role where the immediate target control node or pin is localized.* In the example in Figure 2.6, the initial node is localized to the *user* where the fork node F_1 is localized.

- *Activity final nodes and flow final nodes will be localized to the role where the immediate source control node or pin is localized.* In the given example, both activity final nodes are localized to *server*: one is connected by a flow-step to the terminating pin of *server* role in *Location Service*, and the other is connected by a flow-step to the decision node D_1 that is localized to *server.*

- *Interrupting events in the interruptible activity regions should be assigned to the initiating component.*

  If interrupting events in an interruptible region are localized to more than one component, there may arise a situation of *mixed-initiatives* that needs to be resolved.

2.5 Orchestration Models

The components or orchestrators are objects that have complete local behavior and run locally on a physical device while services involve interactions among components. Components may therefore participate (providing, requiring or just participating) in different services in different contexts. Orchestrators are based on collaborative services and contains:

- *Local activity flows* describing their local behavior.

- *Interfaces* by which a component interact with other components or the environment. Two types of interfaces are considered: *local interfaces* and *semantic interfaces.* Local interfaces are token passing interfaces represented by pins at the component boundary. Semantic interfaces correspond to the collaborations a component participates in, and describes the observable behavior at the external interfaces [KLB11].

2.5.1 Component Synthesis from Elementary Services

A direct realization as discussed in Section 2.3.1 results in local role orchestrations corresponding to the services where a component participates. The synthesized role orchestrations have local interfaces in terms of pins, internal behavior described by activity flows, and semantic interfaces that encapsulate interactions with other role orchestrations as illustrated in Figure 2.16 for the *location service.*

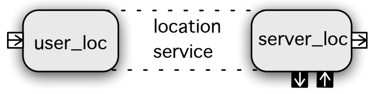


Figure 2.16: Semantic interface representation of location service

2.5.2 Component Synthesis from Composite Services

We first assume that the composite service is defined by one collaboration and choreography. We synthesize each of the composite roles (also called *component types*) of a composite service separately using the direct realization principles. The synthesis involves the following steps [KLB10]: (1) mark the roles of a component type defined by the enclosing collaboration; (2) keep collaboration roles, local actions, control nodes and the flows that are local to the component; (3) resolve interaction flows; (4) derive external dependencies in terms of responding flows; and (5) apply composition pattern to the services if necessary. In the following, each of these steps will be briefly discussed:

1. *Mark the roles CT_R of a component type CT.*

   A component type contains a set of roles which are selected from sub-collaborations C_i in such a way that each role in C_i is bound to a component type CT defined by the enclosing collaboration. For example in a component type *user_walkingtour* in a *Walking Tour* service, $CT_R = \{user\_loc, user\_q, user\_inf\}$. Note that collaborations associated with selected roles are used as *semantic interfaces.*

2. *Keep the local actions and local flows associated with the component type CT.*

   (a) Keep each local actions LA_i and control nodes CN_i which are localized to CT. This is straight forward because in the flow-localized choreography these entities are bound to specific roles.

   (b) Keep each localized control flow $FL(x-y)_i$ where the source and target nodes, i.e., x and y, of a flow are localized to CT, i.e., x,y $\in CT_R$.

   The resulting *user_walkingtour* after this step is shown in Figure 2.17 where one can see incomplete or inconsistent places in the component model. Note that the fork node (F_1) contains only one outgoing leg, and there are no flows between the service roles.

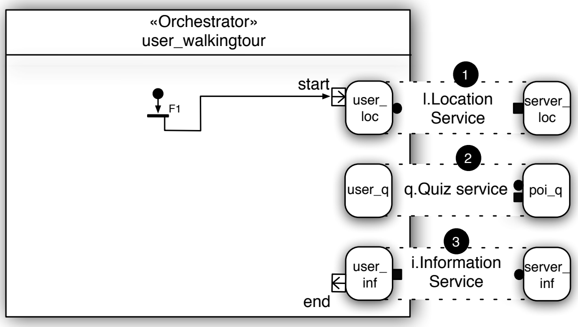


Figure 2.17: Component synthesis of a walking tour service after step 2

3. *Resolve the interaction (or cross-partition) flows.*

   For each interaction flows $FL(x-y)_i$ in the flow-localized choreography where the target of the flow is bound to $R_i \in CT_R$, there are the following possibilities:

   (a) If the flow is not directly realizable according to [KB10] enforce strong sequence using coordination messages for example using send and receive events as explained in Section 2.4.2. Note that such send and receive messages can be extended to the component boundary and represented as incoming streaming pin (for receive event) and outgoing streaming pin (for send events) as local interfaces to a component.

   (b) If the flow is directly realizable traverse backward until there is a collaboration C_j where a role of the collaboration $R_j \in CT_R$, then replace the flow with $FL(R_j - R_i)$ and mark the property of this flow as {weak}.

The resulting component model for *user_walkingtour* after this step is shown in Figure 2.18 where two interaction flows are resolved: from fork node (F_1) to *start.Tour Planner* streaming output pin; from terminating pin of *i.Information Service* to *next.poi* streaming output pin. These pins are used as *local interfaces* to a component.

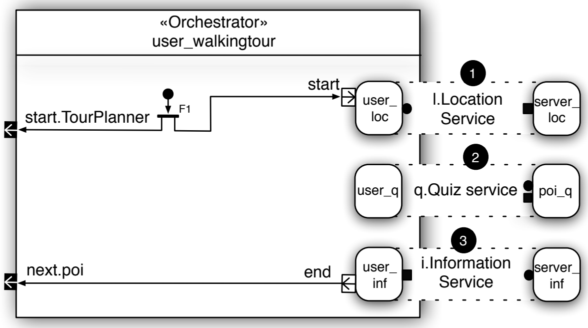


Figure 2.18: Component synthesis of a walking tour service after step 3

4. *Derive the external dependencies in terms of responding flows.*

   For each collaborations C_i selected in step 1 where *a collaboration role* $R_i \in CT_R$ is playing as non-initiating role (a responding role), identify the responding flows as following: In the flow-global choreography, traverse backward from R_i(consider all the possible backward flows if there are nodes like *merge or join* along the flow) to a preceded collaboration that has a role R_j belonging to component type CT, i.e., $R_j \in CT_R$. Then in the component type, add a responding flow from R_j to R_i, i.e., $FL(R_j - R_i)$. If R_j can not be found, add a responding flow from the initial node to R_i, i.e., $FL(N_{init} - R_i)$. If there are several such paths, use the path what allows earliest possible enable of R_i.

   The resulting *user_walkingtour* is shown in Figure 2.19 where responding flows are represented as dashed lines. The responding flow, for example from *l.Location Service* to *q.Quiz Service* specifies that the user component in the walking tour service should be ready to participate in quiz as soon as it finished playing its role in location service.

5. *Apply composition pattern to the services.*

   Not all of the service roles in a collaborative service, after step 4, may have the capacity to initiate or terminate the service via initiating or terminating pins. For example in Figure 2.19 there is no information about the termination of *user_loc*, initiation and termination of *user_q* and initiation of *user_inf.* This means, the *user_q* component will have no clue when the component should start and terminate. This creates

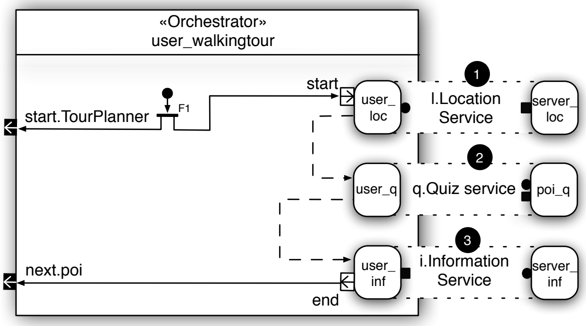


Figure 2.19: Component synthesis of a walking tour service after step 4

a problem, especially when a component is to be composed with other components (in this case with *user_loc* and *user_inf*). In order to solve these kinds of problems, we introduce a general composition enabling pattern that ensures that each service role has pins to coordinate enabling, start and termination of collaborative actions [KLB12].

The pattern added to the location service is illustrated Figure 2.20(a) with red colored bold flows. An abstract representation of the resulting *semantic interface* of the corresponding location service is given in Figure 2.20(b). This pattern provides the following benefits:

- The pre-defined functional behavior of the service is unchanged so that it can be used in other contexts where the pattern is not needed.
- The pattern can be applied whenever it is needed and the need to apply the pattern can be automatically detected during the composition of components.
- The pattern enables the typical object-oriented mechanism *registering to events* where components can register to the start and terminate events of other components.
- The pattern provides connection points for responding flows during component synthesis and composition.

The resulting *user_walkingtour* after this step is shown in Figure 2.21. Note that the applied pattern properly connects the responding flows by pins. For example the responding flow from the *Location Service*

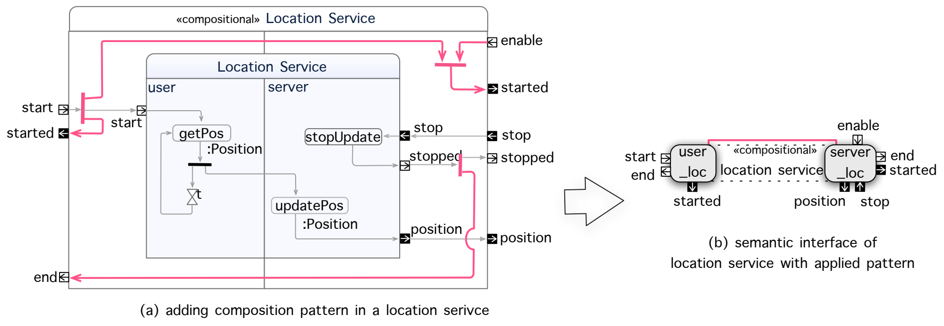


Figure 2.20: Adding composition pattern to location service

to *Quiz Service* is connected by the *started* and *enable* pins that are introduced after the pattern is applied.

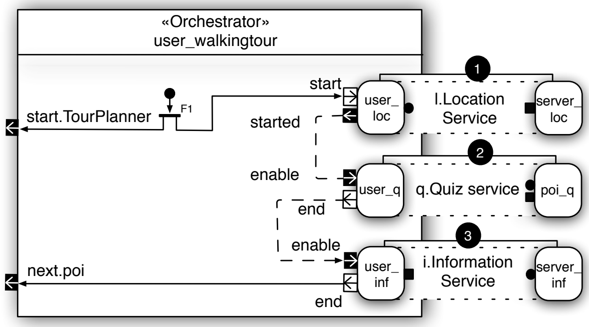


Figure 2.21: Component synthesis of a walking tour service after step 5

Similarly the resulting *server* component of the *Walking Tour* service is shown in Figure 2.22.

2.5.3 Component Representation

An abstract representation of a component (exemplified by the user component of the *Walking Tour* service) is shown in Figure 2.23. In Figure 2.23, the semantic interfaces are abstractly represented by a set $\{ns\}$ where n is a numeric type corresponding to an interface and s is a character having values: i representing that component plays initiating role; t representing

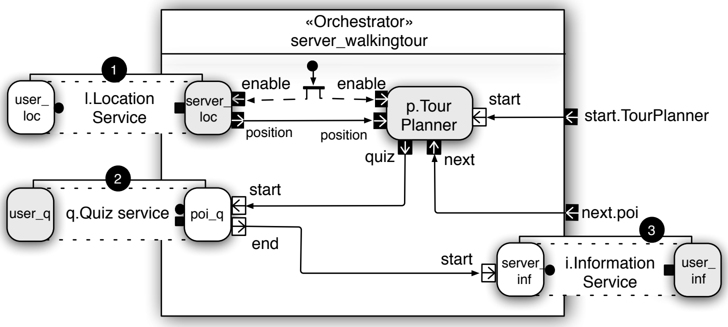


Figure 2.22: Server component of walking tour service

that component plays terminating role; and *p* representing that component plays participating role in a component.

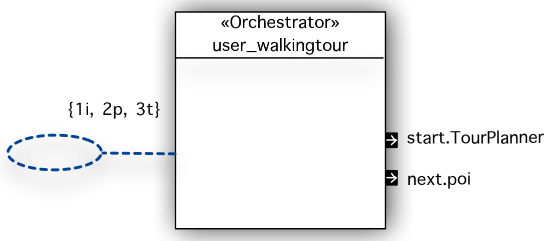


Figure 2.23: Abstract representation of user component of walking tour service

2.6 Component Composition

Sometimes, there is a library of components, and it is desirable that a composite component be composed from set of existing components. An additional motivation for this is that choreography is normally used to define services separately. Many systems can provide several services that will be invoked dynamically without an enclosing choreography to define their ordering. Service invocations may rather be controlled by user interactions. In these cases one need to compose components from service roles.

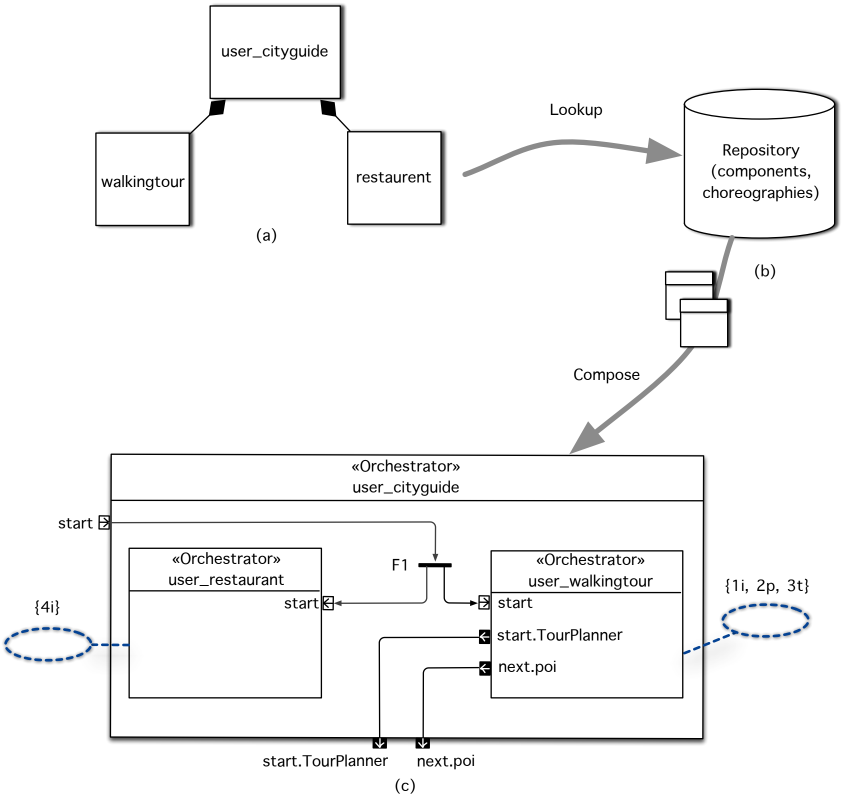


Figure 2.24: City guide user component as a composition of *user walking tour* and *user restaurant service* components

For example given the *user* components for the *Walking Tour* and *Restaurant* services, one may design a composite component that can play part in both services. This is done by composing parts of a component together via *local interfaces* i.e., pins at the component boundary. The following steps are suggested for this process in paper [KLB11].

1. *Specify the roles to be provided by a component and their part structure.* In this example, a user component will need the walking tour and restaurant services roles, and their part structure is specified as shown in Figure 2.24(a).

2. *Find components providing the roles* in a component library. This can be done manually by the composer based on the requirements at hand

or recommended/selected based on the information in a choreography model (if available) from which the roles have been synthesized.

3. *Compose the selected components* together using the pins i.e., the *local interfaces* of the selected roles. Pins may be connected automatically based on the data type that the pin owns or they can be connected manually. Some additional control nodes may be added to link the selected roles together such as the fork node F1 in Figure 2.24(c). Pins that are not connected to any other pins within the boundary of the component are extended to the component boundary. For example the *next.poi* and *start.Tour Planner* pins in the *user_cityguide* component in Figure 2.24(c) are extended to the component border.

It is important to note that the composed component can be put back into the library and potentially be reused in the another composition.

2.7 System Design

In system composition, systems are defined by composing components together to satisfy the specified requirements of the users. System composition differs from component composition in the following ways:

- System composition is the last composition step. This means that a system can not not be reused (or composed) in another system (or system composition)
- In system composition components are composed together via both *local* and *semantic* interfaces
- In system design, components are composed together not to make a new composite component but to make an executable system.

In simple cases where a system provides just an elementary service, the system design may be defined by a simple service model. In more complex cases one needs a separate system design activity where components are composed together, and the compatibility among the composed components are ensured by the following steps [KLB11]:

- By analyzing each semantic interface separately for internal consistency.
- By analyzing each component separately to make sure it is consistent with all its semantic interfaces.

- When two components are linked one need to check that the linked roles are complementing roles of the same semantic interface. This assures that the link is consistent, but not that the ordering of roles using different links are compatible.
- Ensuring compatibility in the ordering of roles by considering either the local ordering within each component against a global choreography, or by considering the responding flows of a component against the initiating flows of linked components.

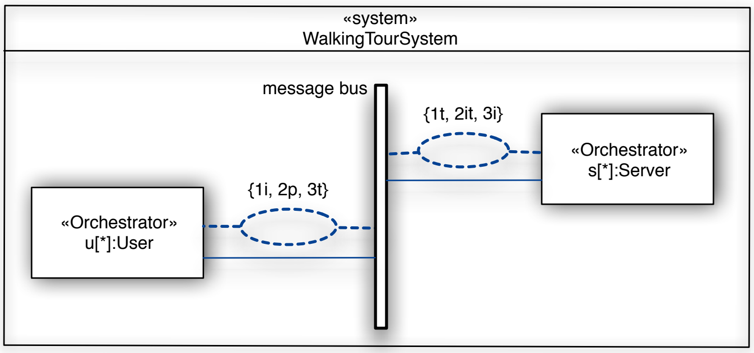


Figure 2.25: Walking tour system

A system design model of the *Walking Tour* system is shown in Figure 2.25. In a system there may be many instances of the same service running concurrently with roles assigned to different system components. There may also be many different types of services provided by the system. In general, the system components may play roles in different services, and different components may be able to play different combinations of roles. Consequently there is a difference between system design and service modeling. In system design one need to develop components that can play different combinations of roles, and one need to design the overall system as a composition of such components. If necessary the resulting system design will enable a dynamic structure where services are combined in ways that are not easily described in service models.

A system model also includes information about multiplicity and dynamicity i.e., information about the instances of components, how the services can be looked up, and service role instances can be created and managed.

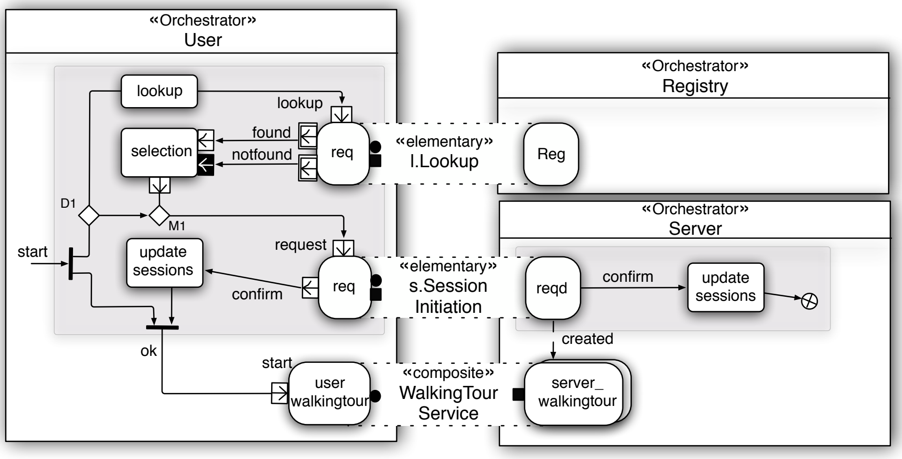


Figure 2.26: Lookup and session support in system design

For illustration purpose, we assume that many users use the *Walking Tour* service simultaneously. This means that there will be many sessions of *server* roles of the *Walking Tour* service at the server side, as shown in Figure 2.26. The figure also shows the mechanism for creating and managing instances of a service roles. Before requesting for a session setup, the requester component i.e., *user_walkingtour* should know whom to request for the session. We may use a generic *Lookup* service for that purpose. The generic *Registry* component contains the information about the registered services, associated roles, and relationship among the services. In this case, we assume that the *Registry* will return the address of a *Server* component that contains the requested role *server_walkingtour*. The *User* will then do session request using the *Session Initiation* service. Once a session setup is confirmed, the *Walking Tour* can be started and the role instances in the *Walking Tour* service will interact with each other. Figure 2.26 also shows how the service roles of the *Lookup* and *Session Initiation* services can be composed with a other service roles within the *User* and *Server* components.

Chapter 3

Results and Evaluation

The proposed methodology in Chapter 2 provides proper notations, mechanisms and tool support that are close to the domain and that hides tricky and detailed technical issues. It provides a basis where the building blocks correspond to collaborative activities understandable by non-technical people and therefore is closer to the mind of of domain experts such as teachers. In a nutshell, the summary of the results in this thesis is the following.

- Platform architecture concepts, their identification and implementation. This is indicated by numeric character "1" in Figure 3.1.
- Notations and mechanisms for the specification and design of services. This is indicated by numeric character "2" in Figure 3.1.
- Service composition mechanisms from the perspective of domain experts. This is indicated by numeric character "3" in Figure 3.1.
- Model transformation techniques from flow-global choreography to flow-localized models. This is indicated by numeric character "4" in Figure 3.1.
- Component synthesis techniques from choreography models. This is indicated by numeric character "5" in Figure 3.1.
- Composition of components to make composite components and system design models.

Using the above results, a domain expert can incrementally compose a composite service or a component from the smaller reusable services and components that are available in the library. The composer can then automatically generate components and application code. Note that a newly

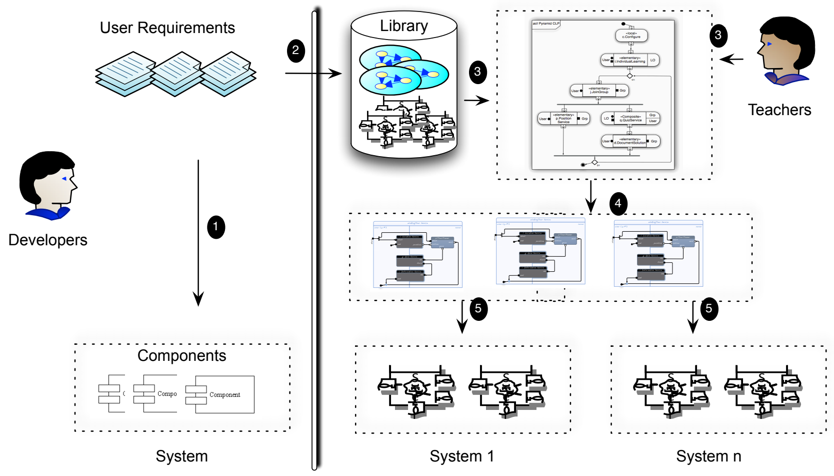


Figure 3.1: Evaluation context

designed service and component can also be put back into the library for later reuse in other applications.

The discussion and evaluation of each of these results is elaborated in the following sections.

3.1 Platform Architectural Concepts

As a first case study of a situated collaborative learning application a treasure hunt system was developed in paper 1 [KB09]. The innovative approach was to analyze the domain and identify stable domain concepts, and their variability. Stable domain concepts were mapped into active objects that jointly performed collaborative services, as explained in Chapter 1.

The active objects were modeled as state machines communicating using asynchronous message passing running on a Java platform that provided state machine and messaging support [BHM02]. The state machines were designed manually using the Ramses tool [Kra07].

The initial design was given as an assignment to about 20 student groups that developed their individual versions within a timeframe of approximately 40-100 hours using the same architectural principles. The system were later refined and implemented using Android for the mobile units.

As far as can be judged the architectural principles have proved satisfactory both regarding flexibility and ease of design. However the service and component development at that level was found to be not well suited for high-level service composers such as teachers and end users. This is mainly because the components were designed using state-machines which, we assume, are too technical for the end users. This triggered our quest for better service engineering methods

3.2 Notations and Mechanisms for Service Specification and Composition

The initial treasure hunt system used the idea of collaborations, but did not apply choreographies for global service behavior. We have found (in previous works [Cas08, Kra08]) that UML activity diagrams provides a simplified way of modeling global service behaviors. As a first experiment in this direction the treasure hunt choreography was modeled in flow-localized form using the Arctis tool [KSH09] as reported in paper 2 [KKB11]. Arctis provides support for specification of flow-localized models where collaborative service specification are specified using the swim-lane form of UML activity diagrams.

We have also found that the user-interfaces concerns of a system can be specified in a unified way [KKB11]. Note however that layouts of user-interfaces has to be defined using GUI editors such as the jFormDesigner for Java Swings and DroidDraw for Android devices.

The results were very promising, a city guide system was modeled and working application code were generated based on Android and the J2SE platform. This experiment inspired us to look for the ways: (a) to improve layout restrictions in the swim-lane form of UML activity diagram based service models, and (b) to add an abstraction layer on top of the service models supported in Arctis. Besides that we found that support for reusable component models was lacking in the Arctis tool since Arctis generates the full application system code directly from service models. These issues have been addressed in paper 3 [KB10], 4 [KLB10] and 6 [KLB11].

3.3 UML Profile and Flow-global Choreography

In order to support the proposed notations for flow-global choreography, a UML profile was implemented as explained in paper 8 [KLB12]. The UML profile was developed using standard meta-modeling tools as Eclipse plugins in order to facilitate integration with many UML tools including Arctis [Kra07].

Currently only the tree-view model of flow-global choreography is supported by the tool. This experiment shows the feasibility of flow-global choreography models. However, improvement is necessary in order to provide graphical editors for flow-global choreography notations so that domain experts or end users will really get the flavor of composing services in an abstract way.

This experiment also motivated us to look after the mechanisms for transforming flow-global choreography into Arctis models so that one can reuse the existing Arctis tool and its functionalities for model checking and code generation. This is addressed in paper 7 [HKLB11] and briefly discussed in the following.

3.4 Model Transformation Algorithm

Proof of concept demonstrator for model transformation algorithms has been partly implemented as an extension to Arctis [Kra07] using graph transformation techniques, as explained in paper 7 [HKLB11]. The proposed model transformation techniques transform flow-global choreography models into flow-localized choreography models. The transformed flow-localized models can be imported into the Arctis tool for further analysis, model checking and code generation.

It is also considered useful that the realizability of flow-global choreography (refer Section 2.3) can be checked during model transformations. This checking has been partly implemented as well - potential realizability problem can be detected and indicated to the composer so that suitable resolution strategy can be applied. Implementation is in a preliminary state, and therefore needs to be thoroughly tested and validated with more complex scenarios.

3.5 Proof of Concept Applications

Several case studies have been used to analyze and to validate various aspects of the service engineering approach:

1. *Treasure Hunt Game:* This case study was implemented for the evaluation of the architectural concepts and platform as mentioned in Section 3.1. The scenario is analyzed, and the basic and application specific services and components identified. Mechanisms are proposed to design and develop them. Implemented components are available in a repository for later reuse in other applications.

2. *City Guide Application:* This is in fact an extended form of the Treasure Hunt Game. The services and components identified in the treasure hunt game are reused. This case study is used for the following purposes:

    - To study and analyze the feasibility of developing user interface concerns using a similar approach as for service design, as explain in paper 2 [KKB11].
    - To illustrate the specification of choreography models, and model transformation from flow-global to flow-localized models as explained in paper 3 [KB10].
    - To partially illustrate the synthesis of components from flow-localized choreography as explained in paper 3 [KB10].
    - To analyze interrupting events and flows, streaming pins and flows, and cases of mixed-initiatives at the flow-global choreography models, as explained in paper 3 [KB10].
    - To study and design collaborative learning activities and collaborative learning patterns, as explain in paper 5 [KB11b].
    - To study and illustrate business service composition frameworks using the proposed method, as explained in paper 8 [KLB12].

3. *European Railway Traffic Management System (ERTMS):* This scenario is used:

    - To identify basic and application specific services and components in [KLB10, KLB11].
    - To explain a component synthesis algorithm from choreography models, as explained in paper 4 [KLB10].
    - To discuss the mechanisms for composition of components from inner components and a system design as a composition of components, as explained in paper 6 [KLB11].

4. *Teleconsultation System:* This case study is mainly used to discuss and analyze realizability issues of the choreography models, as explain in paper 9 [KB11a]. Realizability is discussed in terms of propositions and logical reasoning for problem identification and problem resolution.

3.6 Existing Tools and Further Directions

Various tools have been used for various purposes to analyze and implement techniques proposed in this thesis. The summary is given in the Table 3.1. According to our best knowledge, there is no existing tool that supports flow-global choreography specification, realizability analysis, and synthesis of reusable components and their composition.

In the Ramses tool, there is no explicit support of services. It provides a very convenient way of modeling components that are state-machine based, however the Arctis tool has support for flow-localized choreography specification and provides model checking and code generation. However it is intended for the domain experts who knows UML and system design as it is necessary to resolve realizability and coordination problems in these models.

Table 3.1: Existing tools used

	Ramses [Kra07]	**Arctis** [KSH09]
Service Modeling	no	yes
Component Modeling	state-machine based	no
F-G Chor Support	no	no
F-L Chor Support	no	yes (equivalent)
Realizability	no	no
Component Synthesis	no	yes (but not reusable)
Component Composition	no	no
Model Transformation	no	yes (from F-L to Orchestration)
Code Generation	yes (from component models)	yes (from F-L chor models)
Who is it for	developers who knows state-machines	domain experts who knows UML

Component composition mechanisms presented in this thesis are found to be implementable in the Arctis tool [KSH09]. Each service role or component can be modeled as a local building block that is composed in the usual Arctis way i.e. by connecting pins of Arctis building blocks. Given a manual mapping from flow-localized choreography to local orchestrations defined in this way using Arctis, existing techniques can be used for transforming such UML activity diagrams for orchestration into state machine based models and from there to application code. The *Walking Tour* service components (and their compositions) have been modeled and implemented in this way with application code automatically generated for Android platform. The generated application called *The City Guider* is publicly available in the Android marketplace.

Note that while implementing service roles as local building blocks in Arctis, cross-cutting service interactions are not directly visible in component models. They are hidden within *Call Operation Actions* and implemented in Java code and the implementation is based on asynchronous messaging. For distributed routing of messages, the *Actor Routing Protocol* [BHM02] is encapsulated in an *Actor Router* local building block in Arctis. In the present implementation, each local orchestration system contain an *Actor Router* building block along with other service roles. A generic *Registry* system is also implemented in order to support service registration, de-registration, and service lookup.

There is no existing tool support and no proof-of-concept demonstrators have been developed so far for the following:

- Component synthesis mechanisms presented in this thesis.
- Realizability check based on initiating and responding flows.

It is believed that tool support could be implemented with reasonable effort and time. Implementation of the component synthesis algorithm will be fairly straightforward as both choreography and component models are represented by UML activity diagrams.

Realizability check can be done during model transformation (from flow-global to flow-localized specifications). This means that the existing graph transformation based implementation can be extended to include realizability check based on initiating and responding flows.

Finally, in order to fully evaluate the proposed service engineering approach, the pieces of implemented tool demonstrators (as proof of concept applications) can be integrated together. We found that it is worthwhile to use or reuse functionalities of existing tools like Arctis whenever applicable while developing the proof-of-concept demonstrators discussed above. We are now considering to extend the Arctis tool by integrating all the proof concept applications that has been developed.

Chapter 4

Related Work

This chapter presents related research, technologies and platforms. The presentation is structured into four parts: (1) services and choreography modeling, (2) component synthesis, (3) realizability, and (4) platform and architectures.

4.1 Work on Service Modeling and Choreographies

4.1.1 Reactive Service Domain

In the domain of reactive systems it has been common to use interaction diagrams in one form or another to define interactions among components or participating entities in a service, for instance Sequence Diagrams (SD) and Message Sequence Charts (MSC). UML activity diagrams and interactions diagrams, both are well suited to capture the cross-cutting interaction pattern of services. Typically interaction diagrams specify partial or fragmented behaviors in a system. They show how objects communicate with each other in terms of a sequence of messages. On the other hand, UML activity diagrams are more complete with object flows and workflows without naming messages and use token flow semantics for the behavior specification (c.f. [KB10]). Thus UML activity diagrams seems better suited for complete behaviors and may be considered to be more high level than SD/MSC.

UML interaction overview diagrams (IOD), or high level MSC diagrams are used to define global behaviors in terms of references to interactions defined using SD or MSC [Dec09] and therefore may be used to specify flow-global choreography. As discussed in [KB10], UML activity diagram provides the following benefits:

- Compared to the IODs, activity diagrams allow the representation of roles as partitions of activities which is useful information that helps to understand and analyze the global behavior at the level of choreography.
- IODs excludes rich modeling constructs available in activity diagram such as interruptible regions, flow final nodes, and streaming nodes [Whi10].
- Streaming pins, available only in UML activity diagrams, are useful to model interactions among concurrent actions at a high level (c.f. [KB10, KLB10]).

In principle it is possible to use sequence diagrams for elementary collaborations and activity diagrams for choreography, since call behavior actions can call behavior defined by interactions. This has been done in [CBvB07], but exactly how to combine the two forms is not fully defined in UML. We have found activities to provide a good combination of readability, completeness, and automation potential. This combined with the benefit of staying within one notation are the main reasons for our decision to use activity diagrams throughout. Moreover, our choreography models are intended to be understandable to domain experts rather than programmers or software developers. We found that UML activity diagrams provide suitable abstract notations and mechanisms for composition compared to other approaches based on Use Case Maps [Buh98, Cas05] and state machines.

[Kra08] has been concentrated on choreography where all flows are localized to participating roles which is supported in the Arctis tool [KSH09]. Thus it supports flow-localized choreography. Therefore Arctis can provide a tool basis for analysis and state machine synthesis based on flow-localized choreography. Arctis uses the “swim-lane” form of UML activity diagrams to specify choreography and it currently generates complete systems and not component types that may be used in subsequent component based system composition at the modeling level. The flow-global choreography avoids the layout restriction of the swim-lane notation by representing the participating roles inside the actions [KB10, Cas08]. Our flow-localized choreography notation also overcomes the layout constraints imposed by swin-lanes, but is semantically equivalent.

4.1.2 Web Service Domain

Services in web service domain are considered to be provided by service providers upon the request from service consumers. Unlike web services, we consider general notion of service where two or more entities collaborate to achieve their goals. Collaborating entities are more proactive and may take their initiatives on their own.

There is a large body of work in the web service domain that deals with choreographies, their realizability and their mapping to orchestrations [BO05]. Some visual languages to mention are BPMN, BPEL, and WS-CDL.

BPMN is a non-executable, flow-chart based notation for defining business processes. BPEL and WS-CDL are XML-based languages for orchestration (i.e. focuses on the view of one business participant) and choreography (i.e. focus on global behavior) models. BPEL is executable while WS-CDL is not. There are some works for mapping BPMN, BPEL to UML and vice-versa using UML profiles. Note that UML Activity diagrams provide many of the workflow modeling constructs used in BPMN and other languages. UML provides rich constructs for specifying software-intensive systems and provides flexibility in modeling reactive software systems.

Emerging semantic web technologies use languages such as OWL-S, DAML-OIL and WSDL-S to add semantics (in the form of properties, constraints, meanings, policies, rules, or goals) to the services and their composition specifications. Most of these languages are text-based.

There are some formal approaches to choreography for service specifications, for instance labeled transition system in [KP06], activity traces in [QZCY07], set of conversations in [BGG$^+$06]. There is a body of work based on AI planning where composing services is seen as a planning problem and a sequence of services can be composed together to reach a business or service goal. Initial states and goal states are specified in the requirements by a service requester. A composition plan can then be formalized using finite state machines (FSM), situation calculus and Petri nets.

In general, service composition strategies in the literature are classified as static composition, semi-static and dynamic compositions depending on the time of composition and service binding. Static composition is also called design time composition as a service composer discovers, binds and assembles services during service development. In dynamic service composition, a composition plan is generated at runtime based on a service request. Our framework is concerned with design-time composition. However, we provide mechanisms to support dynamic discovery and linking as explained in Section 2.7. Our approach is based on model-driven development where a work-flow of business services and the behavior of each orchestrator is specified in a model using UML notations, in particular UML activities. We use collaborations to specify service structure. A web-service can be considered as a special case, a two-party collaboration in which is one is playing a requester (initiating) role and the other is responding (participating) role.

4.1.3 Graphical User Interface Aspects

Most service engineering approaches either ignore user interfaces concerns completely or treat them as secondary. Our approach allows both service logic and user interface(UI) concerns to be modeled in a unified way, c.f. [KKB11].

There is a body of work in a literature that discusses model-driven development of UI concerns in an application, for example UWE [BE08, KBHM00], WebML [CFB00], OO-H [GCP01], and MIDAS [CMV04]. A common and interesting feature of these methods is the modeling of the application in different orthogonal levels and aspects such as content, hypertext, composition and presentation modeling. Normally they use UML class diagrams for domain (content or data models) and presentation model are specified in terms of task models specifying the activity-flow model of the user actions. Most of this work either ignore service concerns or do not separate service logic from UI concerns. which means to limit their reusability.

Our models are focused on the event-driven behavior of user interfaces and allows code generation from these models using Arctis. The internal details of the services and UI blocks are encapsulated with their external behavior specified using external state machines (ESM). The ESM allows replacement of internal details while keeping the external behavior [Kra08]. Moreover, our building blocks are composed together by using different types of pins such as initiating, terminating and streaming pins linked together by arbitrary synchronizing logic.

We separate user interface and service concerns but compose in a unified way such that their composition can be verified and validated (automatically) early at the design time. Note that the layouts for graphical user interfaces need to be designed by hand, using for instance App Inventor for Android devices and Swing UI editors for normal Java platforms. Their association with corresponding functionality blocks is done manually using normal activity diagram flows.

4.2 Work on Component Synthesis

Unlike traditional approaches, we consider service models as partial models specifying the cross-cutting concerns where as component models are more complete behaviors from the perspective of a particular entity. Traditionally in the reactive systems domain, component models are defined using state-machines or some logical formulas. Our component models are based on UML activity diagrams and can be automatically synthesized from service models. Thus we adopt a unified approach for both service and component modeling.

Compared to earlier works on protocol synthesis from service models such as in [GvB90], we consider strong, weak and non-causality flows properties while only strong sequencing is considered in their work.

There is a body of work in the literature to synthesize components from service models [QZCY07, SB09, MH05]. They use the projection mechanisms similar to our approach. However there are differences:

- A part of a component behavior may depend on (or be enabled by) some external behaviors. This is considered in terms of "responding flows" in our component models whereas most of the existing work do not consider this.
- Most of the existing work assume manual derivation of components while we synthesize components where interfaces comes there with little or no extra cost (refer Section 2.5). We consider two types of interfaces to the components: local and semantic interfaces. Local interfaces are token passing interfaces. Semantic interfaces involve external message passing and can be considered as contracts among the collaborating entities. They also encapsulate observable behavior across the interface. Semantic interfaces can be reused as building blocks while composing heterogeneous components and be used to ensure that linked components are compatible.
- We generate reusable components composable in both the service and component dimension as discussed in Section 2.5 and 2.6.

4.3 Work on Realizability

The realizability of reactive system specifications, in general terms, was studied in [ALW89]. In the context of interaction diagrams and MSCs, the notion of realizability has been related to the notion of implied scenario, in [AHP96, AEY01, MFEH07]. [BBJ^{+}05] discusses automatic detection and resolution of semantic errors such as blocking, non-local pathologies, non-local ordering, and false-underspecification associated with scenario based requirements captured in the form of UML/MSC sequence diagrams.

In [CBvB07] the authors provide a classification of realizability problems and give some criteria for detecting them at the level of flow-global choreographies. They use an earlier version of the activity diagrams (AD) to define and analyze flow-global choreography. However in [CBB11, CBvB07], sequence diagrams (SD) are used to specify elementary collaborations. Compared to their work, our work, 1. use AD throughout, 2. allow streaming and interrupting flows, 3. introduce the concept of responding flows and role integrity for realizability analysis, 4. defines orchestrations in a way that enable subsequent composition in the system dimension as well as the service dimension.

To our best knowledge there is no other work that uses responding flows to systematically analyze realization problems. Nor is there any work that uses UML activity diagrams and includes both streaming and interrupting flows in choreography specifications. We use the concept of initiating and responding flows for realizability analysis. We believe the concept of responding flows is novel and that {weak} responding flows provides a new way to identify potential realization problem.

There are different techniques in the literature to check the realizability of choreography. In [SB09] behavioral equivalence between the LTS of a choreography and the LTS of a parallel composition of orchestrations is used. [BGG$^+$06, KP06] uses the bi-simulation equivalence for this. Trace equivalence is used by the authors in [QZCY07]. In our work we do not explore global state spaces, but provide rules for analyzing the choreography directly.

In nutshell, a summary of the related works presented above is given Table 4.1.

4.4 Architectures and Platforms for SL

The E-Learning Framework (ELF), the Open Knowledge Initiative (OKI), the IMS Abstract Framework (IAF), and the Open Mobile Abstract Framework (OMAF) are all defining service-oriented specifications and guidelines for the e/m-learning platforms and frameworks [DOL$^+$07]. They define the abstract representation of the services and component descriptions that comprise e/m-learning systems in the broadest sense. A major limitation in these specifications (and research initiatives like MOBIlearn [Web08]) is that they do not address the collaborative and social services to support situated collaborative learning activities.

IMS Learning Design [IMSa, IMSb] provides a framework to specify choreography of learning activities in terms of a work-flow of learning ac-

Table 4.1: Summary of the related works

	[KP06]	[BGG$^+$06]	[MH05]	[QZCY07]	[SB09]	Our Work
Partici-pant	roles	roles	services	roles	peers	roles
Partici-pant struc-ture	set of roles	set of roles		set of roles	UML comm. dia.	UML coll. dia.
F-L nota-tion			UML AD		UML comm. dia.	UML AD
Chor. seman-tic	LTS	set of conver-sation		activity traces	LTS	UML AD, [KH10]
Orches. nota-tion		process traces	UML AD, BEPL			UML AD
Orhes. seman-tics	LTS	process	UML AD, BEPL	activity traces	LOTOS process	UML AD seman.
Synthe-sis	no	no	no	yes	yes	yes
Confor-mance	bi-simula-tion			trace equiva-lence	behavior equiva-lence	by syn-thesis
Impl-menta-tion		no	yes	no	no	yes
Comm-unica-tion	async		sync, async	async	sync, async	async
Mixed Init-iative	yes	no	no	no	no	yes
Data	yes	no	yes	no	no	yes

tivities. It is an XML based specification. The creation of unit-of-learning (UoL) involves (besides the flow of activities) the bundling of all associated resources such as files, web references, learning materials, learning service configurations, etc.

Several authors have pointed out that IMS LD is insufficient to model collaborative learning activities, and accordingly proposed extensions to IMS LD, for example in [RRN04, HLVFAP^{+}06, MHHH05]. A major limitation of IMS LD is that it is not particularly well suited to model social groups, complicated control flows and various forms of social interactions. There are some IMS LD editors such as: RELOAD [rel], CopperAuthor [cop], CoSMoS [Mia05], and MOT+ editor [mot]. These editors provide text-based and graphical user interfaces to facilitate the specification of IMS LD based learning designs. Most of them presume that a learning designer has sufficient knowledge about IMS learning design constructs and specifications. Unlike most of these approaches, we use UML activity diagram which has intuitive and rich flow-constructs for modeling activities. For modeling collaborative learning activities, we encapsulate interaction among collaborating entities in collaborative building block and later these blocks can be composed together. This gives potential reusability compared to existing IMS LD based techniques, specially in composing collaborative activities together. Our approach also allows multiple individuals to interact together in collaborative activities. Summary of the comparison of our work and IMS learning design related works is shown in Table 4.2.

Unlike IMS LD based learning design, our approach provides the reusability of activity-flow models and units of learning. We also support the modeling of user-interfaces in a similar ways as services or activities. We provide intuitive graphical notations for services and their compositions as learning activity work-flow [KB11b]. This means that educational practitioners need to have the bases of workflows but need not be trained with formal semantics of IMS LD.

Various applications have been proposed and developed to support learning based on technologies such as agents, peer-to-peer communication, web services, and grid technologies. Some examples are: EMASPEL [ANA05], I-MINDS [SJA04], XESOP [MBB04], HYDRA [Zua05], Apple [JYY^{+}04], [TY08], MobiLP [CLWC03] and COLLAGE [HLVFAP^{+}06]. A discussion is given in [KB09]. Our approach is different from these approaches in the following aspects:

- *Automation* - In most of the applications mentioned above, components are manually designed from the given requirements. In our approach, platform components can be automatically synthesized from

Table 4.2: Comparative summary of our work and IMS learning design

	IMS Learning Design	**Our Approach**
Development Approach	Top down	Bottom up
Formal Semantics	no	Semi-formal
Notation	XML	UML activity diagram
Learning Activities	Local activities	Local and collaborative
User Interfaces	No	Yes
Reusability	No	Yes
Workflow Design	Sequencing of local activities	Sequencing of local and collaborative activities
Who is it for	Developers/Teachers trained to IMD LD	Teachers knowing basics of work flows

service models.

- *Resuability* - Unlike in most of the applications mentioned above, synthesized components in our approach are reusable i.e. they can be possibly reused in different applications.

- *Collaboration and situatedness* - Unlike our approach, collaboration and situation issues are not particularly in focus in the applications mentioned above.

- *Ontologies and intelligence support* - Unlike some of the approaches mentioned above, ontologies and intelligent reasoning capabilities are not supported in our components. It will be interesting to consider these issues in the future as a supplement to our approach.

Chapter 5

Concluding Remarks

5.1 Concluding Remark

This thesis answers the main research question "*how to rapidly and flexibly design and develop situated, collaborative services and components*", by presenting a methodology for the development of distributed collaborative services. The proposed methodology consists of mainly three types of models:

- *Structure models* which specify the structure of collaborating entities in services and systems. The structural models are specified using UML collaborations.
- *Choreography models* which specify the global behavior among collaborating entities in services and systems. Choreography is specified using UML activity diagrams with some extension via profiles. The choreography is provided at two levels of abstraction: *flow-global choreography* and *flow-localized choreography.*
- *Orchestration models* which specify the local behavior of each collaborating participant in a service or system. The orchestration is specified using UML activity diagrams. The orchestration (or component) models can be synthesized from choreography models and/or it can be composed manually from smaller building blocks.

The proposed methodology provides the following:

- It allows the specification and use of abstract service notations and provides flexible composition mechanisms by mapping stable domain concepts to active objects and cross-cutting behavior to services.

- It provides a notation for *flow-global* choreography specification and design.
- It provides principles and rules for analyzing the realizability of *flow-global* choreography specifications using the concepts of *initiating flows* and *responding flows.*
- It provides principles and rules for mapping *flow-global* specification to *flow-localized* specification.
- It provides mechanism for enabling component synthesis from choreography specifications.
- It provides mechanisms for composing components from smaller roles and components, and system design by composing components together.

Using these methods and mechanisms, domain experts can: (1) specify services and put them into a library, (2) design a composite service by composing reusable services from a library, (3) synthesize reusable components, (4) compose components together to make a composite component and system design, and (5) generate application code from service or system models using existing techniques. The evaluation of the proposed methodology is done using: (1) proof-of-concept demonstrators, (2) analysis of selected case studies, and (3) logical reasonings.

5.2 Future Works

The following issues are considered important in the further development, refinement and extension of the proposed service development methodology.

- *Refinement of transformation rules.*

  Model transformation rules should be implemented and analyzed in few scenarios. Experimentation with other case studies is needed for more extensive evaluation.

- *Realizability of choreographies.*

  The proposed mechanisms for realizability analysis of choreography models are not fully implemented yet. Further work is needed for their implementation.

  Realizability of streaming flows in flow-global choreography is not explored in this thesis. Inherently streaming flows implies concurrency between the connected service roles and it will be interesting to see how streaming flows effect the realizability of choreographies.

- *Graphical editors for flow-global choreography models.*

  A UML profile for flow-global choreography is implemented and tested only with tree-based UML editors. Implementation of the sophisticated graphical editors for flow-global choreography is planned in order to provide the full advantages of flow-global models to the composers i.e. easy and flexible composition mechanisms.

- *Implementation of component composition mechanisms.*

  The preliminary implementation of component composition mechanisms presented in this thesis is done using the Arctis tool. The implementation approach may be validated more extensively by implementing more use cases and scenarios.

- *On the evaluation and validations of the methodology.*

  Implementations and case studies of the methods proposed in this thesis demonstrate the potential and applicability rather than providing rigorous proof of the applicability. Some proof of concept applications and analysis on selected cases have been done. Realization and validation of the complete methodology with full tool support remains.

References

[ADH$^+$04] J.I. Asensio, Y.A. Dimitriadis, M. Heredia, A. Martinez, F.J. Alvarez, M.T. Blasco, and C.A. Osuna. Collaborative learning patterns: assisting the development of component-based cscl applications. In *Proceedings of 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 218–224, 2004.

[AEY01] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Realizability and verification of msc graphs. In *Proc. 28th Int. Colloq. on Automata, Languages, and Programming*, pages 797–808. Springer, 2001.

[AHP96] Rajeev Alur, Gerard Holzmann, and Doron Peled. An analyzer for message sequence charts. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 35–48. Springer Berlin / Heidelberg, 1996.

[ALW89] Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, pages 1–17, London, UK, 1989. Springer-Verlag.

[ANA05] M.B. Ammar, M. Neji, and A.M. Alimi. Emotional multiagents system for peer to peer e-learning (EMASPEL). *Proceedings of the 5th WSEAS International Conference on Distance Learning and Web Engineering*, pages 164–170, 2005.

[BBJ$^+$05] Paul Baker, Paul Bristow, Clive Jervis, David King, Robert Thomson, Bill Mitchell, and Simon Burton. Detecting and resolving semantic pathologies in uml sequence diagrams. In *Proceedings of the 10th European software engineering conference*, ESEC/FSE-13, pages 50–59, New York, NY, USA, 2005. ACM.

[BE08] Peter Braun and Ronny Eckhaus. Experiences on model-driven software development for mobile applications. In *Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 490–493, Washington, DC, USA, 2008. IEEE Computer Society.

[BGG$^+$06] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration conformance for system design. In *COORDINATION, volume 4038 of LNCS*, pages 63–81. Springer, 2006.

[BHM02] R. Bræk, K.E. Husa, and G. Melby. Serviceframe: Whitepaper. Erricsson Norac, 2002.

[BO05] M. Dumas Barros, A. and P. Oaks. A critical overview of web service choreography description language (WS-CDL). BPTrends, March 2005.

[Buh98] R. J. A. Buhr. Use case maps as architectural entities for complex systems. *IEEE Transactions on Software Engineering*, 24:1131–1155, 1998.

[Cas05] Humberto Nicolás Castejón. Synthesizing state-machine behaviour from UML collaborations and use case maps. In *SDL Forum*, pages 339–359, 2005.

[Cas08] Humberto Nicolás Castejón. *Collaborations in Service Engineering: Modeling, Analysis and Execution.* PhD thesis, Norwegian University of Science and Technology, 2008.

[CBB11] H.N. Castejón, G.v. Bochmann, and R. Bræk. On the realizability of collaborative services. *Software and Systems Modeling (accepted for publication)*, 2011.

[CBvB07] Humberto Nicolás Castejón, Rolv Bræk, and Gregor von Bochmann. Realizability of collaboration-based service specifications. In *Proceedings of the 14th Asia-Pacific Soft-*

ware Engineering Conference, pages 73–80, Washington, DC, USA, 2007. IEEE Computer Society.

[CD09] C. I. Canova and M. Divitini. Reflections on the role of technology in city-wide collaborative learning. *International Journal of Interactive Mobile Technologies (iJIM)*, 3, 2009.

[CFB00] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web modeling language (WebML): a modeling language for designing web sites. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 33:137–157, June 2000.

[CLWC03] Yuen-Yan Chan, Chi-Hong Leung, Albert K. W. Wu, and Suk-Ching Chan. MobiLP: A mobile learning platform for enhancing lifewide learning. In *ICALT*, page 457. IEEE Computer Society, 2003.

[CMV04] P. Caceres, E. Marcos, and B. Vela. A mda-based approach for web information system. In *Proceedings of Workshop in Software Model Engineering. Retrieved from: http://www.metamodel.com/wisme-2003*, 2004.

[cop] CopperAuthor Website, http://www.copperauthor.org/. Accessed on December 2010.

[Dec09] Gero Decker. Realizability of interaction models. In *1st Central-European Workshop on Services and their Composition*, 2009.

[DOL$^+$07] Declan Dagger, Alexander O'Connor, Samus Lawless, Eddie Walsh, and Vincent P. Wade. Service-oriented e-learning platforms: From monolithic systems to flexible services. *IEEE Internet Computing*, 11(3):28–35, 2007.

[Fab] Fabula project website. http://www.fabula.idi.ntnu.no/. Accessed on September 2007.

[GCP01] Jaime Gómez, Cristina Cachero, and Oscar Pastor. Conceptual modeling of device-independent web applications. *IEEE MultiMedia*, 8:26–39, April 2001.

[GvB90] Reinhard Gotzhein and Gregor von Bochmann. Deriving protocol specifications from service specifications including parameters. *ACM Transactions on Computer Systems*, 8:255–283, November 1990.

[HKLB11] Fenglin Han, Surya Bahadur Kathayat, Hien Nam Le, and Rolv Bræk. Towards choreography model transformation by graph transformation. In *Proc. of 2nd IEEE International Conference on Software Engineering and Service Sciences*, Beijing, China, 2011.

[HLVFAP$^+$06] D. Hernández-Leo, E.D. Villasclaras-Fernández, J.I. Asensio-Pérez, Y. Dimitriadis, I.M. Jorrín-Abellán, I. Ruiz-Requies, and B. Rubia-Avi. COLLAGE: A collaborative learning design editor based on patterns. *Educational Technology & Society*, 9(1):58–71, 2006.

[HMPR04] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[IMSa] IMS. IMS Learning Design Version 1.0 Final Specification. http://www.imsglobal.org/learningdesign/.

[IMSb] IMS. IMS Learning Design Best Practice and Implementation Guide. http://www.imsglobal.org/learningdesign/.

[JYY$^+$04] H. Jin, Z. Yin, X. Yang, F. Wang, J. Ma, H. Wang, and J. Yin. Apple: A novel p2p based e-learning environment. *Proc. of 6th International Workshop on Distributed Computing*, pages 27–30, 2004.

[KB09] Surya Bahadur Kathayat and Rolv Bræk. Platform support for situated collaborative learning. In *International Conference on Mobile, Hybrid, and On-line Learning, 2009. ELML'09.* IEEE Press, 2009.

[KB10] Surya Bahadur Kathayat and Rolv Bræk. From flow-global choreography to component types. In *System Analysis and Modeling (SAM)*, volume 6598 of *Lecture Notes in Computer Science*. Springer, 2010.

[KB11a] Surya Bahadur Kathayat and Rolv Bræk. Analyzing realizability of choreographies using initiating and responding flows. In *Proceeding of Model-Driven Engineering, Verification, and Validation (MoDeVVA)*. ACM Digital Library, 2011.

[KB11b] Surya Bahadur Kathayat and Rolv Bræk. Modeling collaborative learning services - a case study. In *Proc. of the 2011 International Conference on Collaboration Technologies and Systems*, Philadelphia, USA, 2011. IEEE Press.

[KBHM00] Nora Koch, Hubert Baumeister, Rolf Hennicker, and Luis M. Extending uml for modeling navigation and presentation in web applications. In *Proc. of the Workshop Modeling Web Applications in the UML*, 2000.

[KH10] Frank Kraemer and Peter Herrmann. Reactive semantics for distributed uml activities. In John Hatcliff and Elena Zucca, editors, *Formal Techniques for Distributed Systems*, volume 6117 of *Lecture Notes in Computer Science*, pages 17–31. Springer Berlin / Heidelberg, 2010.

[KKB11] Frank Alexander Kraemer, Surya Bahadur Kathayat, and Rolv Bræk. Unified modeling of service logic with user interfaces. *International Journal of Cooperative Information Systems (IJCIS)*, 20 (2):177–200, 2011.

[KLB10] Surya Bahadur Kathayat, Hien Nam Le, and Rolv Bræk. Automatic derivation of components from choreographies - a case study. In *International conference on Software Engineering*, Phuket, Thailand, 2010.

[KLB11] S. B. Kathayat, H. N. Le, and R. Bræk. A model-driven framework for component-based development. In *Proc. of the 15th International Conference on System Design Languages of the SDL Forum Society*. Springer, 2011.

[KLB12] Surya Bahadur Kathayat, Hien Nam Le, and Rolv Bræk. *Handbook on Reserach of E-Business Standards and Protocols: Documents, Data and Advanced Web Technolgies.*, chapter Collaboration-based Model-Driven Approach for Business Service Composition., pages 1–21. IGI Global, 2012.

[KP06] Raman Kazhamiakin and Marco Pistore. Choreography conformance analysis: Asynchronous communications and information alignment. In *Proceedings of the 3rd International Workshop on Web Services and Formal Methods.*, volume 4184 of *LNCS*, pages 227–241. Springer, 2006.

[Kra07] F.A. Kraemer. Arctis and ramses: Tool suites for rapid service engineering. In *Norsk Informatikkonferanse (NIK 2007). Tapir Akademisk Forlag 2007. ISBN 978-82-519-2272-2.*, pages 115–118, 2007.

[Kra08] Frank Alexander Kraemer. *Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks.* PhD thesis, Norwegian University of Science and Technology, 2008.

[KSH09] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann. Tool support for the rapid composition, analysis and implementation of reactive services. *Journal of Systems and Software*, 82(12):2068–2080, 2009.

[MBB04] Ivan Madjarov, Omar Boucelma, and Abdelkader Bétari. An agent- and service-oriented e-learning platform. In *ICWL*, pages 27–34, 2004.

[MFEH07] Abdolmajid Mousavi, Behrouz Far, Armin Eberlein, and Behrouz Heidari. Strong safe realizability of message sequence chart specifications. In Farhad Arbab and Marjan Sirjani, editors, *International Symposium on Fundamentals of Software Engineering*, volume 4767 of *Lecture Notes in Computer Science*, pages 334–349. Springer Berlin / Heidelberg, 2007.

[MH05] Jan Mendling and Michael Hafner. From inter-organizational workflows to process execution: Generating bpel from ws-cdl. In *OTM 2005, LNCS 3762*, pages 506–515. Springer, 2005.

[MHHH05] Y. Miao, K. Hoeksema, H. U. Hoppe, and A. Harrer. Cscl scripts: modelling features and potential use. In *Proceedings of the 2005 conference on Computer support for collaborative learning: learning 2005: the next 10 years!*, pages 423–432. International Society of the Learning Sciences, 2005.

[Mia05] Y. Miao. Cosmos: Facilitating learning designers to author units of learning using IMS LD. In *Proceeding of the 2005 conference on Towards Sustainable and Scalable Educational Innovations Informed by the Learning Sciences: Sharing Good Practices of Research, Experimentation and Innovation*, pages 275–282, Amsterdam, The Netherlands, 2005.

[mot] MOT+ Editor Website, http://www.licef.teluq.uqam.ca/. Accessed on January 2011.

[OMG09] OMG. Uml 2.2 superstructure specification, February 2009.

[PTRC08] Ken Peffers, Tuure Tuunanen, Marcus Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *J. Manage. Inf. Syst.*, 24(3):45–77, 2008.

[QZCY07] Zongyan Qiu, Xiangpeng Zhao, Chao Cai, and Hongli Yang. Towards the theoretical foundation of choreography. In *Proceedings of the 16th international conference on World Wide Web*, pages 973–982. ACM, 2007.

[rel] RELOAD Project Website, http://www.reload.ac.uk/. Accessed on February 2011.

[RRN04] Manuel Caeiro Rodriguez, Luis Anido Rifon, and Martin Llamas Nistal. Towards ims-ld extensions to actually support heterogeneous learning designs. a pattern-based approach. In *Proceedings of the IEEE International Conference on Advanced Learning Technologies*, pages 565–569, Washington, DC, USA, 2004. IEEE Computer Society.

[SB09] Gwen Salaün and Tevfik Bultan. Realizability of choreographies using process algebra encodings. In *Proceedings of the 7th International Conference on Integrated Formal Methods*, IFM '09, pages 167–182, Berlin, Heidelberg, 2009. Springer-Verlag.

[SCKB05] R.T. Sanders, H.N. Castejon, F.A. Kraemer, and R. Bræk. Using UML 2.0 collaborations for compositional service specification. *Lecture Notes in Computer Science*, 3713:460, 2005.

[SJA04] Leen-Kiat Soh, Hong Jiang, and Charles Ansorge. Agent-based cooperative learning: a proof-of-concept experiment. *SIGCSE Bull.*, 36(1):368–372, 2004.

[TY08] You-Tsung Tia and Meng-Chien Yang. Integrated platform for collaborative learning in mobile environment. In *Proceedings of the 2008 International Conference on Multimedia and Ubiquitous Engineering*, pages 258–262, Busan, Korea, 2008. IEEE Computer Society.

[Web08] Mobilearn Project Website, 2008. MOBIlearn Project: http://www.mobilearn.org. Accessed on February 2008.

[Whi10] Jon Whittle. Extending interaction overview diagrams with activity diagram constructs. *Software and Systems Modeling*, 9:203–224, 2010.

[Zua05] I.A Zualkernan. Hydra: A light-weight, scorm-based p2p e-learning architecture. *Proc. of 5th IEEE International Conference on Advanced Learning Technologies, 2005. ICALT*, pages 484–486, 2005.

Part II

Selected publications

Chapter 6

Paper 1

Title: Platform Support for Situated Collaborative Learning.

In: Proceeding of the International Conference on Mobile, Hybrid, and On-line Learning (eL&mL 2009), February 1-7, 2009. Cancun, Mexico.

Publisher: IEEE Computer Society.

Platform Support for Situated Collaborative Learning

Surya Bahadur Kathayat and Rolv Bræk
Norwegian University of Science and Technology (NTNU)
Department of Telematics
Trondheim, Norway
Email:{surya,rolv.braek}@item.ntnu.no

Abstract

In this paper, we consider the situated and collaborative learning domain, where students (as members of groups) go around the city and collaborate to learn. They are dependent on each other and need to collaborate to accomplish tasks. A treasure hunt game has been developed as a case study to help analyzing the domain and designing generic and flexible platform support for situate collaborative learning. The resulting platform seeks to support the domain as directly as possible by using agents to represent domain entities and providing services as collaboration among roles played by agents. The paper identifies stable and variable parts and explains how the necessary flexibility can be provided.

6.1 Introduction

The ongoing rapid change and development of technologies (Internet, mobile) is also changing the way learners learn. Learners are becoming more mobile, more distributed, community-oriented and are increasingly using technology to communicate, collaborate and acquire information. This creates new opportunities and challenges for learning platforms and learning technologies.

Mobile devices combined with location technologies enable what we call situated collaborative learning (SCL). In SCL learners work together on collaborative tasks outside the classroom, contributing their fair share and accessing situated learning objects i.e. depending on location and state.

It can be argued that better learning comes from collaborative interaction and situated exploration [PG00]. Emphasis is therefore given to supporting collaborative processes of knowledge construction and sharing enabled by seamless mobile networks and location awareness within a city. This work is motivated by key elements of situated and collaborative learning [JJH98] which are: positive interdependence among the learners, promotive interaction, individual accountability, interpersonal and social skills, situatedness, and group processing.

Our aim is to investigate platform solutions that can support the general needs of SCL and enable flexibility in adaptation and ease of introducing new learning services. To support SCL one needs something else than a traditional LMS. One needs a platform than can support mobile, situated and collaborative learning services. We have asked ourself how to best support SCL? We used a treasure hunt game to help answer this. Using domain analysis (Section 6.2), we have identified two types of services - basic support services and application specific services. Basic support services can be reused in many application specific services.

Several solutions have been proposed in the SCL area. We don't claim that our work is completely innovative in terms of the services it offers. Our main concerns have been innovative ideas and principles that makes the proposed platform more flexible and adaptable compared to several existing solutions discussed in Section 7.9. The work presented here aims towards next-[DOL$^+$07] or third-[LK01] generation of learning platforms which considers communities, collaboration, and social context as a center of attention.

The rest of the paper is organized as following. Description and analysis of the selected scenario is given in Section 6.2. Section 6.3 discusses the domain of SCL. Section 6.4 describes our agent based platform in detail. Experiments and evaluation of the platform are discussed in Section 6.5. Related works are described in Section 7.9 and followed by discussion in Section 6.7.

6.2 Case study - a treasure hunt game

As a case study of SCL we used a treasure hunt game.

Players logs on to the system using their handheld terminals and go around in the city. Players will be able to participate in several social/group activities. When the game starts each player will receive a clue to identify a "treasure", which can be any historical place, museum, or location within the city. They have to figure out which location and where it is by using knowledge they should have acquired. When they arrive at the right place, this is sensed by means of location technology (or alternatively participants will signal the system with a response which is dependent on their being at the correct location). A question-answer session is then established between the treasure and the player where players are asked questions about the treasure, something they can find out at location. Players can collaboratively answer the questions by interacting with group members during the question-answer session. When they (the group) have answered correctly, they are given another clue pointing to the next treasure. This continues until a group of players correctly finds all the treasures, answers all questions and arrives at the end of the game.

The game can be played in several modes. One possible way of playing is that all group members receive the same clues and questions at the same time and try to identify the same treasure. Another possible way is that group members receive different clues and try to identify different treasures, but may communicate to help each other. In either cases, players may track the position of other members of the group and may collaborate to find out the location and to answer the questions using group support functionalities (like collaborative interaction, group chat, instant messaging or discussion threads).

The main objective of the player is learning through participation, problem solving, and fun rather than defeating someone. Using situated learning material and making the stronger feeling of connectedness (with fellow learners), it motivates the players/students to learn by engaging them in a learning activity and immersing them into the material so that they learn more effectively. It also encourages the students to learn from their mistakes.

6.3 Problem Domain

A well known principle in system engineering is to start by analyzing the problem domain, rather than starting from specific technological solutions.

6.3.1 Domain concepts

In the SCL domain, one has the following stable domain concepts.

User - is one who uses a system. *User* can further be classified as *Teachers* and *Learners*. Users can have different profiles, privileges and data.

Group - is a set of *Users*. *Groups* can be classified as Workgroups, Socialgroups, etc. A group may contain other group(s).

MeetingPlace - provides functionalities to support group activities such as group chat, group discussion, etc. It is associated with a group sharing some common interest. It may be associated with a physical or virtual location.

LeaningObjects - any entity (digital or non-digital) that can be used for learning. A learning object may contain other learning objects. It can be classified as *Active*, or *Passive*. Active *LearningObjects* are objects with behavior for gaming, questioning, etc. Passive *LearningObjects* are some kind of documents which can be accessed and used by *Learners* for example web pages, pictures, videos, etc.

Locations - a location is a place having geographic boundaries and associated *LearningObjects*.

Class - class is a special kind of *Group* and may have associated *MeetingPlaces*. It has a teacher and manages the overall learning activities.

In the treasure hunt game, we use a configuration of these concepts and add few additional game specific ones.

Players - another name for *Learners*.

Treasures - kind of *LearningObjects* with associated *Locations*.

QuestionAnswerObject - kind of active *LearningObjects*. It contains information (questions and answers) about *Treasures*. *Players* can link to it and dynamically interact with it when they are in the vicinity of the *Treasures*.

GameManager - kind of *LearningObjects* which manages the flow of learning activities and associated learning objects (*Treasures, QuestionAnswerObject*).

6.3.2 Domain services

We have identified a number of services categorized into basic and application specific services listed in Table 10.1. A detailed description about the agents and roles associated with these services is given in Section 6.4.2. Several of these services are classified as active services according to [SB08]. They are not merely responding to user initiatives but takes initiatives towards the users as well. Information pushed when a user (or group member) reaches the location of specific *Treasures*, message from one user to another, time dependent notification etc are example of such initiatives.

Such services are not merely interfaces (like web services), but involve collaboration among several objects playing roles [BHM02] as illustrated in Figure 6.1. The IM service for instance is a collaboration among two users.

Table 6.1: Collaborative services and associated roles

Service description	Roles	Type
Group chat	GCp, GCg	Basic
IM	IMp	Basic
Group discussion	GDg, GDp	Basic
Basic awareness	PA, GA, GmA, CA	Basic
Location awareness	PosA, POp, GA	Basic
Clue	GmA, TrA, GGg, PGp	App
Scoring	GmA, GGg, QasA	App
Interactive QA	GGg, QasA, PGp	App
Configuration	TeA, CfA, CA, GmA	App

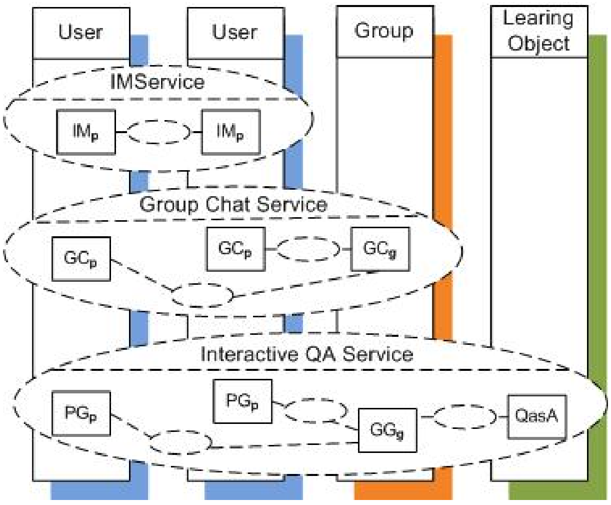


Figure 6.1: Cross cutting nature of services

Thus, a service may involve several domain objects and a domain object may participate in several services as shown in Figure 6.1. This is called the cross cutting nature of services by several authors. Domain objects normally have a much longer lifespan than individual service execution. Service sessions come and go while the domain objects stay. Domain objects are more persistent, have identities, associated data and profiles/preferences that services must relate and adapt to.

6.3.3 Variability

We may now classify the domain concepts and services into those that are general and independent of learning activities (users, groups, class, etc) and those that are specific to learning activities (treasures, question-answer objects, etc). There are two main dimensions of variability as shown in Table 6.2:

Configurating general concepts and services: There is need for a more or less constant adaptation to users and groups (their preferences and privileges). Therefore there must be powerful and user friendly means to adapt and setup new users (their preferences and privileges), groups, and meeting places. This is done by configuring the general domain entities such as users (number of users, their profiles), groups, meeting places and etc.

Developing new learning objects and services: A teacher should be able to easily create and configure new learning activities. In the general case this requires creating new learning objects with associated services and corresponding roles. In the treasure hunt case, a teacher can configure the new game with clues, treasures, and question-answers.

Table 6.2: Variability of domain concepts

Domain objects	General	Learning specific
Users	profiles, group membership	roles (from LO)
Groups	members, MP	roles (from LO)
Meeting place(MP)	IM, group chat, discussion	
Class	users, groups,	LO, loc
	MP	LO, loc
Location (loc)	MP	LO
Learning objects(LO)		loc, services

6.4 Distributed Agent based platform

6.4.1 Design principles

A well-known principle supported by much experience is to design systems in a way that closely models/reflects the domain they are serving. Thus we have chosen a platform design based on the following:

- representing the individual domain entities (introduced in section 6.3.1) by agents i.e. user agents, treasure agents, group agents, etc.
- representing services by roles played by agents.
- using a P2P communication architecture with asynchronous message passing which effectively handles active service and distribution.
- ensuring distribution transparency through logical addresses and flexible routing.
- supporting dynamic linking and dynamic deployment of agents/applications.

In [BF04], we have found agent orientation useful because it is centered around relatively stable domain concepts (users, groups, etc) and can support active, collaborative services. Most of the contemporary service-oriented and client-server learning platforms (found in the literature) are made for passive services and do not support active services very well. In many cases they are structured according to technology choices such as web services and not by domain reflection. Agent orientation also contributes to the clarity of modeling of stateful behaviors (where as server oriented architecture works best for truly stateless behavior) [BF04]. Agent and role behaviors are described using state machines. Agents may contain inner agents.

6.4.2 Outline of the platform

By analyzing the domain concepts and services that we have identified in Section 6.3.2, we identified the agent structure and the roles shown in Figure 6.2. A summary of services and roles is also listed in Table 10.1. The platform uses library of reusable agents and roles. Roles can be dynamically bound to the corresponding agents.

Player Agent (PA): A player (*Learner*) is represented by this agent. A GUI attached to this agent allows it to interact with the player. When the player logs on to the system, it register itself to the class (meeting

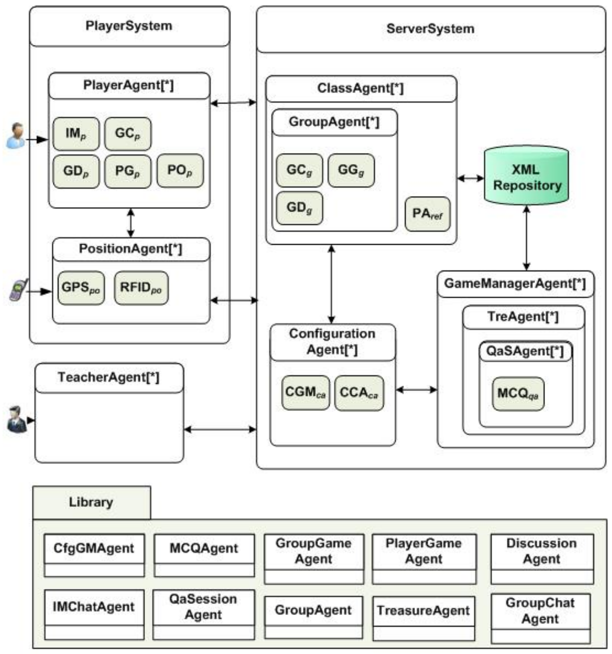


Figure 6.2: Overview of distributed agent based learning platform

place) and gets the references of available groups, games, other players in the class. Currently 5 reusable roles have been implemented: PGp(to play game), IMp(for instant messaging), PDg(to participate in group discussion), POp(for positioning), GCp(to participate in group chat). All of these roles are implemented independently and can be combined. Players for example can play the IM role during the game as well as participate in social groups and social activities. The player agent is associated with a position agent to get the position transparently to location technologies. Currently, only GPS positioning role is implemented in the position agent to read a GPS data. But it is easily extensible to incorporate other kinds of technologies like RFID, WLAN etc.

Group agent (GA): The group agent supports group related activities such as group chatting, group discussions, interactive question-answer, etc. It also keeps tracks the location and group related information of each players in a group. Currently three reusable roles have been implemented for the group agent. The GCg role supports group chat and the GDg supports group discussion. The GGg role supports players in a group to play the game. For instance it forwards clues and questions (about treasures) to the players. It creates and maintains a question-answer session with QA session agent (QasA). It also supports collaborative interaction within a question-answer session where each player selects their answer option with a certain confidence level that that is shared with other group members.

Teacher agent (TeA): A teacher agent can configure (add, modify, remove) players, groups, treasures, and questions in a game. A teacher agent cooperates with the configuration agent to configure different games and system components dynamically.

Class agent (CA): Class agent (CA) manages the groups and players within a class. It creates the group agents and associated the player agents (when it receives the registration signal after a successful login of a player) with groups.

Configuration agent (CfA): Configuration agent is used to configure all the system agents and their corresponding roles. Currently, two roles have been implemented - to configure a game manager (CGM) and to configure a class (CCA). A teacher agent uses this agent to configure system components (agents, roles, etc).

GameManager agent (GmA): The game manager agent is specific to an application, in this case the treasure hunt. It provides registry and management functionality for the dynamic associations in the game. When a new game starts, the game manager reads game configuration and creates necessary treasure agents and provide them with the questions and answers.

It sends join game invitation to the associated groups and members.

During the game, the game manager issues clues to the group agents which forward the clues to associated players. The game manager tracks the location of the the players and if a treasure is within a given distance, it returns a reference to the treasure agent to the player. The player agent will then set up a QA session (by sending a role request message to the treasure agent e.g IM session in Figure 6.3). During QA session, players collaboratively interact with the QA agent and group agent, and solve the learning problem (questions about treasure). When player(s) successfully have answered all the questions then QA agent will notify the game manager agent and terminate the QA session. The game manager then checks whether there are other treasures to be identified by the player (or group) and if so it will send the next clue to the players and next QA session will be established and so on. When all players in a group collaboratively have found all treasures answering all questions correctly, the game is finished and ranking information is sent to the group and corresponding players.

Treasure agent (TrA): The treasure agent is specific to the treasure hunt game and represents a treasure. It contains QA session agents, that will be dynamically created when a player is at the correct locations. QA agent will first load the set of questions and send these questions to the players. It will then receive the answers and verify the correctness of answers. When the group (players collaborating) correctly answers all questions, it will notify the game manager.

6.4.3 Development Process

For active and collaborative services, we use a *service-oriented* approach. We call it service-oriented because service models expressed using UML 2 collaborations are the primary artifacts. From the service models we derive the design models (agents and roles), expressed using state machines. The implementation code ready for deployment (in the form of OSGi bundles) is then derived using automatic code generation techniques, using our rapid service engineering tool - Ramses [Kra07].

6.4.3.1 Role creation and role binding

In order to manage dynamic links, we have used the role creation pattern as shown in Figure 6.3. A requester agent sends a *RoleRequest* message to the requested agent to play the requested role. If requested agent can play the role, the requested role is created and a *RoleConfirm* message is send back to the requester. This is shown for the IM service in Figure 6.3. Otherwise

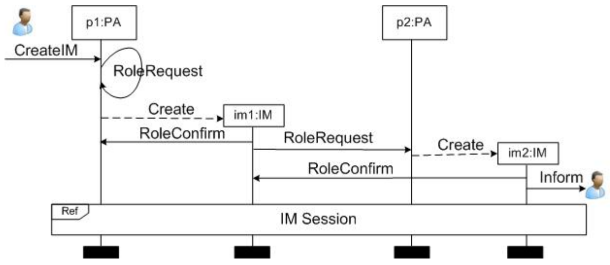


Figure 6.3: Role creations IM service

a *RoleDenied* message is send back to the requester. We assume that both players (p1 and p2) can play IM role. When the session is completed, user sends the *RoleRelease* message (not shown in Figure 6.3) to terminate the role. The role then terminates and the parent agent is informed with the *RolePlayEnded* signal. This mechanism also supports coordinating different roles being played by an agent.

6.4.3.2 Collaborative services and agents design

In order to explain the collaborative services and agent design principles, we consider the *interactive QA service* as described in Figure 6.4. The structure of the service is defined using the UML 2 collaboration diagram, identifying the roles. The behavior of each role is described using state machine diagrams. The goal (collaboratively answering questions about a treasure) of this service is achieved by the collaboration among 3 agents namely a player agent, a group agent and a QA session agent. More specifically in this service, the player agent plays a PGp role, the group agent plays GGg and the QA session agent plays *QasA*. This is a composite collaborative service as it includes two elementary collaborations - one between PGp role and GGg, and another between GGg and *QasA*.

The collaboration between QA session agent and the group agent is as following. When the session is established (between GGg and QasA), the session agent send the first question signal (*qsn* signal in Figure 6.4) to the group agent and enters into *waitForAns* state. When it receives a answer from the group agent (*smtAns* signal), it will check the correctness of the answer. If the answer is not correct, game info is updated (score will be deducted) and the same or another question will be send again. If the submitted answer is correct, some game information (like scoring, time taken, etc) is updated and it will check whether there is a next question

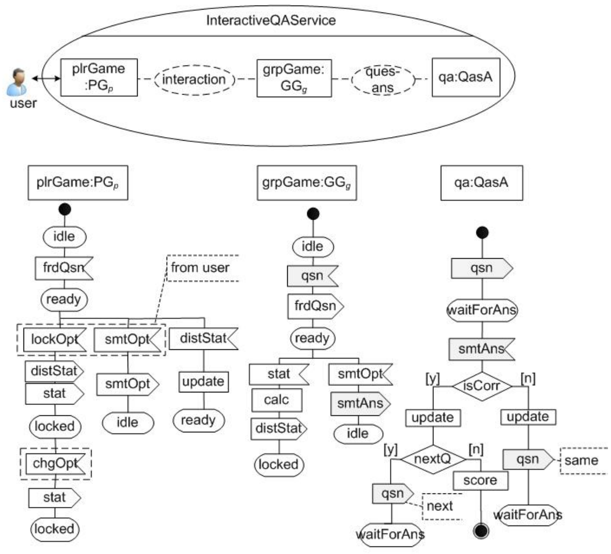


Figure 6.4: Overview of interactive QA service

available to be answered by a group. If next question is available it is then forwarded to the group. Otherwise a final score for this QA session is calculated and the game manager will be informed about it. The session then will be terminated.

Collaboration between group agent (GG*g*) and player agent (PG*p*) is as following. When the group agent received a question signal (*qsn* in Figure 6.4), it is forwarded to all the players within that group. When a player agent receives *qsn* signal, it will update the GUI and wait in the *ready* state. When a player selects an answer option with some confidence level (*lockOpt* signal), the player agent send the statistics (*stat* signal) to group agent. The player can also distribute the statistics to other group members (*distStat* signal). When group agent receives statistics (*stat* signal), it calculates the group statistics for this answer-option and broadcast back to all group members (*distStat* signal). The player (group leader) can see the statistics and submit the answer (*smtOpt* signal) to the group agent and wait in the *idle* state.

6.4.4 Execution and deployment

ActorFrame [BHM02] is a Java based framework that provides the support for agents, roles, sessions, p2p messaging, state machines, routing mechanisms, etc and is therefore used as a basic support. For the dynamic deployment support of agents (bundles and applications), the knopflerfish OSGi framework is used.

As shown in Figure 6.2, agents are grouped in to three types of OSGi application bundles, and can be dynamically deployed at distributed locations. Player agents and position agents are bundled into player system bundle. The server system bundle contains most of the server side agents like class agents, group agents, configuration agents, treasure agents, QA session agents, etc. Teacher system bundle consists of teacher agent. XML repository is used for storing player, group, game configurations.

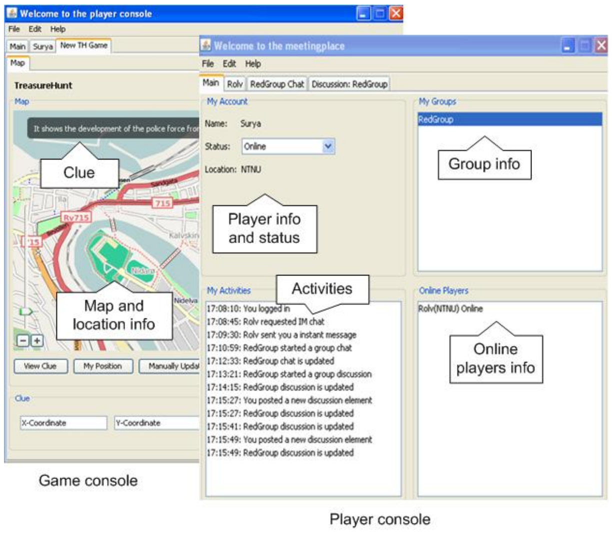
6.5 Experiment and Evaluation

The platform has been tested both in distributed and centralized configurations using both fixed and mobile (laptops) terminals within NTNU campus network to run player and teacher agents. HOLUX GPSlim240 Gps receiver and Google Maps (as interface) are used for the positioning.

Sample screen shots for the player and teacher agents are shown in Figure 6.5. Players can set their status, see the information about their group and other online players, and be aware of class activities. Players can even configure their awareness level. Players can flexibly collaborate using general meeting place functionalities like instant messaging, group chat, and group discussion. Basic meeting place functionalities can be combined with other collaborative services e.g. interactive QA service, clue services.

A teacher can flexibly setup and configure new learning objects; for instance assign clues and question-answers to treasures. A teacher can flexibly setup the game configuration. Players can flexibly join and play the game at the same time being part of other social groups. The players can have interactive clues and QA sessions during the game. When the game is started, players get a clue and try to identify the location of treasures. A player can see its location and the location of other players in Google map. When the players find the location of a treasure, they receive the questions. They can select an answer option with some confidence level, can see the group statistics, and can submit the answer.

The preliminary results are very promising. Despite a short development time (2 months) the system has sufficient functionalities to serve as demonstrator and proof of concept. We are now focusing on various issues (discussed in Section 6.7) in order to make it more robust and more flexible.



Name: Surya
Status: Online
Location: NTNU

```
17:08:10: You logged in
17:08:45: Rolv requested IM chat
17:09:30: Rolv sent you a instant message
17:10:59: RedGroup started a group chat
17:12:33: RedGroup chat is updated
17:13:21: RedGroup started a group discussion
17:14:15: RedGroup discussion is updated
17:15:27: You posted a new discussion element
17:15:27: RedGroup discussion is updated
17:15:41: RedGroup discussion is updated
17:15:49: You posted a new discussion element
17:15:49: RedGroup discussion is updated
```

Rolv(NTNU) Online

Figure 6.5: Sample screen shots

6.6 Related work

E-Learning Framework (ELF), Open Knowledge Initiative (OKI), IMS Abstract Framework (IAF), Open Mobile Abstract Framework (OMAF) are defining service-oriented based specifications and guidelines for the e/m-learning platforms and frameworks [DOL$^+$07]. MOBIlearn [Web08], Akogrimo [Web] are some service-oriented research initiatives. They define the abstract representation of the services and component descriptions that comprise e/m-learning systems in the broadest sense. We have considered their ideas of having basic (reusable) and application specific services in SOA based elearning platforms. A major limitation with such specifications (and research initiatives like MOBIlearn [Web08]) is that they don't address the collaborative and social services to support situated collaborative learning activities.

Various learning platforms based on agents, peer-to-peer communication, web services, grid technologies, etc have been proposed e.g. [ANA05], [SK04], [SJA04], [XYS03], [PBQH04], [MBB04], [Zua05], [JYY$^+$04], [TY08], [CLWC03] and [HLVFAP$^+$06]. I-MINDS [SJA04] has the support of two types of agents (teacher agent and student agent), however, this platform is inflexible and focused on a point solution for classroom based learning. Authors in [PBQH04] proposed an agent based learning platform where agents are represented as avatars corresponding to objects in the real world. Collaboration and situation issues are not addressed. An interesting point that can be considered from this system is that agents uses action planning mechanism in order to fulfill their objectives and goals. XESOP [MBB04], is a web-service and agent based learning platform. Course content is stored in XML databases and propagated via web services with the help of Helper agent (the only agent in the system). Limitation lies in its inflexibility. There is only one centralized agent that manages almost everything. There is no cooperation issues included among the learners.

HYDRA [Zua05] is a P2P based platform that facilitates searching, publishing, and downloading services for SCORM based learning objects. It does not address the collaboration issues. SCORM based learning object support may be considered for our future work. EMASPEL [ANA05] contains agents like interface agent, emotional agent, curriculum agent, tutor agent and etc. Agents in this case are intelligent agents (having reasoning capability) and communication among the agents is based of FIPA ACL. This system is able to recognize the current emotion of the learner based on the facial expressions and accordingly adapt the learning materials. This idea is promising to see and test in our situated collaborative learning scenarios.

6.7 Discussions and future improvements

We define collaborative services as collaborations among several agents where each agent can play several roles to achieve the service goal. Most of the learning services supported by existing learning platforms are based on single initiatives from the clients. Services are requested by client and a server respond to these requests and provides services or required information to the clients. This client-server technology is based on synchronous communication (such as J2EE, HTTP servers etc) and works well in many situation. However, there are fundamental limitations as well as scaling and capacity problems when the service is to support users that are collaborative on an equal basis and learning objects that take initiatives towards the user. Our proposed platform handles and supports the above mentioned issues well.

Most of the existing solutions are point solutions with limited support for situated, social and collaborative learning aspects. Our innovative approach has been to analyze the domain and identify stable domain concepts (as agents), and their variability. With the concepts of agents playing different roles in services, we provide the mechanisms to support active, situated and collaborative services based on asynchronous message passing. We have identified basic support services which can be reused in different contexts and applications. The treasure hunt is a first case we have chosen and implemented.

Based on the existing services/components, we believe, new SCL services like a city learning guide, or learning in museums can be easily developed and supported with our platform. In case of a city learning guide, different locations within a city (historical buildings, statues, etc) can be considered as learning objects. Learners will go around the city with positioning devices. Based on their location, a learning activity (some kind of question-answer) session will be established. The learners can collaboratively solve learning problems and share the knowledge interacting with collocated or distributed learners. In a museum learning case, different objects within a museum (e.g. tagged with RFID) can be considered as learning objects. Learners will visit the museum with RFID positioning device. When the learner reaches the vicinity of any learning object, new nearing services will be available and corresponding learning sessions will be established. Learners can interact and share their experience with other colocated/distributed learners within a group.

Our platform is in active development. We aim to refine and extend it considering the following issues: designing new SCL applications reusing existing basic services (components) and identifying new services and components; deployment of such services in handheld devices; adaptation and self-configuration of available or newly arrived components in the system; interfacing to some existing LMS, adding more intelligence to our agents for example negotiation, and reasoning capabilities.

Acknowledgment

We wish to thank NTNU and the FABULA project for funding this work. The FABULA team has been extremely valuable through numerous constructive discussions. Our special thanks to K. A. Martinsen, M. Knutsen, and G. C. Wiese for assisting in implementing the core of the platform.

References

[ANA05] M.B. Ammar, M. Neji, and A.M. Alimi. Emotional multi-agents system for peer to peer e-learning (EMASPEL). *Proceedings of the 5th WSEAS International Conference on Distance Learning and Web Engineering*, pages 164–170, 2005.

[BF04] R. Bræk and J. Floch. Ict convergence: Modeling issues. In *In System Analysis and Modeling (SAM), 4th International SDL and MSC Workshop, pages 237–256, Ottawa, Canada.*, 2004.

[BHM02] R. Bræk, K.E. Husa, and G. Melby. Serviceframe: Whitepaper. Erricsson Norac, 2002.

[CLWC03] Yuen-Yan Chan, Chi-Hong Leung, Albert K. W. Wu, and Suk-Ching Chan. MobiLP: A mobile learning platform for enhancing lifewide learning. In *ICALT*, page 457. IEEE Computer Society, 2003.

[DOL+07] Declan Dagger, Alexander O'Connor, Samus Lawless, Eddie Walsh, and Vincent P. Wade. Service-oriented e-learning platforms: From monolithic systems to flexible services. *IEEE Internet Computing*, 11(3):28–35, 2007.

[HLVFAP+06] D. Hernández-Leo, E.D. Villasclaras-Fernández, J.I. Asensio-Pérez, Y. Dimitriadis, I.M. Jorrín-Abellán, I. Ruiz-Requies, and B. Rubia-Avi. COLLAGE: A collaborative learning design editor based on patterns. *Educational Technology & Society*, 9(1):58–71, 2006.

[JJH98] D.W. Johnson, R.T. Johnson, and E.J. Holubec. *Cooperation in the classroom.* Interaction Book Company Edina, Minn, 1998.

[JYY$^+$04] H. Jin, Z. Yin, X. Yang, F. Wang, J. Ma, H. Wang, and J. Yin. Apple: A novel p2p based e-learning environment. *Proc. of 6th International Workshop on Distributed Computing*, pages 27–30, 2004.

[Kra07] F.A. Kraemer. Arctis and ramses: Tool suites for rapid service engineering. In *Norsk Informatikkonferanse (NIK 2007). Tapir Akademisk Forlag 2007. ISBN 978-82-519-2272-2.*, pages 115–118, 2007.

[LK01] J. Laister and A. Koubek. 3rd generation learning platforms requirements and motivation for collaborative learning. *EURODL*, 2001.

[MBB04] Ivan Madjarov, Omar Boucelma, and Abdelkader Bétari. An agent- and service-oriented e-learning platform. In *ICWL*, pages 27–34, 2004.

[PBQH04] Dorin Mircea Popovici, Cedric Buche, Ronan Querrec, and Fabrice Harrouet. An interactive agent-based learning environment for children. In *CW '04: Proceedings of the 2004 International Conference on Cyberworlds*, pages 233–240, Washington, DC, USA, 2004. IEEE Computer Society.

[PG00] M.J. Packer and J. Goicoechea. Sociocultural and constructivist theories of learning: Ontology, not just epistemology. *Educational Psychologist*, 35(4):227–241, 2000.

[SB08] Haldor Samset and Rolv Bræk. Dynamic service discovery using active lookup and registration. *services*, 0:545–552, 2008.

[SJA04] Leen-Kiat Soh, Hong Jiang, and Charles Ansorge. Agent-based cooperative learning: a proof-of-concept experiment. *SIGCSE Bull.*, 36(1):368–372, 2004.

[SK04] S.K. Sharma and F.L. Kitchens. Web services architecture for m-learning. *Electronic Journal on e-Learning Volume*, 2(1):203–216, 2004.

[TY08] You-Tsung Tia and Meng-Chien Yang. Integrated platform for collaborative learning in mobile environment. In *Proceedings of the 2008 International Conference on Multimedia and*

Ubiquitous Engineering, pages 258–262, Busan, Korea, 2008. IEEE Computer Society.

[Web] Akogrimo Project Website. Akogrimo Project: http://www.mobilegrids.org/. Accessed on February 2008.

[Web08] Mobilearn Project Website, 2008. MOBIlearn Project: http://www.mobilearn.org. Accessed on February 2008.

[XYS03] Z. Xu, Z. Yin, and AE Saddik. A web services oriented framework for dynamic e-learning systems. *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*, 2, 2003.

[Zua05] I.A Zualkernan. Hydra: A light-weight, scorm-based p2p e-learning architecture. *Proc. of 5th IEEE International Conference on Advanced Learning Technologies, 2005. ICALT*, pages 484–486, 2005.

Chapter 7

Paper 2

Title: Unified Modeling of Service Logic with User Interfaces.[1]

In: International Journal of Cooperative Information Systems (IJCIS). Volume: 20, Issue: 2 (June 2011).

Publisher: World Scientific Publishing.

[1]This is the extended version of the paper originally published in the proceeding of the first international workshop on Model driven service engineering (MoSE 2009). Published by ACM (2009)

Unified Modeling of Service Logic with User Interfaces

Frank Alexander Kraemer and **Surya Bahadur Kathayat** and Rolv Bræk Norwegian University of Science and Technology (NTNU)
Department of Telematics
Trondheim, Norway
Email:{kraemer,surya,rolv.braek}@item.ntnu.no

Abstract

We describe a method based on UML activities for the unified specification of collaborative service behavior and local user interfaces. The method enables a model-driven development process, which effectively combines the need to express service collaborations involving several components with the need to provide detailed operations for user interfaces. Our service models use activities as the primary building blocks that encapsulate self-contained functionalities. We show, how a complete distributed system can be decomposed into such building blocks, how this decomposition leads to a separation of user interface concerns from service collaboration concerns, and how they may be combined with an event-driven composition mechanism based on activity parameter nodes. We also demonstrate how different UI frameworks can be supported, and illustrate the method with a case study of a situated collaborative learning service.

Keywords: Model-Driven Development; Service-Oriented Architecture; User Interfaces; UML Activities; UML Collaborations.

7.1 Introduction

During the development of mobile services for end-users, a major challenge is the difference of perspective that some parts of the system demand. On the one hand, cross-cutting service behaviors need to be specified so that each component of the system may interact consistently with the other parts. A challenge here is the complexity that arises from the coordination of distributed behavior in general and asynchronous, multi-initiative peer-to-peer interaction in particular. On the other hand, end-users expect highly sophisticated and responsive user interfaces. Not only do these depend on the availability of certain libraries for user interfaces, like for instance Java Swing or Android, but they should also fit perfectly with the device they are executed on, to match screen size and input methods, for example. This implies a level of details that, using conventional modeling and programming techniques, is hard to combine with the cross-cutting view on service interactions needed to get the overall system right.

What unites these two perspective is their reactive nature; both user interfaces as well as distributed communication must react on events from an environment, that means user input or the reception of signals. Furthermore, they are tightly coupled by such events: User inputs may trigger communication with other devices, and, vice versa, the arrival of signals triggers changes in the user interface.

Our method, first described in [KH06] and complemented in [Kra08], therefore focuses on the specification of reactive behavior, how it may be encapsulated in the form of special building blocks, and how it may be composed effectively to achieve the desired overall system functionality. In this paper, we focus on the combination of collaborative behavior with local user interfaces, and introduce a separation between two kinds of building blocks:

- **Collaborative Service Blocks** model cross-cutting behavior among several components. The major concern with these building blocks is the specification of the coordination necessary to accomplish distributed tasks and communicate data.

- **User Interface Blocks** are executed locally on a device, and encapsulate all interactions from and to the users, as well as the detailed operations on user interface elements, like windows or buttons.

Both kinds of building blocks are equipped with parameter nodes that can be used to compose them together, either to pass data or to notify a block about an event detected by another block. Due to this composition style, the construction of system specifications resembles that of wiring together blocks. Further, we observed that the data types and events needed for composing blocks together, usually correspond to concepts from the problem domain, which makes specifications easier to understand by application developers familiar with a certain domain. The encapsulation of building blocks makes it possible that applications refer to abstract tasks (for example to provide user credentials), and that dedicated implementations may

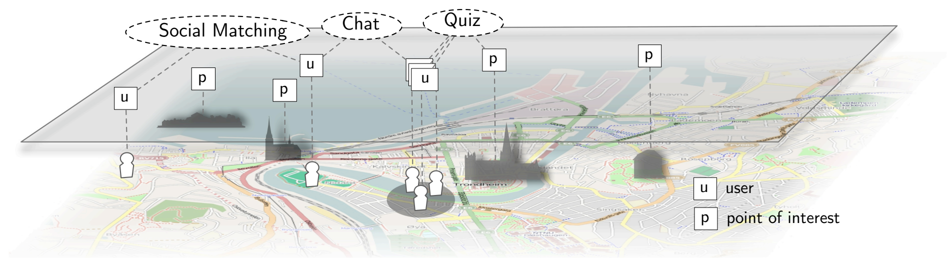


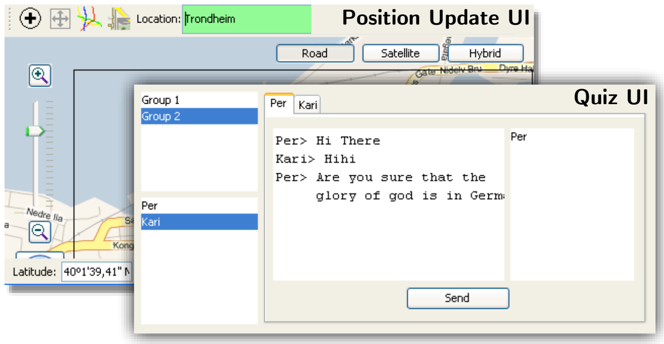
Figure 7.1: Mapping of domain objects to system entities and collaborations

realize this task in the best way for the corresponding platform. To exemplify this, we show two implementations of a UI building block realized for both Java Swing and Android. For the latter we detail a step-by-step method that integrates with the layout editor for user interfaces of the Android SDK.

The novelty of our approach is the way in which collaborative service logic is represented together with the local user interfaces: On the one hand, these concerns are separated into different specification units (the blocks), but on the other hand their events can be composed on the necessary level of detail to build responsive applications. Moreover, the use of building blocks leads to an incremental working process, since they can be developed and analyzed in isolation, and serve as interfaces between experts of different domains. These benefits are further discussed in Sect. 7.10. The rest of the paper is organized as follows: We describe the system that is subject of our case study in Sect. 7.2 and continue with a brief introduction to our engineering method in Sect. 7.3. Then, we describe how the system can be decomposed into subordinate building blocks in Sect. 7.4. The modeling of collaborative services by means of UML collaborations and activities is presented in Sect. 7.5, followed by the encapsulation of user interfaces in UML activities explained in Sect. 7.6. After that, Sect. 7.7 and 7.8 outline the automated verification and implementation of the specifications. In Sect. 7.9, we summarize and discuss related approaches and close in Sect. 7.10 with concluding remarks.

7.2 Case Study: Exploring the City

As an example, we specify a service for students to learn about different historical places in a city using mobile devices. It is based on the situated learning service introduced in [KB09]. The system tracks the position of



```
Per> Hi There
Kari> Hihi
Per> Are you sure that the
     glory of god is in Germ
```

Figure 7.2: User interfaces for the individual services

the students and responds when students with similar interests are closely located. Based on this feedback, users may create groups to cooperate. When a user comes close to a point of interest, a location-specific service can be started. Such a service can simply provide information and also engage students in learning activities. For our example, we realized an interactive quiz. During a quiz session, users are asked questions related to the particular point of interest. Users can interact with each other to answer the questions.

As illustrated in Fig. 7.1, the system is composed of objects reflecting domain entities, such as users, user groups, and different types of point of interests like museums. Between these objects, we identified the following services:

- A social matching service helps to create groups based on user interests.
- A position update service tracks the location of a user and makes it available to other users, for instance other members of the same group.
- A chat service allows the users to interact.
- A number of learning services associated with a particular point of interest are used to support the actual learning, in our case the quiz service.

Figure 7.2 shows two of the user interfaces involved in the realized system. For our application, we have built user interfaces in Java Swing for laptop computers, as well as user interfaces running on Android for mobile devices. There are considerable differences between these user interfaces, but the service logic is the same. Therefore it is desirable to separate these two concerns of the system such that they may later be composed. In the following sections, we describe our service engineering approach that enables a controlled composition of these concerns.

7.3 The SPACE Engineering Method

The main specification units in our engineering method are special building blocks that are expressed as UML models combined with Java code covering the details of operations. Building blocks may describe the local behavior executed by only one component, but they may also span across *several components* and describe *collaborations* among them. This flexibility is the key to the approach presented in this paper, since we will later use building blocks to encapsulate both local user interface behavior (as single-component blocks) as well as the collaborative service behavior involving several components (as collaborative blocks). Our building blocks have previously proven to enable a high degree of reuse (see [KH09] for a survey). Therefore, the development of a system starts with the consideration of libraries of reusable building blocks for different domains, as illustrated in Fig. 7.3. This leads to a drag-and-drop-like specification style, in which building blocks are composed to more comprehensive ones, until a complete system specification is obtained. Functionality not yet available is provided by new building blocks that may be stored for later reuse. Due to the formal semantics underlying our specification style, building blocks can be analyzed by means of model checking, which we explain further in Sect. 7.7. Once a system specification is complete and sound, it may be implemented in an automated process detailed in Sect. 7.8.

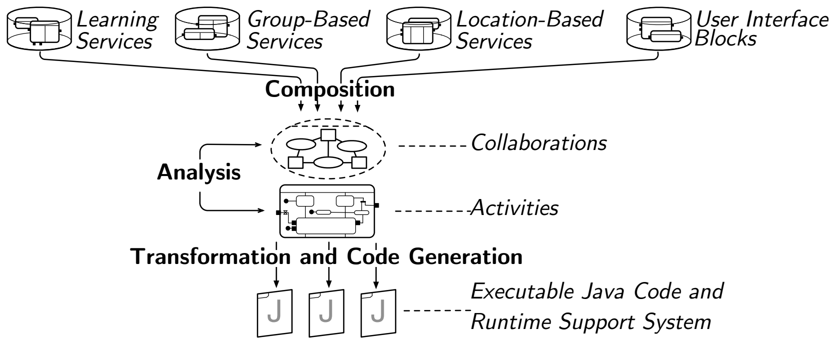


Figure 7.3: The SPACE engineering method

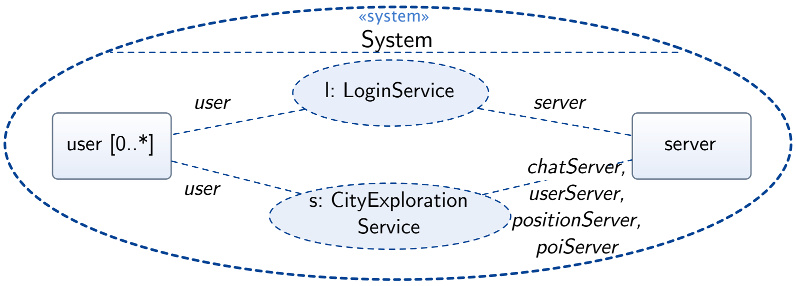


Figure 7.4: Collaboration for the complete system

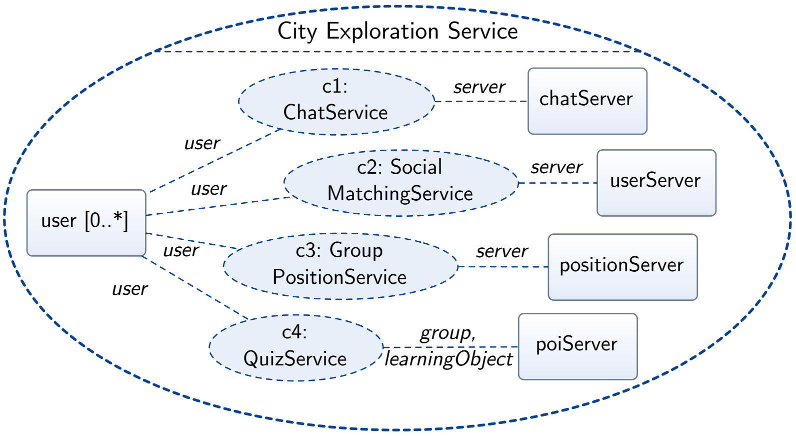


Figure 7.5: City Exploration Service Collaboration

To capture behavior involving several components, we use *collaborations* as major specification units. Fig. 7.4 shows the city exploration system on its highest decomposition level in the form of a UML collaboration. It consists of a number of users connected to a central server, represented by the rectangular collaboration roles. The ellipses between them refer to subordinate collaboration uses, namely *l* to a login service and *s* to the actual city exploration service. The latter is defined as a composition of collaboration uses, as shown in Fig. 7.5. The server is partitioned into four collaboration roles: a separate chat server, a user server, a position server and a point-of-interest server. The interactions of these servers with the users are in turn described by collaboration uses *c1...c4* for chat, social matching, group positioning and the quiz service. The labels on the lines that connect collaboration uses to collaboration roles are role bindings.

While the UML collaborations in Fig. 7.4 and 7.5 already document some aspects of the system structure and the distribution of responsibilities among the components, they do not specify any detailed behavior. The behavior of each UML collaboration is defined by a corresponding UML activity, shown in Fig. 7.6 for *System*. The activity contains one partition for each collaboration role, (i.e., *user* and *server* for the top-level system), which are usually assigned to different executable components. Each subordinate collaboration is represented by a call behavior action. For example, the login service *l* and the city exploration service *s* in Fig. 7.4 are represented by the call behavior actions *l* and *s* in Fig. 7.6. Note that they cross the partition borders since they are executed by users as well as a server. Lighter shaded blocks, such as *u: Login UI* refer to local building blocks, which are only represented in the activity perspective. Building blocks defined as UML activities have pins labeled by detailed events. These refer to the externally visible events that can be used to link blocks together, possibly inserting some control logic between them. The additional shadow at the server side of the login service indicates that the server handles several instances of them, as we detail later.

Users start their services by activating the user interface for the login, represented by block *u* that is started via initial pin *start* following the initial node. On this level, we are not interested in the internal details of the login user interface. Instead, it is sufficient to consider the external view provided by the special state machine in Fig. 7.7. It is a so-called external state machine (ESM), that defines the allowed sequences of tokens passing through pins. From this ESM we read that, after the login block has been started, it will emit a token via pin *login*. This pin is drawn filled, denoting that it is a streaming pin, which can pass tokens while

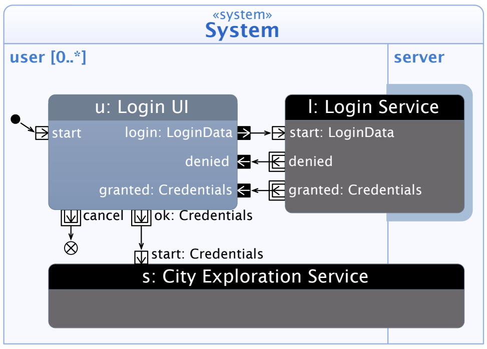


Figure 7.6: System activity

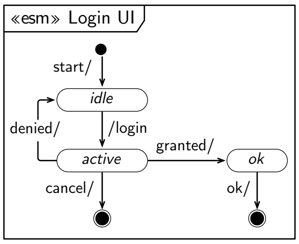


Figure 7.7: ESM of the Login UI block

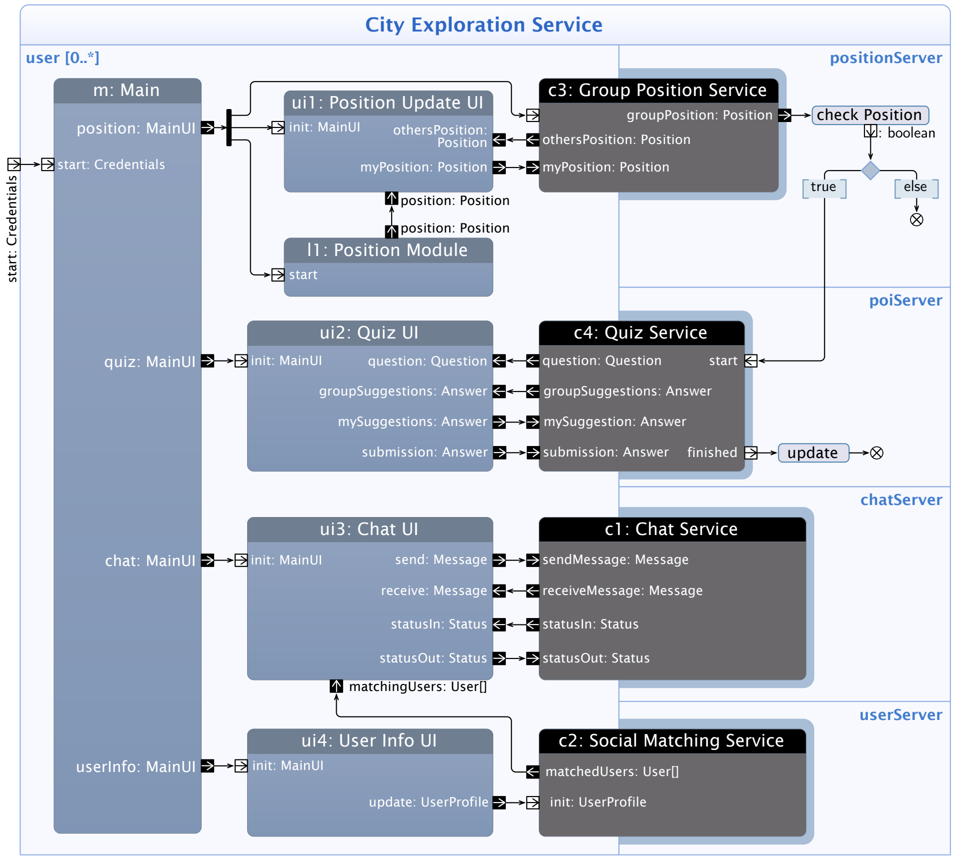


Figure 7.8: Activity for the City Exploration Service

an activity is active. The provided login data is used to start the actual login service as a collaboration with the server. As shown in Fig. 7.6, this collaborative building block has no streaming pins, and may only terminate in two alternative ways, represented by the two pins *granted* and *denied*. (These are mutually exclusive and belong to different parameter sets, noted by the additional box around them.) The corresponding result is fed back to the login user interface block, which will either forward a successful login via *ok* or eventually cancel. In the latter case, the flow simply ends in the flow final node. In case of success, the actual city exploration service is started via its pin *start*.

The decomposition of the login phase into a *Login UI* and a *Login Service* from Fig. 7.6 is a typical example for the general pattern that we ise for the separation of service logic from user interface logic. The local block *Login UI* encapsulates all logic specific for user interfaces, and the *Login Service* block models the collaborative behavior. We explain the internals of these blocks in Sect. 5 and 6.

7.4 System Decomposition

The decomposition of the city exploration service into its sub-services was already introduced in Fig. 7.5 with respect to the collaborations between the entities; Fig. 7.8 shows the corresponding decomposition in the activity diagram view, which also adds details about the coupling of events between the sub-services, and, as the main focus in this paper, the detailed user interface blocks. Each user interface part as the ones illustrated in Fig. 7.2 is represented by a separate UI block in the activity for the city exploration. In the Java Swing UI framework, these correspond to separate windows. For Android applications running on mobile phones, these correspond to different application screens, as we shall later see.

When a user successfully logs into the system, a user component of the city exploration service is started via starting node *start*, which activates the life-cycle controller of the city explore service *m: Main*, from which the other UI elements are controlled. At the beginning, the position module *l1* and the UI block for the position update UI are activated. The position module is continuously forwarding the position of the user to the UI via pin *position*. The group positioning service receives position updates from every user and forwards them to the other members of the group via the *othersPosition* streaming output pin. The Position Update UI then refreshes the position information of the user and other group members on the map.

The position server checks with operation *checkPosition* if any member

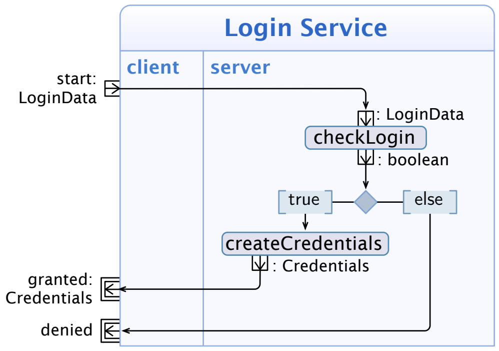


Figure 7.9: Activity for the Login Service

of a group is close to a point of interest. In such a case, it informs the point of interest server to start the quiz service. The Quiz service then forwards the question via its streaming outpin *question* to all members of the group (detailed in Sect. 7.5). During the quiz service, all the user can suggest an answer via *mySuggestion* streaming output pin of the *quiz UI* block which is connected to the streaming input pin *mySuggestion* of the quiz service. This suggestion is then forwarded to the other members of the group and group suggestions are updated to the quiz UI. One group member is assigned to be the group leader and submits the final answer via output pin *submission* of the quiz UI.

In parallel with the quiz and positioning services runs the social matching service. Users specify their interests and create profiles via *User Info UI*. The social matching service then returns the list of matched users through its output streaming pin *matchingUsers*. A list of matched users is also made available to the chat user interface component *Chat UI*, so that users can communicate with each other using chat service *c1*.

7.5 Collaborative Service Blocks

Fig. 7.9 shows the activity for the Login Service. It is activated by providing the login credentials via pin *start*, which are forwarded to the server role. There, the login data is checked, whereupon the service eventually terminates at the client side by either *granted* or *denied*.

The Quiz Service is shown in Fig. 7.10. Within one execution of the service, one learning object and one group object participate, as well as any number of users in a group, here emphasized by the multiplicity [0..∗]. Note that due to the role binding in Fig. 7.5, the groups and learning objects are

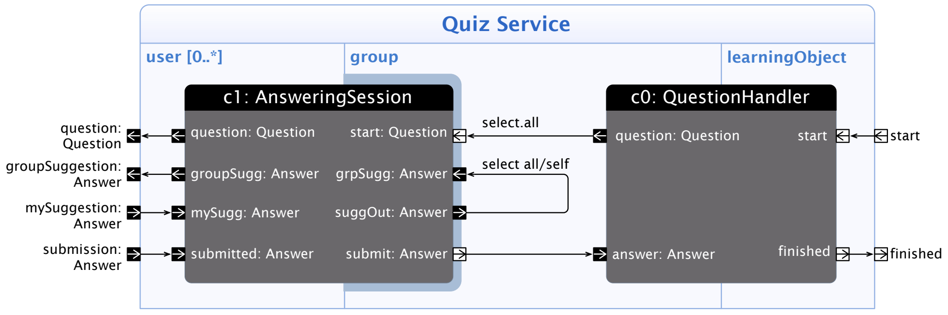


Figure 7.10: Quiz Service

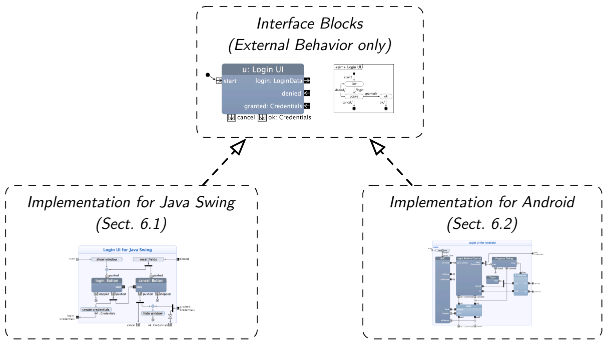


Figure 7.11: Realizations of *Login UI* for Swing and Android

provided by the partition *poiServer* in Fig. 7.8. The question handler service *c0* is responsible for providing the questions and evaluating the answers. Once a question is provided, it is sent to the answering session service. As indicated by the shadow, this collaboration is executed separately for each participating user and the group therefore handles multiple instances of it. The question is provided to *all* users, declared by the operator **select all**. The question is then forwarded within the answering session. Suggestions by the users are picked up via pin *mySugg*, internally forwarded to pin *suggOut* within the group partition. From there, they are distributed to all *other* users, declared by operator **select all/self**. These selection operators work as symbolic address filters and are further explained in [KBH07].

7.6 User Interface Blocks

Just as the behavior of the distributed services, the behavior of user interfaces is triggered by distinct events. In addition to the triggers from signal

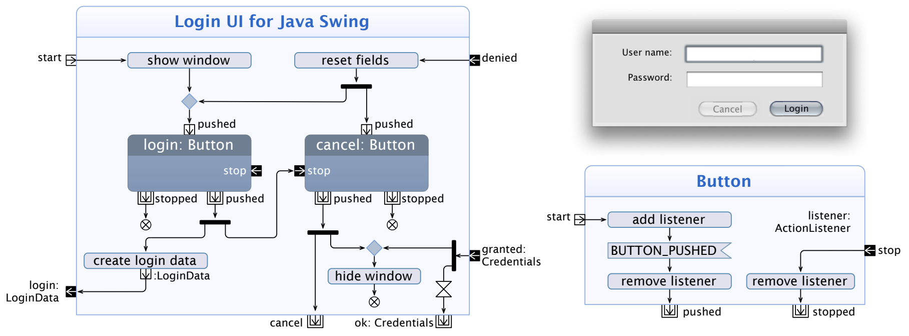


Figure 7.12: Activities for *Login UI* and *Button*

receptions and timeouts, also direct actions from users (like the tap on a button) must be taken into account. In contrast to the distributed service behavior, however, the *internal* behavior for user interfaces is highly dependent on the specific devices they are executed on: Not only do different platforms (like Java Swing or Android) offer different user interface elements and layouts, they also assume different life cycles of these elements which has influence on their overall behavior. Interestingly, we observe that the *external* events often are the same so that these differences may be encapsulated. User interface blocks may therefore have the same pins and the same ESMs while having very different internal realizations. For this reason, user interfaces may in the first place be described by an abstract block that only defines their external behavior such as the one from Fig. 7.7. Other building blocks may implement this external behavior, expressed in UML as the *realization* dependencies depicted in Fig. 7.11. This means that an application can be ported to different devices by exchanging only the internals of the specific building blocks, while the overall composition and especially the coupling to the service logic can stay unchanged. Our transformation tool therefore selects implementations of the Login UI block depending on the desired target platform. (The actual selection mechanism is part of the deployment and not detailed here.) Since the ESMs are identical, the overall applications will behave equivalent.

In the following, we will start in Sect. 7.6.1 by presenting one such implementation of *Login UI* with a simple building block for Java Swing, which introduces the mechanisms for the coupling to user interface elements. In Sect. 7.6.2 we extend our method and design another implementation of *Login UI*, specific for Android, which makes use of the layout editor of the Android SDK and takes care of the particularities of mobile devices.

7.6.1 Simple User Interface Block for Java Swing

Figure 7.12 shows the Login UI building block specific to Java Swing. It contains two buttons, one for sending the login data and one for canceling, each represented by a corresponding building block. The internals of the buttons are presented to the right. Once a button is started, it registers a listener to the graphical element. When this listener is activated, it sends an internal signal *BUTTON_PUSHED* to the underlying runtime scheduler. The behavior triggered by this signal follows after the accept signal action for *BUTTON_PUSHED*: The listener is removed and the button block terminates via *pushed*. To deactivate a button, a token may be sent via *stop*, upon which the block is terminated. The text field is created together with the window and the other elements, but since it does not trigger any events, no further elements are necessary for it in the model.

The external behavior of *Login UI* is defined by the ESM given in Fig. 7.7. After its start, it shows the window illustrated at the right hand side of Fig. 7.12, with the login button activated. When the user presses login, a token leaves block *login* via *pushed*, whereupon the login data is created from the user name and password provided and sent out. From then on, the block awaits the arrival of either *denied* or *granted*. In the denied case, users may select to retry or to cancel using the then activated cancel button. In the granted case, the *Login UI* block terminates via *ok*.[2]

A similar approach works for all the other user interface blocks as well, for instance the *Quiz UI*. Its external behavior is shown in Fig. 7.13. Once it is started, it goes to the *idle* state and changes into *active* once it receives a question. In this state, users may suggest the answer to the question while they receive suggestions from the group members. Once the final answer is submitted, the quiz UI enters into the *submitted* state and waits for the result. The internals of the block are similar to the ones from the *Login UI* in Fig. 7.12. Operations are used to update the state of the UI elements and register listeners to them, which send events back to the building block.

7.6.2 User Interface Blocks for Android

Figure 7.14 shows the login user interface on Android. Screen layouts as the one illustrated are called *activities*. (To distinguish them from UML activities, we always refer to them as *Android activities* in the following.)

[2]The ESM from Fig. 7.7 demands that parameters *granted* and *ok* are part of separate execution steps, for instance to leave time for releasing resources. Since the operation to hide a window in Fig. 7.12 is executed within one step, an intermediate delay element is added.

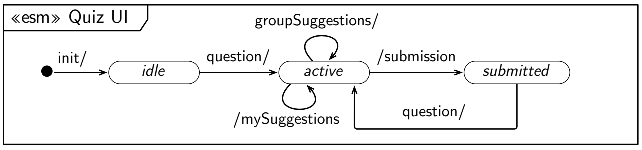


Figure 7.13: ESM of the QuizUI building block

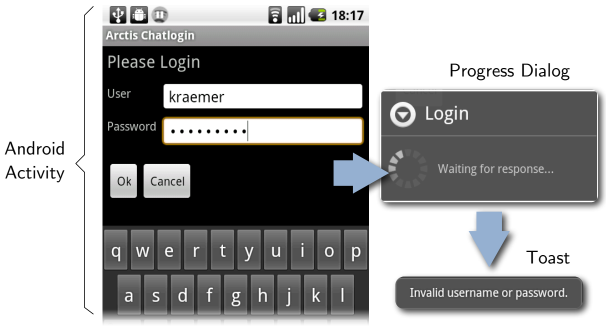


Figure 7.14: Screenshot of the login user interface on Android

In addition, a progress dialog is shown during the login, and the result is displayed with an Android-specific user notification, called *toast*. While the different user interface elements may look similar to those for Java Swing, there are some important differences:

- Only a single Android activity can be displayed on the screen at once. This means in particular that displaying several windows as on desktops is not possible.

- Since applications may be interrupted by other tasks, for instance an incoming call, any Android activity must be prepared to be moved into the background. In case the memory gets low, Android activities also have to be prepared to store their state and wait for a later reactivation.

- The appearance of user interfaces is determined by rather general layout files, which are created by graphical tools like the ones provided by the Android SDK. The instantiation of the user interfaces in terms of an object structure is done by the operating system based on this file, taking into account the current device configuration, regarding screen size and orientation, for instance.

For a modeling approach, this imposes several challenges, since the lifecycle of Android activities has to be taken into account, and the layout files somehow have to be connected to the building blocks in UML. To meet these additional challenges, we use the method depicted in Fig. 7.15, which first produces a building block to encapsulate the Android activity and then composes this block further.

1. The layout file is created with a graphical editor. In the example, this file arranges the text fields and buttons shown in Fig. 7.14.
2. Events are identified that originate at the user interface and that trigger service logic (such as the activation of the button *OK*). Vice-versa, events are identified that update the user interface, such as the arrival of a login denial.
3. With an ESM, constrains on the sequence of the events identified above are described. In the example, the ESM is similar to the one in Fig. 7.7.

The result of these steps is the external shell of a *layout control block* that encapsulates the user interface elements of the Android activity, in our example called *Login Activity Control.* Since all steps are focused on the visible part of user interfaces and in which sequence users should interact with them, they can be accomplished by a user interface designer without specific programming skills. In the following, a programmer adds the internals to this block and composes it with other blocks to form the final UI block:

4. The internals are added to the layout control block. These are methods and listeners that interact with the elements defined in the layout file, similar to how the listeners and operations work in the block of Fig. 7.12.
5. The contents of the methods is edited to update the user interface elements upon events (for incoming events) and listeners are registered to catch events originating at the user interface.
6. In a final step, the layout control block is combined with other user interface elements such as dialogs from a UI library to create a comprehensive Android UI block. A special block taking care of the lifecycle of an Android activity is added as well.

As a result of this method, we obtain the Login UI block for Android shown in Fig. 7.16. From an external view, it behaves as described by *Login*

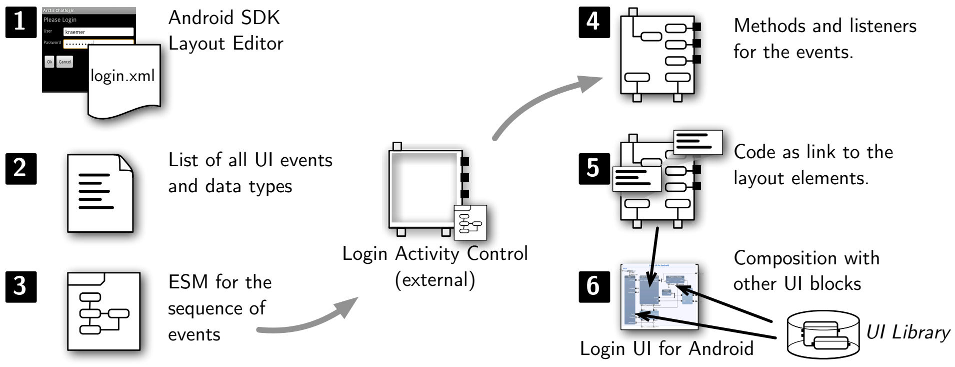

1
login.xml
Android SDK Layout Editor
2
List of all UI events and data types
3
ESM for the sequence of events
Login Activity Control (external)
4
Methods and listeners for the events.
5
Code as link to the layout elements.
6
Composition with other UI blocks
Login UI for Android
UI Library

Figure 7.15: Method to create Android UI Blocks

UI from Fig. 7.7. Internally, it is composed from several blocks and adheres to Android's particularities:

- Block *ALC* (for *Android Lifecycle Controller*) creates the Android activity for the login screen. Since the creation must be done by the operating system, only the class is passed to it. The instance is returned via the pin *onCreate*. Furthermore, *ALC* monitors the lifecycle of the activity. If another application is coming to the foreground, the pins *onPause* and *onResume* trigger an event upon which data can be persisted. (Not shown here.) Eventually, the Android activity can be terminated via *finish*, or is destroyed by the operating system via *onDestroy*.

- Block *Login Activity Control* makes the user interface elements of the Android activity available, as described above. Input pins update the text fields and output pins forward events that originate from the buttons.

- Block *Progress Dialog* displays a dialog while the login credentials are evaluated by the server. The user may also cancel the login process using this dialog, which is expressed by the corresponding pin.

- Block *Toast* displays a message in case the login was denied.

The blocks *Gate* and *Crossover* are taken from our standard library and are used to intercept two alternative flows that arrive to them via *in1* and *in2*. *Crossover* terminates the progress dialog upon the arrival of either *denied* or *ok* via *flow*, and *Gate* forwards *ok* or *cancel* to the termination

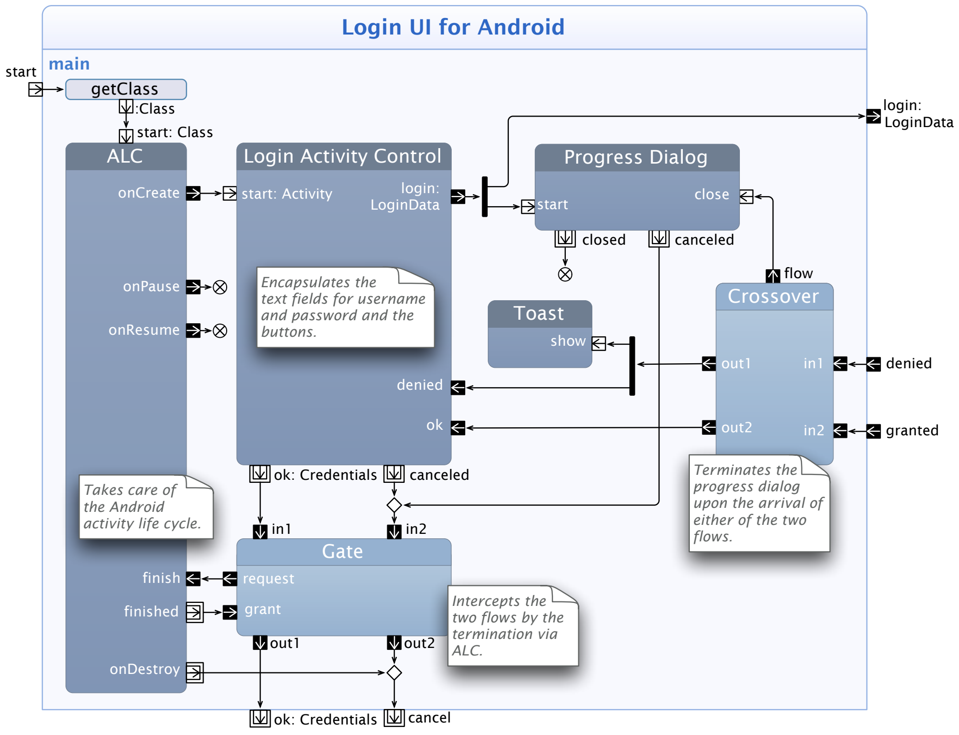


Figure 7.16: Android version for the login block

but intercepts this forwarding by a termination of the Android activity via block *ALC*.

7.7 Validation and Automatic Verification

Due to the formal semantics [KH10], the specifications expressed by the UML activities can be analyzed by the model checking tools described in [KSH09], [KBH09]. The encapsulation of building blocks in their ESMs leads to a compositional verification style, in which each building block can be analyzed separately. When a specification is composed of several blocks, for instance several services and UI blocks as in Fig. 7.8, then all its subordinate blocks are abstracted by their ESMs. This keeps the state space of the analysis rather small, as shown in [KSH09]. In addition, it is possible to study design solutions that are not completely finished yet. For instance, the block for the login on Android in Fig. 7.16 can be validated even if the internals of *Login UI Control* are not yet finished. This can provide feedback at earlier stages of the development, which reduces costs for changes.

As a means of validation, i.e., whether a behavior is suitable to solve a certain problem, the behavior of a building block may be simulated by means of a graphical animation, as shown in [KBH09]. In such a simulation, designers can step through possible behaviors by looking at the sequences of actions and states. The actions are triggered by events and the states are defined by the ESM states of the inner blocks and other elements of the UML activities.

In addition to the validation that relies on the judgement of the designer examining selected paths through the state space implied by a specification, a thorough and automatic verification is possible that takes the entire state space into account. Such an analysis reveals errors in the interactions within collaborations, for example unbounded queues, deadlocks, race conditions and inconsistent terminations, i.e., situations that are generally undesired and that are most likely design flaws. (This means the given specification is verified against a set of desirable properties.) With respect to the user interface blocks proposed here, two properties deserve special attention:

- A building block must obey its own ESM description. For instance, the *Login UI* block may push a token through *ok* if (and only if) it received a *granted*, since its ESM allows these events only in that order.
- A composition of building blocks must obey the ESMs of all the blocks. For instance, the login gui block may only be composed in such a way that, once it has emitted a *login*, it will eventually receive either a *denied* or *granted* (but not both of them). This is very useful for user interfaces that maintain a certain state (for example if a UI element is enabled or not) and relieves the developer of ensuring these conditions manually.

Once errors are detected, the model is annotated and may be animated to help the designer to understand the error situation and correct it, as shown in [KBH09].

7.8 Automatic Implementation

To implement the specifications given in terms of UML activities and the complementing Java methods for the call operation actions, we developed a two-step process. In a first step, the activities are transformed into UML state machines. Each partition of the system activity in Fig. 7.6 denotes a separate component, for which we generate a UML class. The transformation considers which collaboration roles are bound to the respective components and generates state machines for the respective activity partitions. This transformation is detailed in [Kra08].

In a second step, code is generated from the UML state machines. The state machine logic is translated to special transition methods that execute the state machine transition actions, such as sending signals to other machines, setting timers or executing the Java operations that have been copied directly from the building blocks. Our code generators produce code for different platforms, such as standard Java [Bje09], embedded Sun SPOT devices [KSH09a], as well as Android [Hau09].

7.9 Related Work

In general, we observe that some service engineering approaches ignore the details of user interfaces completely and leave such aspects to the implementations. Other approaches have user interfaces as their primary focus, but treat the service logic as secondary.

The possibilities of model-driven user-interface development have been explored for instance in [AJI05, LSHA08, SP00]. Most of these approaches focus on web applications. In [AJI05], user interface behavior is modeled with UML use case diagrams which are detailed with activity diagrams showing the interactions between users and the system. User interface components such as Java applets are then generated. Link et. al [LSHA08] use extended UML activity diagrams to capture UI aspects both from the user and the system perspective. UI models are used to specify the assembly of user interfaces components, from which code can be generated. In [SP00], user interfaces are modeled using several types of UML diagrams. Class diagrams are used for the domain model representing domain entities and their relationships, while activity diagrams are used for task modeling. In contrast to these approaches, we do not try to automate UI development as such, but provide a way to factor out and encapsulate UI elements from services and use them in application composition at the modeling level.

UWE [BE08, KBHM00], WebML [CFB00], OO-H [GCP01], and MIDAS [CMV04] are model-driven approaches for the domain of web-based systems. In [BE08], user interfaces are modeled and implemented by code generation. This approach considers only web services which are mapped to Java methods. UI components just invoke a web service and wait for the result in order to present it. In UWE, information aspects are specified by content models using UML class diagrams. Nodes and links of the hypertext structure are specified in a navigation model using UML class diagrams. Composition of the presentation elements are specified in a presentation model using stereotyped UML class and interaction diagrams. Behavior is specified by the process flow model using UML activity diagrams. Similar to UWE, WebML also has the concepts of structural model for data modeling, composition model for the page contents, and personalization model for the customizing features. All the concepts of WebML are associated with a graphic notation and WebML specifications can also be translated into web pages. In OO-H, structural domain information is captured using UML class diagrams. From there, different navigation models are created for each user type. Then, using different mapping steps, a default web-interface is generated. Presentation models based on templates are combined with the aid of a set of patterns to improve the quality of the generated interfaces. A model compiler is used to generate the user-interfaces for internet applications. The MIDAS approach has a system core that defines domain and business models. Over this central core, it defines structural and behavioral dimensions of the web-application using conceptual and platform-specific models for content, hypertext and presentation. A common and interesting feature of these methods is the modeling of the application in different orthogonal levels and aspects such as content, hypertext, composition and presentation modeling. One limitation of these approaches is that they consider only web applications and client-server type of services. In contrast, we provide an approach for reactive services in general without technology bindings. Our service models are collaborative and encapsulate the details of interactions and distribution. We also treat UI elements in the same way as service behavior.

SoaML [Gro08] is a UML profile for the structural aspects of the service. Services contracts are modeled in a way similar to your approach of specifying collaborative services. UML4SOA [KMH$^+$07] is a profile for specifying behavioral aspects of services. It contains specialized elements for modeling service interactions, compensation, event and exception handling. Neither of these UML profiles, however, deal with details of user-interfaces.

Since UML in general does not provide any dedicated concepts or diagrams for developing user interfaces, some approaches use profiles specifically for user interface design. For instance, [Lie04] presents how UML activity diagrams can be used to show the flow of windows and other UI elements, by representing actions that a user can invoke while working with the user interface. The approach introduces stereotypes to further categorize these actions. In comparison to our work the focus lies on the navigation between local windows, but no connection to service logic or its coupling with events. Van den Berg and Coninx [VdBC05] propose a UML profile for the description of user interfaces in relation to context models that represent the situations and environment in which the interface is used. Similar to our approach is the use of UML activity diagrams, but the emphasis on what they represent differs. Our models are focused on the technicalities of user interfaces and are very detailed with respect to the event-driven behavior, to an extend that allows code generation from these models. In contrast, their models pay more attention to the relations between the system, the user and the environment by representing the latter explicitly in the diagrams.

App Inventor [Weba] is a visual programming environment for creating mobile applications by connecting visual boxes like puzzle pieces. Instead of writing code, the programmer specifies application logic using boxes for specific functionalities along with control constructs that realize programming structures like conditions and loops. Its development environment is similar to StarLogo TNG [WMWK06], and Lego Mindstorms [Webb] but targeting mobile applications for Android. The diagrams of App Inventor resemble Nassi-Shneiderman diagrams known from structured programming, while the activity diagrams underlying our approach model general data flows, which also offer synchronizing elements like join nodes. It is therefore not clear to us how more general reactive behaviors, in which reactions on events depend on complex states, can be expressed, especially if more than one event need to be synchronized. In our approach, the internal details of the services and UI blocks are encapsulated with their external behavior specified using ESM. This allows replacement of internal details while keeping the external behavior. Moreover, our building blocks are composed together by using different types of pins such as initiating, terminating and streaming pins linked together by arbitrary synchronizing logic. We also separate user interface and service concerns but compose in a unified way such that their composition can be verified and validated (automatically) early at the design time. As in our approach, the layouts for graphical user interfaces need to be designed by hand in App Inventor, and their association with corresponding functionality blocks is done manually.

7.10 Concluding Remarks

We proposed a method to integrate collaborative service behavior with the local control of user interfaces using building blocks based on UML activities. We observe that the demonstrated specification style leads to a system decomposition in which user interface elements are represented by separate, self-contained building blocks that may be developed by UI experts. These may be combined with collaborative service blocks to form the complete system specification, which can then be analyzed and implemented in automated processes. Since the proposed specification style is an extension of our existing method for reactive system engineering, it inherits several properties that we consider as beneficial for the development of services in general:

- The decomposition into collaborative building blocks based on activities models systems in reasonably compact but readable form. Our case study involves multiple users and mobile devices communicating with each other using a variety of services each involving several system participants. The complexity of the system therefore goes beyond that of simple toy examples. However, we are still able to present its overall specifications on a few pages, as shown by Fig. 7.6 and 7.8.
- Interactions and coordination of collaborative behavior is handled explicitly on the service model level, with activities that provide an overview of the cross-cutting behavior executed by several components.
- The automated and incremental verification encourages developers to formally analyze their specifications often and from the beginning, block by block.
- The automated implementation makes the service specifications the canonical description from which everything else is derived. This avoids inconsistencies.

With respect to the integration of user interface behavior, our approach has several important properties:

- **Encapsulation of UI details.** The building blocks encapsulate detailed operations on user interfaces, which would otherwise obstruct the overall specification and which would make it difficult to understand the cross-cutting services.
- **Separation of expertise**. Due to the separation of concerns enabled by the decomposition into building blocks, developers with different fields of expertise may work largely independently from each other.
- **Protection of operation call sequences.** The above mentioned separation could also be achieved by object-oriented techniques with classes separated by interfaces. But by using building blocks, we also

ensure that operations are called in the right order, secured by the ESMs. Furthermore, since building blocks are at the service model level, the analysis of the overall behavior is easier than at the code level.

- **Application-Oriented Composition.** The coupling by means of activity flows is application-oriented: Data types and events correspond to concepts of the domain, like the types *Position* and *User* in Fig. 7.8. This makes specifications easier to understand.
- **Interchangeability of UI Frameworks.** Since operations and resources belonging to a certain UI are encapsulated as building blocks, they can be exchanged easily so that a system may use different UI frameworks. If the new building blocks adhere to the same ESMs, they will behave consistently.

So far, we encapsulate user interfaces into building blocks in a manual, although highly structured method as outlined in Fig. 7.15. Since this invokes recurring patterns, we see potential for the automation of this process and think of solutions that encapsulates artifacts produced by GUI builders for different UI frameworks automatically. Especially the creation of building blocks controlling the Android activities can be automated further, taking the layout files created by the Android SDK as input.

References

[AJI05] Jesus M. Almendros-Jimenez and Luis Iribarne. Designing gui components for uml use cases. In *Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, pages 210–217, Washington, DC, USA, 2005. IEEE Computer Society.

[BE08] Peter Braun and Ronny Eckhaus. Experiences on model-driven software development for mobile applications. In *Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 490–493, Washington, DC, USA, 2008. IEEE Computer Society.

[Bje09] Marius Bjerke. Runtime support for executable components with sessions. Master's thesis, Norwegian University of Science and Technology, 2009.

[CFB00] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web modeling language (WebML): a modeling language for designing web sites. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 33:137–157, June 2000.

[CMV04] P. Caceres, E. Marcos, and B. Vela. A mda-based approach for web information system. In *Proceedings of Workshop in Software Model Engineering. Retrieved from: http://www.metamodel.com/wisme-2003*, 2004.

[GCP01] Jaime Gómez, Cristina Cachero, and Oscar Pastor. Conceptual modeling of device-independent web applications. *IEEE MultiMedia*, 8:26–39, April 2001.

[Gro08] Object Management Group. Service oriented architecture modeling language (soaml) - specification for the uml profile and metamodel for services., November 2008.

[Hau09] Stephan. Haugsrud. Developing android applications with arctis. Master's thesis, Norwegian University of Science and Technology., 2009.

[KB09] Surya Bahadur Kathayat and Rolv Bræk. Platform support for situated collaborative learning. In *International Conference on Mobile, Hybrid, and On-line Learning, 2009. ELML'09.* IEEE Press, 2009.

[KBH07] Frank Alexander Kraemer, Rolv Bræk, and Peter Herrmann. Synthesizing components with sessions from collaboration-oriented service specifications. In *Proceedings of the 13th international SDL Forum conference on Design for dependable systems*, SDL'07, pages 166–185. Springer-Verlag, 2007.

[KBH09] Frank Alexander Kraemer, Rolv Bræk, and Peter Herrmann. Compositional service engineering with arctis. *Telektronikk*, 105(2009.1), 2009.

[KBHM00] Nora Koch, Hubert Baumeister, Rolf Hennicker, and Luis M. Extending uml for modeling navigation and presentation in web applications. In *Proc. of the Workshop Modeling Web Applications in the UML*, 2000.

[KH06] Frank Alexander Kraemer and Peter Herrmann. Service specification by composition of collaborations–an example. In *Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology*, WI-IATW '06, pages 129–133, Washington, DC, USA, 2006. IEEE Computer Society.

[KH09] Frank Alexander Kraemer and Peter Herrmann. Automated encapsulation of uml activities for incremental development and verification. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems*, MODELS '09, pages 571–585. Springer-Verlag, 2009.

[KH10] Frank Kraemer and Peter Herrmann. Reactive semantics for distributed uml activities. In John Hatcliff and Elena Zucca, editors, *Formal Techniques for Distributed Systems*, volume 6117 of *Lecture Notes in Computer Science*, pages 17–31. Springer Berlin / Heidelberg, 2010.

[KMH$^+$07] Nora Koch, Philip Mayer, Reiko Heckel, L. Gönczy, and C. Montangero. UML for service-oriented systems. Technical report, 2007. Technical report, Deliverable 4.14a of the SENSORIA Project, 2007.

[Kra08] Frank Alexander Kraemer. *Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks.* PhD thesis, Norwegian University of Science and Technology, 2008.

[KSH09a] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann. Model-driven construction of embedded applications based on reusable building blocks: an example. In *Proceedings of the 14th international SDL conference on Design for motes and mobiles*, SDL'09, pages 1–18, Berlin, Heidelberg, 2009. Springer-Verlag.

[KSH09b] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann. Tool support for the rapid composition, analysis and implementation of reactive services. *Journal of Systems and Software*, 82(12):2068–2080, 2009.

[Lie04] Benjamin Lieberman. Uml activity diagrams: Detailing user interface navigation, November 2004. http://www.ibm.com/developerworks/rational/library/4697.html, Accessed on December 2010.

[LSHA08] Stefan Link, Thomas Schuster, Philip Hoyer, and Sebastian Abeck. Focusing graphical user interfaces in model-driven software development. In *Proceedings of the First International Conference on Advances in Computer-Human Interaction*, ACHI '08, pages 3–8, Washington, DC, USA, 2008. IEEE Computer Society.

[SP00] Paulo Pinheiro Da Silva and Norman W. Paton. User interface modelling with uml. In *In Proceedings of the 10th European-Japanese Conference on Information Modelling and Knowledge Representation*, pages 203–217. IOS Press, 2000.

[VdBC05] Jan Van den Bergh and Karin Coninx. Towards modeling context-sensitive interactive applications: the context-sensitive user interface profile (cup). In *Proceedings of the 2005 ACM symposium on Software visualization*, SoftVis '05, pages 87–94, New York, NY, USA, 2005. ACM.

[Weba] Google App Inventor Website. http://appinventor.googlelabs.com/. Accessed on December 2010.

[Webb] Lego Mindstroms Website. http://mindstorms.lego.com/. Accessed on December 2010.

[WMWK06] Kevin Wang, Corey McCaffrey, Daniel Wendel, and Eric Klopfer. 3d game design with programming blocks in starlogo tng. In *Proceedings of the 7th international conference on Learning sciences*, ICLS '06, pages 1008–1009. International Society of the Learning Sciences, 2006.

Chapter 8

Paper 3

Title: From Flow-Global Choreography to Component Types.

In: 6th Workshop on System Analysis and Modeling (SAM 2010), 4th-5th October 2010. Oslo, Norway

Publisher: Springer (LNCS 6598). [1]

[1]Selected and revised papers from the workshop are included

From Flow-Global Choreography to Component Types

Surya Bahadur Kathayat, and Rolv Bræk
Norwegian University of Science and Technology (NTNU)
Department of Telematics
Trondheim, Norway
Email:{surya, rolv.braek}@item.ntnu.no

Abstract

The need for global behavior definitions is well established both in the domain of embedded reactive systems and service-oriented (business) systems. The problem has been to define the global behavior with sufficient, rigor and completeness to fully cover the intended behavior and not just some scenarios, and to enable automatic synthesis of component behaviors in practical systems and service development. In this paper we build on previous work where UML collaborations are used to structure systems and services into reusable building blocks, and UML activities to model global behavior, called choreography in the following. We identify two forms of choreography: one where all flows are localized to the roles participating in collaborations and another where the flows are not localized and thus more abstract. We propose a novel approach to map the flow-global choreography to a flow-localized choreography and further to distributed component behaviors (orchestrations) with well defined interfaces from which implementation code can be generated using existing techniques. The overall approach combines the merits of global choreographies and collaborative building blocks with the flexibility of component oriented designs. The approach is illustrated using a city guiding system as case study.

keywords: Service choreography, component design, model-driven development.

8.1 Introduction

In reactive systems as well as in business systems there is a need to define global collaborative behavior as well as local component behavior. We use the term *choreography* to denote global collaborative behavior involving two or more participants, in contrast to *orchestration* that denotes the local behavior of each participant. Choreography is needed to define and analyze the overall service behavior, whereas the orchestration is needed to completely define component behaviors for implementation. These terms come from the domain of business services and SOA and are in accordance with common use in that domain, see e.g. [Erl07].

Ideally the choreography should be sufficiently precise and complete that component behaviors can be automatically synthesized. This would enable the service engineer to work mainly on the level of choreography, focusing on the intended global behavior, and not the detailed component behavior. For this to be possible one needs suitable building blocks and composition mechanisms for defining the intended global behavior completely and precisely, and one needs a systematic way to go from global behavior to distributed local behaviors such that all the coordination problems of distributed realizations are properly handled. UML collaborations provides a structural framework to do this. They define a structure of participating roles where collaborations taking place among the roles may be represented as collaboration uses referring to separately defined collaborations with associated behavior. We distinguish between *elementary* collaborations that are not further decomposed into collaboration uses, and *composite* collaborations. In this paper we use UML activity diagrams for the choreography of both elementary collaborations and composite collaborations as well as for orchestration. Motivation for this choice and comparison with alternatives such as sequence diagrams and interaction overview diagrams is given in Sect. 8.6. Notation and semantics are in accordance with the UML definition except for a small notational addition that will be explained. The overall approach we address in this paper is illustrated in Fig. 8.1.

- *Flow-global choreography* is used to define the intended global behavior of composite collaborations on a high level of abstraction avoiding details of localization and resolution of coordination problems that may occur at the level of orchestration. The behavior is defined by an activity diagram connecting actions by flows that are not localized to any particular role. Actions may either represent the behavior of a collaboration or a local activity. This is the right level for discussing the intended behavior with end-users and other stake holders. It is

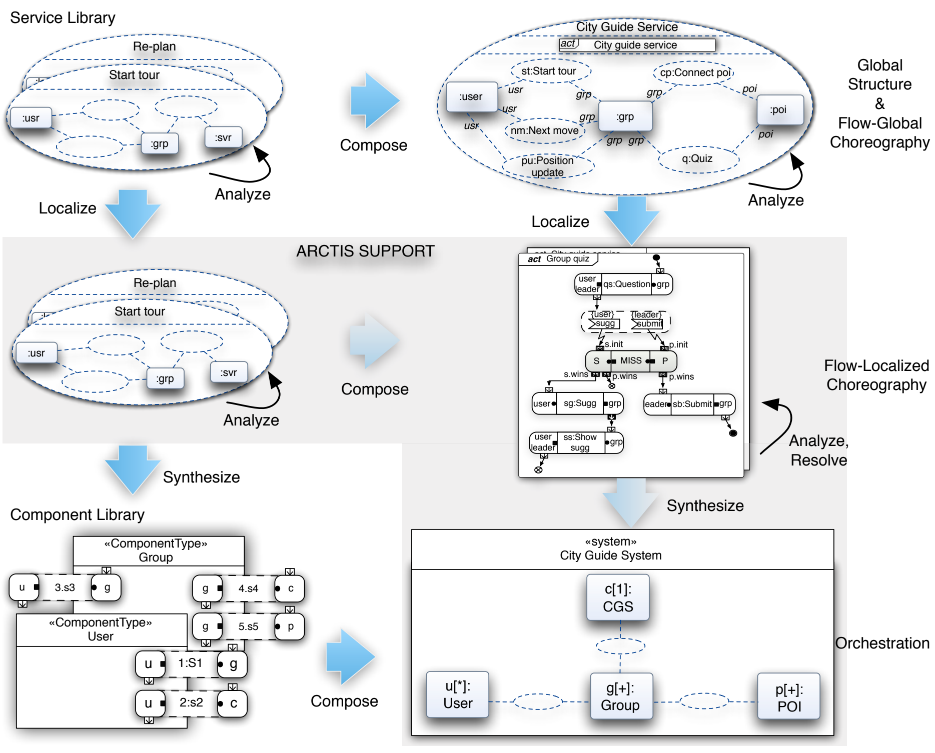


Figure 8.1: Overview of the proposed approach

also a useful first step in the formalization of requirements.

- *Flow-localized choreography* is used to define global behavior in sufficient detail to allow extensive analysis and to automatically synthesize the behavior of component types providing orchestration. The behavior is defined by an activity diagram connecting actions, that either represent the behavior of a collaboration or a local activity, by flows that are localized to the roles.

- *Component types* define the local activity flow, the orchestration, of each component. They are derived using a projection of the flow-localized choreography to include only local actions performed by the component. Information about the collaborations it participates in and the ordering imposed by external activity flows are retained in order to support automatic realization as well as compatibility checks during subsequent composition of components into working systems.

Tool support for part of this approach is provided by the Arctis toolset [KSH09], as indicated by the grey area in Figure 1. It enables a compositional development of reactive systems where building blocks representing collaborative behavior are defined separately using a "swim-lane" like notation for UML activity diagrams, put into libraries, and then composed in different ways. The Arctis tool provides extensive support on the flow-localized choreography level. It supports model checking and can synthesize complete system behavior in the form of communicating state machines that are automatically realized using code generation techniques.

In this paper we elaborate on the parts not supported by Arctis, i.e. the flow-global choreography, the mapping to a flow-localized choreography and the derivation of component types that may be used in subsequent system composition. When deriving component types it is necessary to detect and resolve the *realizability problems* that may occur. When a choreography is mapped to a composition of component behaviors and the global behavior emerging from the composition differs from the global behavior specified in the choreography we say there is a realizability problem. It may for instance happen that messages are received in a different order than specified or that additional messages are needed to ensure the specified ordering. Realizability problems have been extensively studied in the context of interaction diagrams where they sometimes are referred to as *implied scenarios* or *races*. Similar realization problems may also occur when activity diagrams are used, and therefore are discussed in this paper.

In the domain of embedded, reactive systems components will often be linked to physical objects like users with autonomous behavior that may take independent initiatives needing to be coordinated and resolved. In business systems it is more common that services are provided by passive components that only respond to requests, and never take autonomous initiatives. This is however, gradually changing with services becoming more pro-active and "pushing". Dealing with mixed initiatives is therefore one of the fundamental coordination problems to be addressed in distributed reactive service systems. We therefore consider the case where components in general may be active and take independent initiatives either triggered by user actions, time or other events. In our approach we seek to identify all such problems and to resolve them either in the choreography or the orchestration.

The rest of the paper is structured as follows: A case study is introduced in Sect. 8.2 and used throughout the paper to illustrate our approach. Section 8.3,8.4, and 8.5 describes our proposed approach for the choreography representation, flow-localization and component design. Related work are

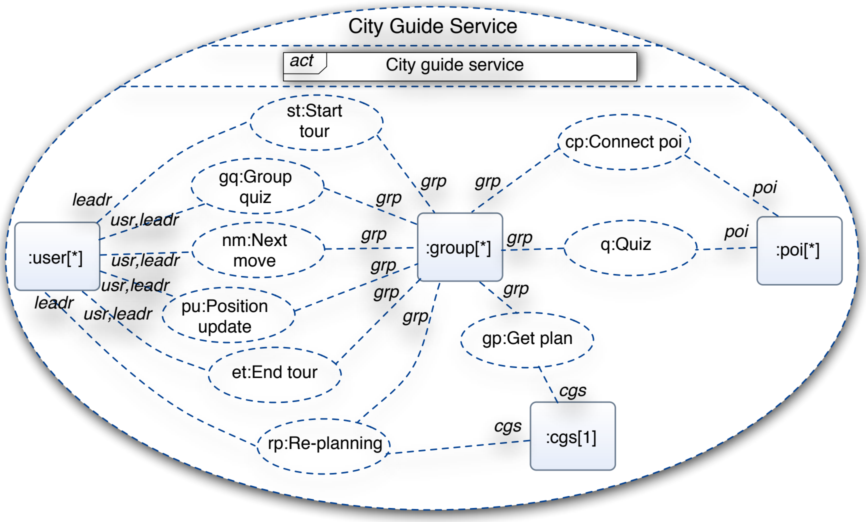


Figure 8.2: Roles and sub-collaborations in a City guide service

discussed in Sect. 8.6. Finally, concluding remark are given in section 8.7.

8.2 Service Structure

We illustrate our approach using a city guide service example, which has been implemented in our previous work [KB09, KKB11]. It is a complex and challenging example, and therefore provides a realistic basis for evaluation. The *users* are students that have just arrived in a city and want to learn about different places of interest. They use GPS enabled hand held devices and may create profiles specifying their preferences and interests. Based on their locations and profile information, the users participate in different groups and social activities. For instance, when a user is trying to find a particular point of interest (e.g. a historic building), the user may interact with other users having similar interests. Users may have a peer-to-peer chat, group chat and discussions. When users are in the vicinity of a point-of-interest (POI), they may participate in a group quiz provided by a particular POI.

Fig. 8.2 shows the city guide service as a UML collaboration with four different roles that will map to corresponding component types: the *user* role that represents a user of the system; the *group* role that represent a group of users and support group functionalities; the *poi* role that represents a point-of-interest and provide quizzes and finally, the *cgs* role (city guide manager) that provides a guided tour to the users. A number of collaborations uses represent sub-services that are defined separately:

- *Start tour* performs the tour initiation triggered by the lead user. It has two roles: *leadr* that is bound to *user* and *grp* bound to *group.*
- *Get plan* fetches a plan which is a list of points of interests.
- *Re-planning* is a composite activity that stores the plan and allows the lead user to re-plan for the group (details are not discussed in this paper).
- *Position update* is used to continuously update the current position as users are moving.
- *Connect poi* will inform the POI to initiate a *Quiz* session with the group.
- *Group quiz* performs a quiz session where each group member may suggest answers and the lead user may submit an answer.
- *Tour manager* manages the city guide tour. It holds a plan in the form of a list of POIs. It provides the next POI to be located when a current POI is finished.
- *Proximity detector* checks if the users (using their positions) in a group are within a defined proximity of a POI and initiate *Connect poi* when this is the case.

We assume that the behavior of each of these collaborations is defined by an activity diagram having the same name as the collaboration. For example, the activity diagram for the *Position update* collaboration is shown in Fig. 8.3(a) using flow-localized choreography. Such diagrams are easily derived from corresponding sequence diagrams, if one is starting from a set of sequence diagrams rather than activity diagrams. Each collaboration encapsulate interactions across an interface between two or more participating roles and is used as a collaborative building block in the following. They are referenced by the actions nodes in a choreography model (discussed in Sect. 8.3) according to the notation given in Fig. 8.3(b).

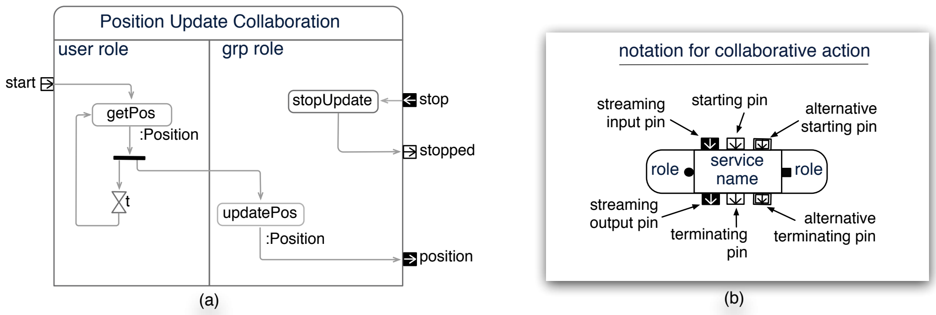


Figure 8.3: Elementary service model and Service notation used in choreography

8.3 Flow-Global Choreography Models

The flow-global choreography defines the desired global execution ordering of collaboration uses in a composite collaboration. For example the flow-global choreography of the City Guide service is shown in Fig. 8.4 using a UML activity diagrams.

All actions in the diagram call activities defined separately. The called activities are either local to a role called local activities (here the *prox. detector* and *tour manager*) or defining the choreography of a collaboration (the rest). For example, the start tour action *st.Start tour* and the position update action *pu.Position update* in Fig. 8.4 call the activity given for the *st:Start tour* and *pu:Position update* collaborations in Fig. 8.2 respectively. In this way the diagram defines the choreography of, possibly nested, collaboration behaviors. The *gq.Group quiz* call behavior action calls the group quiz activity represented in Fig. 8.5.

Note that the participating roles of collaborations are indicated by partitions defined by solid lines in the action symbols. The initiating and terminating roles of the collaborations are indicated using dots and squares, respectively. This (dot and square) notation is not part of UML, but may be provided by additional profiling. An action can have all the different types of pins that UML allows, such as initiating, streaming and terminating pins. An initiating pin will start the called activity if it is not started yet. Terminating pins will terminate the called activity. Streaming pins can pass tokens while the called activity is active, i.e. a flow between two actions connected by streaming pins allows the called activities to interact without being stopped. For example the streaming flow from *quiz* to *group*

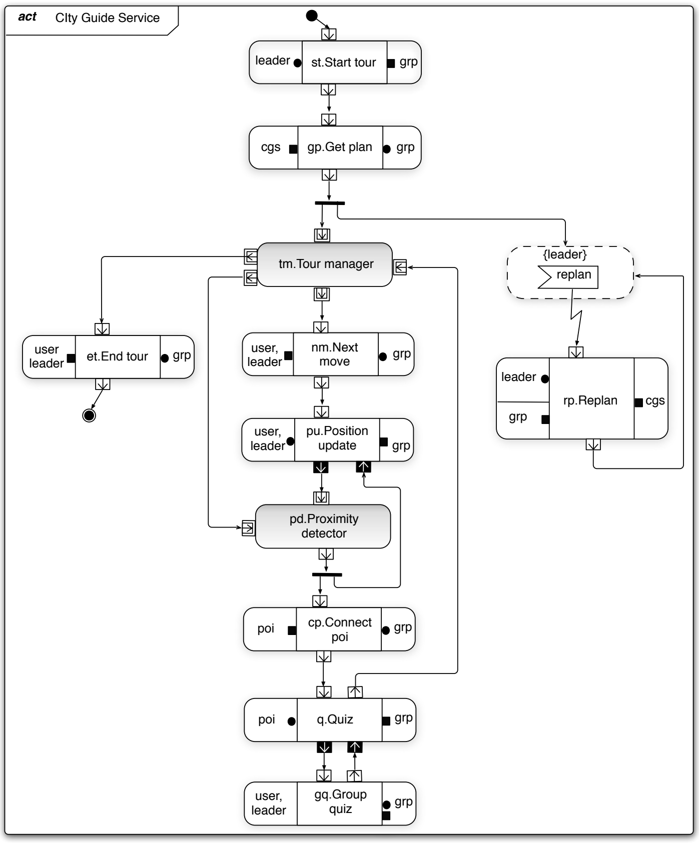


Figure 8.4: Flow-Global choreography model

quiz in Fig. 8.4 means that the *quiz* action, without being terminated, may interact with the *group quiz* i.e. they overlap. A question is sent from the quiz service via the streaming pin to the group quiz service (quiz is still active and group quiz is started). Using the group quiz service, the members of the group may discuss the possible answers and agree on one solution before a leader submits the answer.

Note that variables and data are omitted in Fig. 8.4. However, it is possible to define and use data in the normal way defined for UML activity diagrams.

The particular case of initiative choices are modeled using interruptible regions (denoted by dotted box) and interrupting flows as shown in Fig. 8.5. In a group quiz service, there may arise a situation where a user is proposing an answer at the same time as the leader is submitting an answer. What

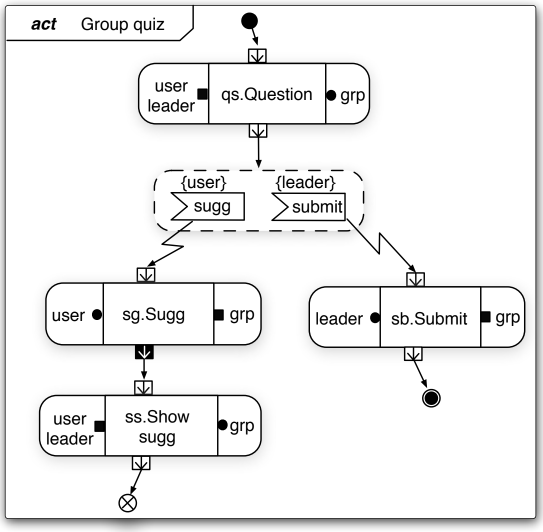


Figure 8.5: Flow-global specification of Group quiz service

we want is to choose either one or the other as specified in Fig. 8.5, but in a distributed realization they may occur nearly simultaneously and require additional coordination to resolve which initiative to select. More on this is in Sect. 8.4.5. Optional behavior triggered by external events may be modeled in a similar way. A leader may for example opt to re-plan while the other users are using a current plan, triggered by the *replan* event in Fig. 8.4.

We note that there is a certain amount of concurrency in the choreography in Fig. 8.4, that follows explicitly from the parallel fork and implicitly from the streaming pins. It is important to note here that this concurrency reflects the nature of the problem and is not imposed by the notation as such. In simpler cases, activities have only starting and terminating pins. Connecting such pins will result in a normal sequential ordering. We use streaming pins where we want activities to interact without being stopped.

8.4 Flow Localization

At the flow-global level we define the *intended* ordering including streaming pins, interruptible regions and initiative choices. At the flow-localized level we need to ensure the ordering in terms of local flows. In Sect. 8.4.1 and 8.4.2, we first consider direct flows with no control elements between the

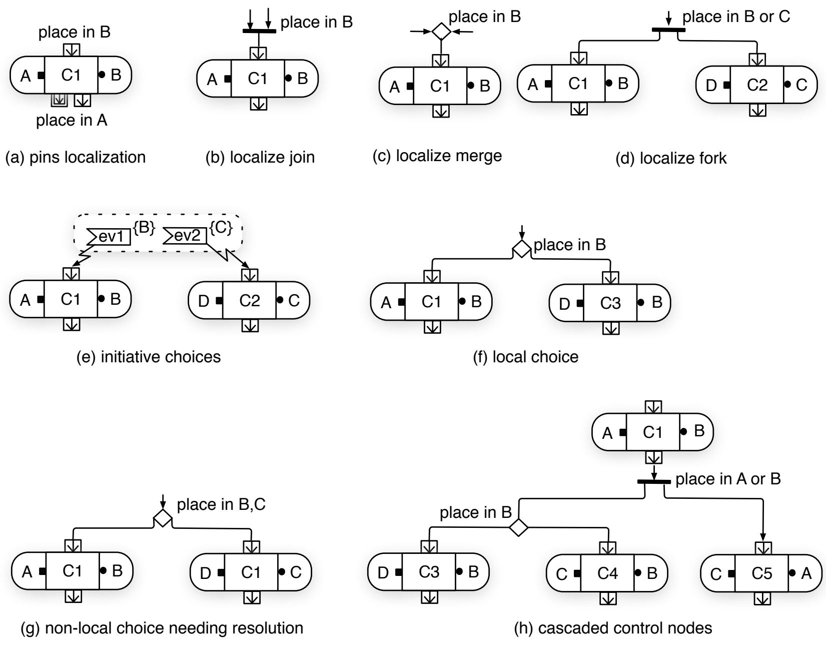


Figure 8.6: Summary of the rules for localizing pins and control nodes

action nodes. In Sect. 8.4.3, the localization of the control nodes and notion of path is introduced. Streaming, and interruptions are then discussed in the following sections.

8.4.1 Strong Sequence Localization

In this step we ensure the *strong sequencing* semantics of UML AD, i.e. that leaving an activity through an activity final node terminates the entire activity. It means that there will be no overlap or concurrency among activities that follow each other sequentially according to the flow-global choreography, and hence realization problems caused by overlaps are avoided.

We now assume that the collaborations on the lowest level of decomposition have only one initiating role and one terminating role, with a starting input pin assigned to the initiating role and a terminating output pin to the terminating role. In the case of parameter sets, all alternative pins need to be assigned to the same role. This is summarized in Fig. 8.6(a). The global flows are then localized in the following way:

- A global edge directly from collaboration action C1 to C2 is attached to the corresponding terminating output pin or parameter output pin

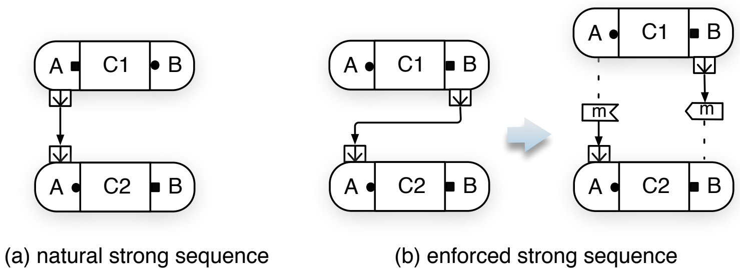


Figure 8.7: Strong sequence localization

of C1 and the starting pin or parameter input pin of C2.

- Every external event is localized to a single role, as illustrated in Fig. 8.6(e). For example the *replan* event within the interruptible region in Fig. 8.4 is assigned to a lead user.

Two cases are possible after this:

- *Natural strong sequence*: the flow between two collaborations C1 and C2 is completely localized to one role, as shown in Fig. 8.7(a). This means that strong sequence is ensured by the global flow directly. In the city guide choreography (Fig. 8.4) there is a natural strong sequence between most activities.

- *Enforced strong sequence*: the flow between two collaborations C1 and C2 is not completely localized to one role and therefore implies a coordination message being passed to enforce the strong sequence. This is shown in Fig. 8.7(b) where the terminating pin of service C1 is connected to the initiating pin of C2. Flows linking pins on different roles imply interaction using send and receiving events at the terminating and initiating roles respectively. Although this ensures a strong sequence, it adds a communication overhead.

8.4.2 Weak Sequence Localization

Flows with enforced strong sequence require additional communication that sometimes is neither desirable for performance reasons nor necessary for realizability reasons. In such cases we consider using weak sequencing in stead. *Weak sequencing* is the normal semantics of Sequence diagrams, and

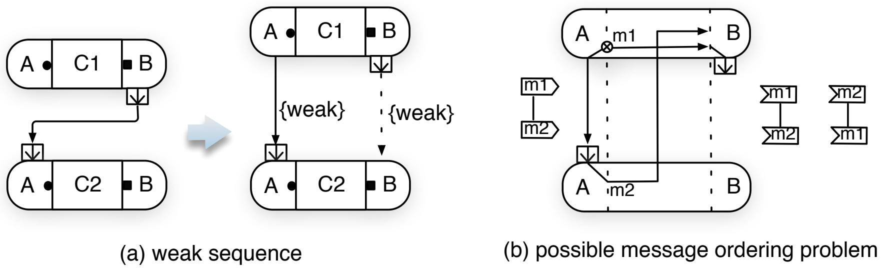


Figure 8.8: Weak sequence localization

distributed system in general. This means that a component may initiate C2 as soon as it is finished with its role in C1, even if messages may still be in transfer so that all roles of C1 are not completely finished. It means that there may be some overlaps between collaborations that potentially may cause realizability problems such as races. In order to model the weak sequencing we attach a local terminating pin to the role in C1 that initiates C2. This is modeled as a streaming pin with the property {weak}, as indicated in Fig. 8.7 and 8.8. So far we have considered flows that connect to the initiating role of the next collaboration, what we call *initiating flows*. For the non-initiating roles the implication of these flows is that they must be ready to respond as indicated by the dashed lines in Fig. 8.8(a). This represents what we call *responding flows*, to be elaborated in Sect. 8.5. Interestingly the presence of a weak responding flow implies overlap between role behaviors, and thus it serves to indicate a potential realization problem. The problem is that events in overlapping roles may interleave in ways not specified in the choreography. Whether this is possible or not depends on the underlying communication medium linking the roles.

As we can see in Fig. 8.8(b), since A initiates C2 only after it finishes its role in C1, m2 is always sent after m1. However, if the communication link between components A and B may reorder the messages, the messages m1 and m2 may arrive in any order at B. This is a realizability problem that must be resolved. In order to resolve this kind of realizability problems we have three options:

- Resolve the problem on the level of orchestration (component design), i.e. deal with overlapping behavior in the component design, or use a communication medium that conserves the sending order.

- Modify the choreography to ensure natural strong sequence.

- Enforce the strong sequence by additional communication.

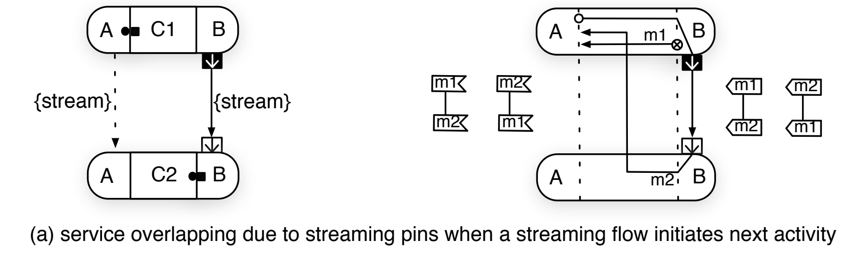
In general according to [CBvB07], when a composite role participates in two consecutive collaborations that are not strongly sequenced and plays a non-initiating sub-role in the second one there will be overlap and potential message ordering problems like in Fig. 8.9(a) and 8.9(b). If, for example the component B in C2 in Fig. 8.8 is replaced with another component D then there will be no message ordering problem due to overlapping roles in C1 and C2.

In the choreography of the city guide given in Fig. 8.4, there is no case of weak sequence localization.

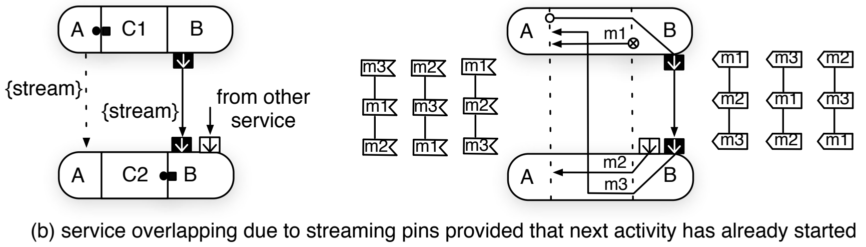
8.4.3 Control nodes and paths localization

Flows connecting a source action node C1 and target action node C2 may contain intermediate control nodes such as decisions, merges, forks and joins. In such cases, we say that two action nodes C1 and C2 are linked by a *flow-path* through a number of intermediate control nodes linked by direct *flow-steps*. Each intermediate flow-step and control node may be part of several flow-paths. The intermediate control nodes are localized according to the following rule:

1. Each *flow-path* from Ci to Cj is first localized to the initiating role of Cj.
2. Find all the flow-paths through each intermediate control node.
3. If all paths through a node are local to the same component, localize the node to that component.
4. If some paths involve an interaction to enforce strong sequencing, the node can be localized to either the source component or the target component of the path as long as all paths through the node can be localized to the same component.
5. If the node is a choice node, it must be localized to the target component of the path and it must be possible to make the choice based on information local to that component
6. If localization according to steps 3, 4 and 5 above is impossible, then there is a realization problem to resolve.



(a) service overlapping due to streaming pins when a streaming flow initiates next activity



(b) service overlapping due to streaming pins provided that next activity has already started

Figure 8.9: Message ordering problems due to the overlapping of services

Fig. 8.6(b), 8.6(c) and 8.6(f), illustrate cases where the control nodes can be localized to component B. In 8.6(g), there is a non-local choice which is not directly realizable and must be resolved as explained in Sect. 8.4.5. In the case in Fig. 8.6(d), the flow-paths though the fork node may be localized to either B or C if the additional communication also is needed to enforce strong sequencing. The fork node may either be localized to B with an interaction flow to C or the other way around. In more complex cases having more than one intermediate node, as in Fig. 8.6(h), we can use the same approach. For example the fork node may either be localized to A or B. However, in this case we also need to make sure that all the nodes in a path should be localized to the same component, and if that is not the case a interaction flow is needed. When the fork node in Fig. 8.6(h) is localized to A, then we may need to use an interaction flow from the fork node to the choice node.

8.4.4 Streaming Flow Localization

Streaming pins must be assigned to roles such that output pins are assigned to the role that generates the output token, and input pins assigned to the role that receives the input token. Once the streaming pins have been localized, the flows and intermediate control nodes are localized in the same general manner as control flows, explained above. We now classify flows initiating from and/or ending on streaming pins as *streaming flows.* The following cases are possible:

- Streaming output to starting input: This kind of situation exists, for example, between the *quiz* and *group quiz* services in Fig. 8.4. Due to the streaming pins, we may have overlapping service roles, and this may cause problems as illustrated in Fig. 8.9(a). This kind of flow is akin to the weak sequencing discussed above and is indicated by the {stream} property on the corresponding responding flow.
- Streaming output to streaming input: This means that both activities will be running in parallel and that streaming inputs may interleave with other events. This may lead to overlapping roles, like in Fig. 8.9(b), causing realizability problem. As we can see, m1 is the last message from B to A in C1. However, B may send m3 to A before m1 provided that C2 has been already started by some service other than C1. Therefore there is a possibility that B may send messages m1, m2 and m3 in any order and eventually that A receives them in any order also. This may or may not be what the composer want to specify.
- Terminating output to streaming input: This type of flow causes no overlaps and is directly realizable as long as C2 is ready to accept the token from C1.

For streaming flows we distinguish two main cases:

- Local streaming: the flows are entirely within one component (composite role). This can be coordinated within the component and will be unproblematic in most of the cases. We note here that it is desirable that streaming flows are local to one role.
- Global streaming: a flow that implies communication. This may be problematic and needs careful consideration.

As a method guideline global streaming flows should preferably be encapsulated in collaborations.

8.4.5 Initiative Choice Localization

Fig. 8.5 defines the intended behavior of an initiative choice. There is no way to prevent the initiatives from happening simultaneously, so-called

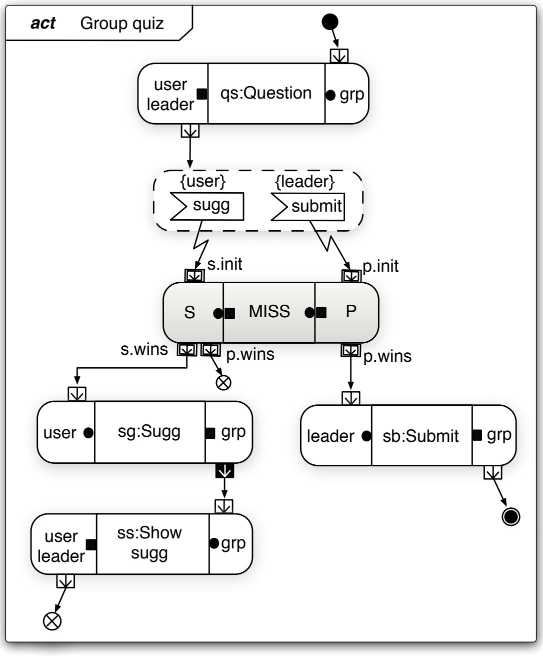


Figure 8.10: Mixed initiatives resolution in flow-localized choreography

mixed initiatives. This requires additional resolution behavior and therefore initiative choices are not directly realizable. The resolution strategy will depend on the problem at hand and is difficult to solve in a general way. In stead one may develop a library of alternative solutions to select from. One way to handle such situations is to assign primary and secondary priorities to the conflicting partners and let an initiative from the primary side be accepted in all cases. For the secondary side, this means that it must be prepared to receive a primary initiative even after it has issued an initiative itself, and obey the primary one; the secondary is in this case discarded. This feature can be modeled in a flow-localized choreography using a mixed initiative resolution building block, called MISS in [KSH09].

The MISS building block is used in the group quiz service as shown in Fig. 8.10 (in terms of flow-localized choreography). Fig. 8.11 illustrates the same using the swim-lane style of Arctis. In the cases of conflicts, a priority is given to the *leader*. In the Arctis style choreography, we have chosen to represent the lead-user and ordinary users as distinct roles in order to show the difference, we have also included user interface blocks [KKB11] where the *sugg* and *submit* events are captured. The group quiz service is started

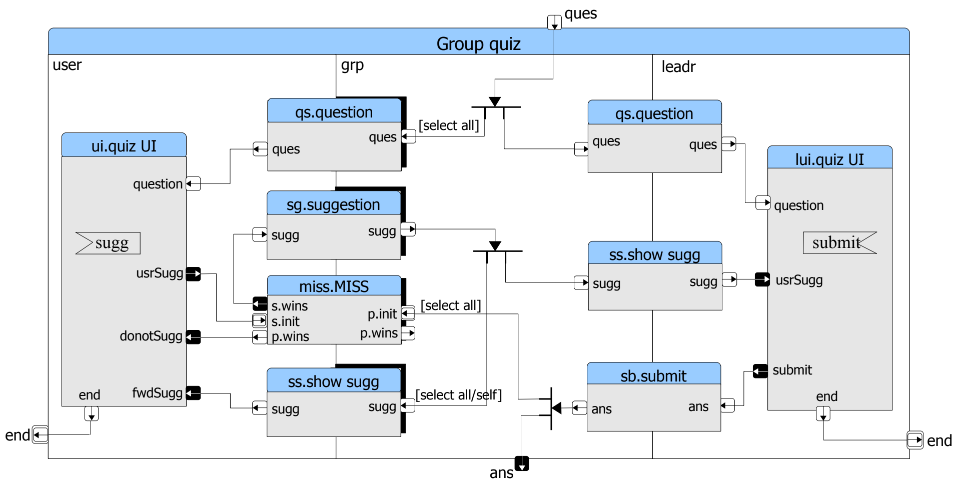


Figure 8.11: Arctis style modeling of the Group quiz service

when a group receives a question via the *ques* starting pin. The question is then forwarded via the *question* service to all users (shown by multi-session in Fig. 8.11) and also to the leader. When users start suggesting answers to the question via the *suggestion* service, the group forwards the suggestions to the other users (shown by session select statement, for detail refer [Kra08]) and to the leader. In the case of conflicting initiatives, i.e. the user is suggesting an answer at the same time as the leader is submitting, priority is given to the submitted answer (the group is given a primary role in the MISS service) and the user role is prevented from suggesting any further answers via the *p.wins* pin at the user role.

8.5 Deriving Components

Given a flow-localized choreography the next step is to derive a local activity flow for each component type, the so-called orchestration, consisting of the actions and control nodes that are local to the component, i.e. the collaborative actions and local action as well as the fork, join, decisions and merge nodes. This is done by making a projection of the flow-localized choreography onto the components in the following way:

- For each component make a copy of the flow-localized choreography
- Replace each collaboration action and local action where the component is not participating by a no-operation node, and keep all the

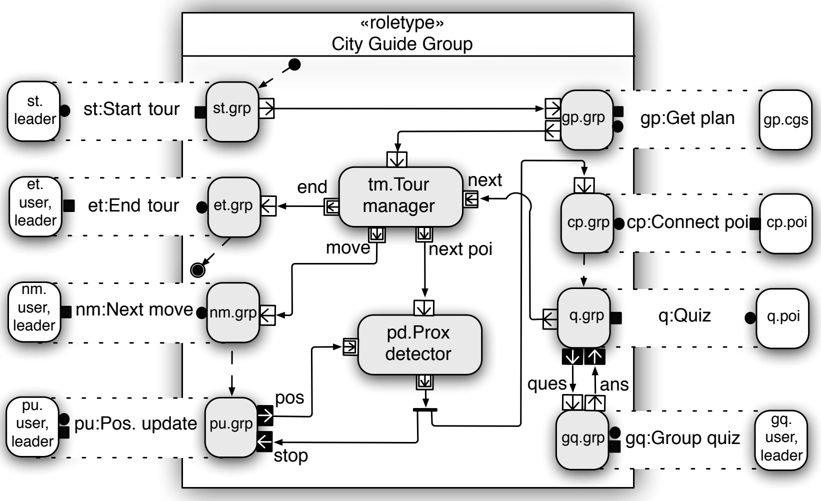


Figure 8.12: The city guide group component models

collaborations and local actions where it participates.

- Include all the flows that are completely localized to the component type. These types of flows are called *local flows* (local initiating and local streaming flows). Note that local actions and control nodes will be associated with such flows.

- Flows that cross the component boundary and therefore involve communication, are called *interaction flows.* Such flows map to send event actions on the sending side and accept event actions on the receiving end.

- Flows that are completely external to the component, are marked as *responding flows.* The responding flows determine when the component must be ready to participate in collaborations initiated by other components, i.e. collaborations where the component plays a non-initiating role.

The resulting component types for *group* and *leader* are shown in Fig. 8.12 and Fig. 8.13 respectively. Note that we have kept reference to the external collaborations for later use during composition. Inside the role actions there will be send actions and receive actions for the interaction

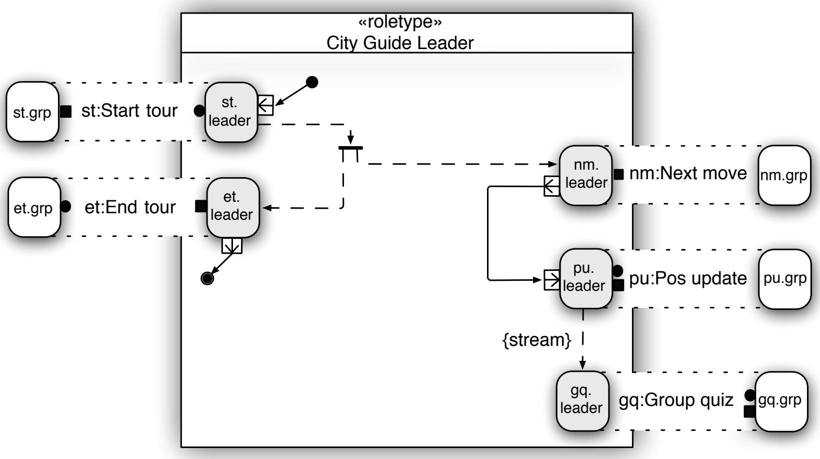


Figure 8.13: The city guide leader component models

flows of the collaborations. The component models have local flows representing local control flows and responding flows (denoted by dashed lines) representing flows external to the component. Note that a responding flow may be a simple edge or a complex flow involving external control nodes as well as no-op nodes representing collaborations and local actions where the component is not participating.

Responding flows in the group component type are mostly simple responding flows. A responding flow between *cp.grp* and *q.grp* specifies that the group should be ready to participate in a quiz session with a POI once the group has initiated a *connect poi* collaboration. A responding flow between *nm.grp* and *pu.grp* represents the case where the group informs the user about a next move, and then should be ready to participate in position update collaborations.

There are several responding flows in the *Leader* component type in Fig. 8.13 representing flows involving external control nodes as well as no-op nodes that have been removed from the diagram. For example a responding flow between *pu.Pos update* and *gq.Group quiz* involves three no-operation nodes (*pd.Prox detector*, *cp.Connect poi* and *q.Quiz*, c.f. Fig. 8.4) and one external fork.

Responding flows that have {weak} or {streaming} properties needs special attention since they may cause activity overlaps and thus potential race situations, that must be resolved. Note that there may be several possible responding flows ending on a collaboration activity. This means that tokens can arrive along the different paths in different order. Each responding path may be shortened and simplified into the shortest and most direct path representing the earliest situations when a token may arrive. For the detailed algorithm on simplifying a responding path, c.f. [KLB10].

The resulting component models may now be used to compose more composite components and systems. They may also be translated separately into equivalent UML state machines from which corresponding application code can be generated using existing tools such as Arctis.

8.6 Discussion and Related Work

In the domain of reactive systems it has been common to use interaction diagrams in one form or another to define global behaviors. UML interaction overview diagrams (IOD), or high level MSC diagrams may then be used for flow-global choreography [Dec09]. Compared to the IODs, ADs allow the representation of roles as partitions of activities which is useful information that helps to understand and analyze the global behavior at the level of choreography. According to the UML specifications in [OMG, p. 512], IODs are specialization of AD that show control flow between set of interactions. IODs can define sequential and parallel composition of interactions, as well as loops and alternatives. However, IODs excludes many rich modeling constructs available in activity diagram such as interruptible regions, flow final nodes, and streaming nodes [Whi10]. Interruptible regions of activities are useful for modeling initiative choices, exception handling and external priority interrupts. Note that there is an interaction operator *break* in UML, but it can only be applied in a sequence diagram not at the interaction overview level. Streaming pins are useful to model interactions among concurrent actions at a high level (as discussed in Sect. 8.3 between *quiz* and *group quiz*). In principle it is possible to use sequence diagrams for elementary collaborations and activity diagrams for choreography, since call behavior actions can call behavior defined by interactions. This has been done in [CBvB07], but but exactly how to combine the two forms is not fully defined in UML. The realizability problems associated with weak sequencing and initiative choices we have discussed here are the same as those identified in [CBvB07] and therefore not particular to activity diagrams. We have found activities to provide a good combination of readability, completeness, and automation potential. This combined with the benefit of staying within one notation are the main reasons for our decision to use activity diagrams throughout.

Authors in [RGG03] identify the needs to encapsulate the interaction behavior of distributed components by so-called collaboration modules. They have suggested collaboration based design with a tighter integration between interaction and state diagram models, and created a specific language called CoSDL to define collaborations. The global behavior specified by the composition of such collaborations modules in CoSDL is at the flow-localized level. The CoSDL is aligned to SDL [SDL].

Use Case Maps can be used for both flow-global and flow-localized choreography [Cas05, Buh98]. Related work in the embedded systems and web-services domains [KP06, BGG^{+}06, QZCY07, SB09, MH05] has concentrated on flow-localized choreography using interactions and activity diagrams. Most of this work assumes a manual derivation of components. Unlike most approaches [BO05, MH05, SB09, WRS^{+}09], we encapsulate interactions in collaborative building blocks. Most of the approaches are only control flow oriented, except [KP06] which considers data as well. Initiative choices, called mixed initiatives in [BF04], which are normal and unavoidable cases in many reactive systems are handled only by us and [KP06]. There are different techniques to check the conformance between choreography and orchestration. In [SB09], behavioral equivalence is used. The equivalence is checked between the labeled transition system (LTS) for the parallel composition of orchestrators and the LTS of the choreography. The authors in [BGG^{+}06, KP06] use the notion of bisimulation [Mil89] for the conformance and trace equivalence is used in [QZCY07].

Some of our previous work has been concentrated on choreography where all flows are localized to participating roles [Kra08] which is supported in the Arctis tool [KSH09]. Note that our flow-localized choreography is semantically equivalent to Arctis models, and therefore Arctis can provide a tool basis for further analysis and state machine synthesis. However, the Arctis tool currently generates complete systems and not component types that may be used in subsequent component based system composition at the modeling level.

The flow-global choreography avoids the layout restriction of Arctis (and the swim-lane notation) by representing the participating roles inside the actions as proposed in [Cas08]. In [CBvB07], the authors suggested a classification of realizability problems and some criteria for detecting them at the flow-global level. In this paper we elaborate on localization i.e. how flow-global choreography is mapped to flow-localized choreography and further to orchestration of distributed component types with external interfaces. We believe the concept of responding flows is novel and that {weak} responding flows provides a new way to identify potential realization problem. This will be addressed in a forthcoming paper.

8.7 Conclusion

A general approach for the definition of flow-global choreography of services is presented where the global behavior of a service is specified by linking together actions representing collaborative and local behavior. From the flow-global choreography we derive flow-localized choreography models that may be extensively analyzed using existing tools such as Arctis. Distributed component implementations may then be automatically generated. Our work is unique in its combination of: 1) the development trajectory it covers; 2) the use of the full power of AD; 3) the support for reusable building blocks both on the level of collaborations and on the level of components.

Responding flows are introduced to represent flows external to a component. This makes it possible to synthesize state machines and implementations for each component separately, and also provides a key to identify a class of realization problems. Components may be stored in libraries and reused in subsequent system composition. Component types provide information about their interface collaborations that may be utilized for compatibility checks, dynamic linking and binding of component instances. In this way our approach supports reuse and composition both on the level of collaborative building blocks and on the level of components. Further work is planned to provide full tool support and to elaborate on component composition.

Although we have not discussed web service in this paper, we can compose them using the same general approach by encapsulating web services in collaborations. Related works on encapsulating web services in Arctis building blocks is described in [KSB09].

References

[BF04] R. Bræk and J. Floch. Ict convergence: Modeling issues. In *In System Analysis and Modeling (SAM), 4th International SDL and MSC Workshop, pages 237–256, Ottawa, Canada.*, 2004.

[BGG^{+}06] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration conformance for system design. In *COORDINATION, volume 4038 of LNCS*, pages 63–81. Springer, 2006.

[BO05] M. Dumas Barros, A. and P. Oaks. A critical overview of web service choreography description language (WS-CDL). BPTrends, March 2005.

[Buh98] R. J. A. Buhr. Use case maps as architectural entities for complex systems. *IEEE Transactions on Software Engineering*, 24:1131–1155, 1998.

[Cas05] Humberto Nicolás Castejón. Synthesizing state-machine behaviour from UML collaborations and use case maps. In *SDL Forum*, pages 339–359, 2005.

[Cas08] Humberto Nicolás Castejón. *Collaborations in Service Engineering: Modeling, Analysis and Execution.* PhD thesis, Norwegian University of Science and Technology, 2008.

[CBvB07] Humberto Nicolás Castejón, Rolv Bræk, and Gregor von Bochmann. Realizability of collaboration-based service specifications. In *Proceedings of the 14th Asia-Pacific Software Engineering Conference*, pages 73–80, Washington, DC, USA, 2007. IEEE Computer Society.

[Dec09] Gero Decker. Realizability of interaction models. In *1st Central-European Workshop on Services and their Composition*, 2009.

[Erl07] Thomas Erl. *SOA: principles of service design.* Prentice Hall Press, Upper Saddle River, NJ, USA, first edition, 2007.

[KB09] Surya Bahadur Kathayat and Rolv Bræk. Platform support for situated collaborative learning. In *International Conference on Mobile, Hybrid, and On-line Learning, 2009. ELML'09.* IEEE Press, 2009.

[KKB11] Frank Alexander Kraemer, Surya Bahadur Kathayat, and Rolv Bræk. Unified modeling of service logic with user interfaces. *International Journal of Cooperative Information Systems (IJ-CIS)*, 20 (2):177–200, 2011.

[KLB10] Surya Bahadur Kathayat, Hien Nam Le, and Rolv Bræk. Automatic derivation of components from choreographies - a case study. In *International conference on Software Engineering*, Phuket, Thailand, 2010.

[KP06] Raman Kazhamiakin and Marco Pistore. Choreography conformance analysis: Asynchronous communications and information alignment. In *Proceedings of the 3rd International Workshop on Web Services and Formal Methods.*, volume 4184 of *LNCS*, pages 227–241. Springer, 2006.

[Kra08] Frank Alexander Kraemer. *Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks.* PhD thesis, Norwegian University of Science and Technology, 2008.

[KSB09] Frank Alexander Kraemer, Haldor Samset, and Rolv Bræk. An automated method for web service orchestration based on reusable building blocks. In *Proceedings of the 7th International IEEE Conference on Web Services (ICWS)*, pages 262–270. IEEE Computer Society, July 2009.

[KSH09] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann. Tool support for the rapid composition, analysis and implementation of reactive services. *Journal of Systems and Software*, 82(12):2068–2080, 2009.

[MH05] Jan Mendling and Michael Hafner. From inter-organizational workflows to process execution: Generating bpel from ws-cdl. In *OTM 2005, LNCS 3762*, pages 506–515. Springer, 2005.

[Mil89] R. Milner. *Communication and concurrency.* Prentice-Hall, Inc., 1989.

[OMG] OMG. Unified Modeling Language 2.1.1 Specification (superstructure 07-02-05) 2007. http://www.omg.org.

[QZCY07] Zongyan Qiu, Xiangpeng Zhao, Chao Cai, and Hongli Yang. Towards the theoretical foundation of choreography. In *Proceedings of the 16th international conference on World Wide Web*, pages 973–982. ACM, 2007.

[RGG03] Frank Rößler, Birgit Geppert, and Reinhard Gotzhein. Cosdl - an experimental language for collaboration specification. In *Proceedings of the 3rd international conference on Telecommunications and beyond: the broader applicability of SDL and MSC*, SAM'02, pages 1–20, Berlin, Heidelberg, 2003. Springer-Verlag.

[SB09] Gwen Salaün and Tevfik Bultan. Realizability of choreographies using process algebra encodings. In *Proceedings of the 7th International Conference on Integrated Formal Methods*, IFM '09, pages 167–182, Berlin, Heidelberg, 2009. Springer-Verlag.

[SDL] ITU-T Recommendation Z.100: Specification and Description Language (SDL). (2002).

[Whi10] Jon Whittle. Extending interaction overview diagrams with activity diagram constructs. *Software and Systems Modeling*, 9:203–224, 2010.

[WRS$^+$09] S. Wieczorek, A. Roth, A. Stefanescu, V. Kozyura, A. Charfi, F. M. Kraft, and I. Schieferdecker. Viewpoints for modeling choreographies in service-oriented architectures. In *WICSA/ECSA*, pages 11–20. IEEE, 2009.

Chapter 9

Paper 4

Title: Automatic Derivation of Components Using Choreographies - A Case Study.

In: Proceeding of the International Conference on Software Engineering (SE 2010). Phuket, Thailand.

Publisher: GSTF Digital Library.

Automatic Derivation of Components Using Choreographies - A Case Study

Surya Bahadur Kathayat, Hien Nam Le, and Rolv Bræk
Norwegian University of Science and Technology (NTNU)
Department of Telematics
Trondheim, Norway
Email:{surya, hiennam, rolv.braek}@item.ntnu.no

Abstract

In this paper we use a model driven development approach for the derivation of the component behaviors from choreography models. We consider the reactive system domain where services are collaborative in nature involving one or more active components that may take initiatives on their own. Two different forms of choreography models having different level of details are used. An algorithm is proposed to automatically derive reusable component types utilizing these choreography models. The approach is illustrated with a case study - the European Rail Traffic Management System.

9.1 Introduction

Choreography defines observable behavior from a global perspective. This is normally a kind of work-flow with a desired ordering of activities (services). Component behavior on the other hand defines the local behavior of a component. The choreography models are needed to define and analyze the overall service behavior, whereas the component models are is needed to completely define component behaviors for implementation.

Ideally the choreography should be sufficiently precise and complete that component behaviors can be automatically synthesized. This would enable a choreographer to work mainly on the level of choreography, focusing on the intended global behavior, and not the detailed component behavior. For this to be possible one needs suitable building blocks representing reusable functionalities and composition mechanisms for defining the intended global behavior completely and precisely, and one needs a systematic way to go from global behavior and enable automatic synthesis of component behaviors in practical systems and service development. The situation is even more complex in reactive systems domain where a service behavior normally cross-cuts the distributed components and involve a collaboration among components. In stead of requesting or providing a service like in web-services, participating components are active entities that may take initiatives on their own [BF04] and play their part in order to achieve a service goal.

In this paper, we use the combination of UML collaborations and activity diagrams to specify the global behavior [Kra08]. We consider choreographies with different level of details [CBvB07, KB09]. One is called *flow-global choreography* where all the variables, control constructs, and flows connecting activities are not-localized to any participating roles and thus are more abstract. The other is called *flow-localized choreography* where such flows are localized to particular roles and thus are more detailed. The flow-global specification is less detailed but still complete in terms of the intended behavior seen from a choreographers or end-users point of view while the flow-localized specification is sufficiently detailed to allow extensive analysis and to automatically synthesize the dynamic behavior of component types [KB09]. In this paper, we elaborate a case study showing on how the dynamic behavior of each of the system components can be derived from flow-global choreography models.

The structure of the paper is as following. Sect. 9.2 describes the case study. Choreography models are described in Sect 9.3. Methods of component derivation from choreography models are illustrated in Sect. 9.4. Related work is given in Sect. 10.6 and conclusion in Sect. 10.7.

9.2 A Case Study

We illustrate our approach using the specification for the European Rail Traffic Management System (ERTMS) [ert].

While moving in a physical geographical region, a train must always be supervised by a radio block controller (RBC) whose responsibilities are to monitor and control all train movements within its region. When a train starts its journey, the train will first contact a RBC to register its movement and repeatedly report its current position to the RBC. The RBC will continue issuing movement authority (MA) to the train, which specifies a safe distance that the train can travel. During its predefined journey, the train may travels across more than one RBC regions; this means the supervision of the train movement is handled by more than one RBCs.

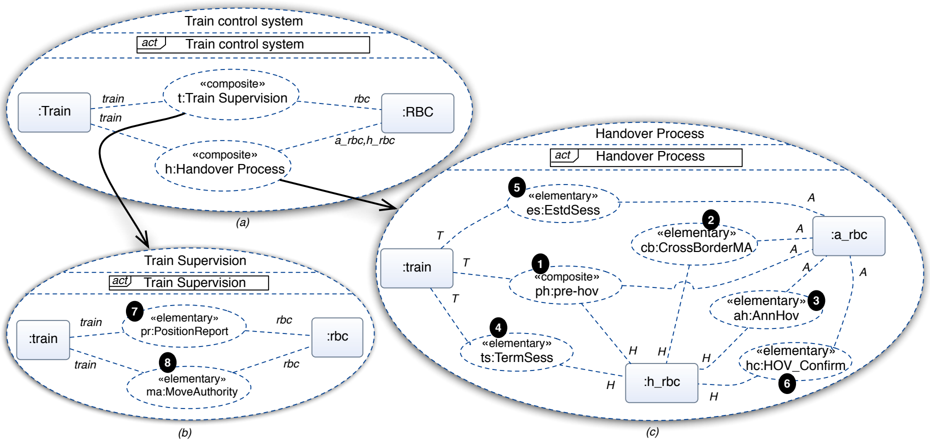


Figure 9.1: Roles and Sub-collaborations in Normal Train Control

When the train crosses a border between two regions, the RBC of the current region will handover its control to the next RBC.

We identify two main control states of a train: normal control state and handover control state. When in a normal control state the train will report its position to the RBC and receive the MA from the RBC. The new MA should be send to the train before the current MA expires. The train control state will change to handover control state when the train is about to exit the current geographical region controlled by the current RBC and entering the ad-junction region controlled by the neighbor RBC.

During the handover process, the current RBC plays a handing over role (H_RBC) and the neighbor RBC where the train is approaching plays accepting role (A_RBC). In this paper, we will assume that at any moment, the train only has the capacity to communication with one RBC (as described in the ERTMS specification, a train may have the capacity to handle two communication channel with two RBCs concurrently [ert]). The detailed handover process is the following:

- Pre-announcement handover. When the RBC detects that the train is approaching the border, the RBC will change role to H_RBC and send notification to both the train and the A_RBC to initiate a handover process.

- MA generation. After the pre-announcement handover, the H_RBC will contact the A_RBC to negotiate a cross border movement authority. The result of a successful negotiation is that a cross border MA is send to the train so that the train can travel safety to the next region.

- Handover announcement: when the train front-end reaches the border, the H_RBC initiates the handover process by forwarding the train position to the A_RBC.

- Termination of communication session. When the train rear-end crosses the border, the H_RBC will issue a termination communication session order to the train. When the train receives this order, it will disconnect from the H_RBC.

- Establish new communication session. After terminating communication with the H_RBC, the train will try to establish a new communication session with the A_RBC. After this, the train will start sending its position report to the A_RBC.

- Handover supervision. When the A_RBC receives the position report from the train, it will contact the H_RBC to inform that the handover process is completed.

When the handover process is completed, the train will switch back to normal control state (i.e., the train will now be under supervised by the new RBC).

9.2.1 Describing Service Structure

A UML collaboration diagram is well suited to model role structures and identify collaborations among the roles. For example Fig. 9.1 illustrates the global structure of train control system. An elementary collaboration (which can not be decomposed further) can be expressed as

$C =< cname > \{R_i\}$

where *cname* represents the name of the collaboration and R_i represents the set of roles involved. A composite collaboration can be decomposed into smaller collaborations. The train control system in Fig. 9.1(a) i.e. $< Traincontrolsystem > \{Train, RBC\}$ contains two sub-collaborations *Train Supervision* and *Handover Process.* These sub-collaborations are further decomposed into smaller sub-collaborations as shown in Fig. 9.1(b) and 9.1(c) respectively. The handover process as shown in Fig. 9.1(c) is a complex process involving several collaborations. During this process a RBC plays either h_rbc or a_rbc role:

The behavior of a composite collaboration is defined in an activity diagram which is referenced as $act < cname >$ for example the behavior of train control system collaboration is referenced to $act < Traincontrolsystem >$

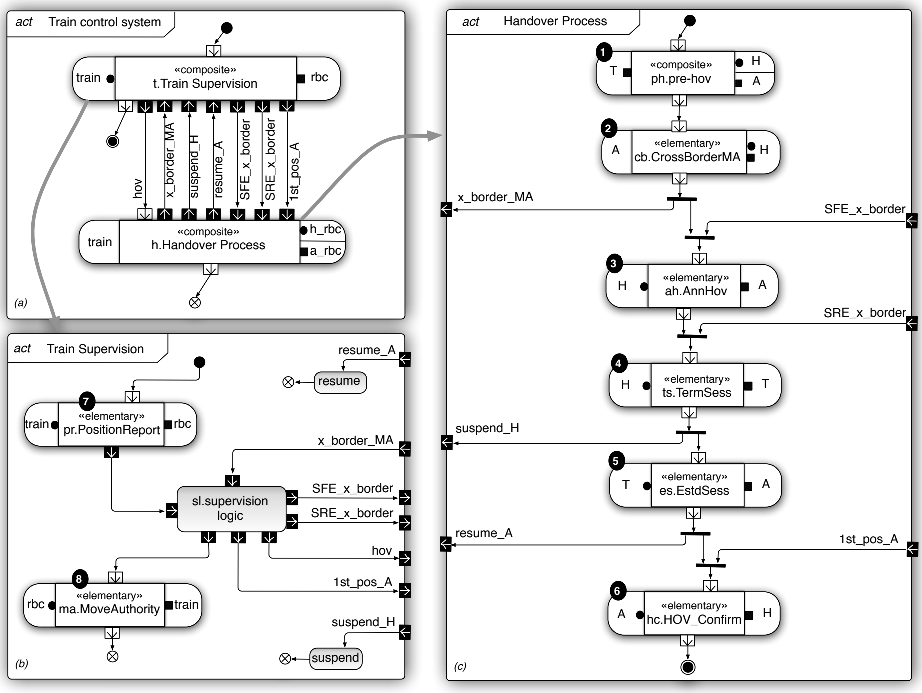


Figure 9.2: Flow-Global Choreography of Train Control System

in Fig. 9.1(a). The behavior of the elementary collaboration can be described using UML activity diagrams or sequence diagrams, however we have found activities to provide a better combination of readability, completeness, and automation potential [Kra08, KB10]. In this paper, it is assumed that the behavior of each elementary collaborations is given and are available in the library.

9.3 Choreography models

Given a library of collaborative (service) building blocks, the next problem is to define the global behavior in terms of the ordering and causality among the collaboration activities. This can be defined by linking service building blocks together, called choreography. We use UML activity diagrams for that purpose. We use choreography models having different level of details as following:

9.3.1 Flow-global choreography

Flow-global choreography defines the intended global execution ordering by connecting actions, that either represent the behavior of a collaboration or a local activity, by flows that are not localized to any particular role. They are more abstract avoiding details of localization and resolution of coordination problems. The flow-global choreography specification of the train control system corresponding to its structures Fig. 9.1(a), 9.1(b) and 9.1(c) is shown in Fig. 9.2(a), 9.2(b) and 9.2(c) respectively. For simplicity variables and data are omitted in Fig. 9.2.

In Fig. 9.2, the participating roles of a collaborations are indicated by partitions indicated by using solid lines in each action symbols. The initiating and terminating roles of a collaborations are indicated using dots and squares respectively. Actions are connected via different types of pin [1] and control elements such as forks and joins. Note that pins to an action and control elements in flow-global specification are not localized to any participating roles. The details of flow-global specification can be found in [KB10].

9.3.2 Flow-localized choreography

The flow-localized choreography is sufficiently detailed to allow extensive analysis and to automatically synthesize the behavior of component types. The flow-localized choreography of the *handover process* is shown in Fig. 9.3 which is derived from flow-global choreography of the *handover process* in Fig. 9.2(c) by applying the localization rules defined in [KB10]. In the flow-localized specification, we may see the following:

- pseudo states N such as initial and final nodes, and control nodes CN such as fork and join are localized to particular roles.
- actions A representing tasks such as local activities LA and collaborative activities C. Local activities are assigned to particular roles. Service pins in collaborative activities are localized to particular role.
- parameter nodes PN i.e. pins at boundary, giving a possibility to interact with other activities.

[1] An *initiating pin*, represented as unfilled incoming pin, will start the called activity. *Terminating pins*, represented as unfilled outgoing pin, will terminate the called activity. *Streaming pin*, represented as filled incoming and outgoing pins, can pass tokens while the called activity is active i.e., a flow between two actions connected by streaming pins allows the called activities to interact without being stopped.

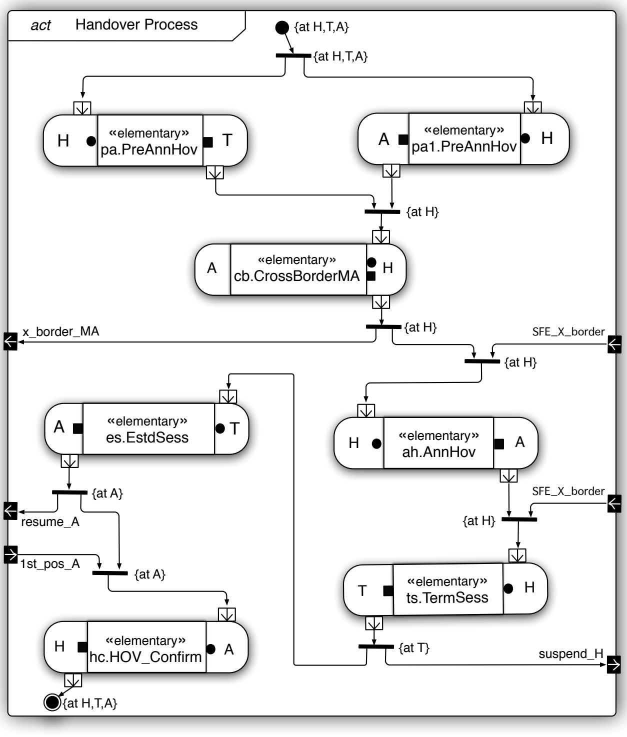


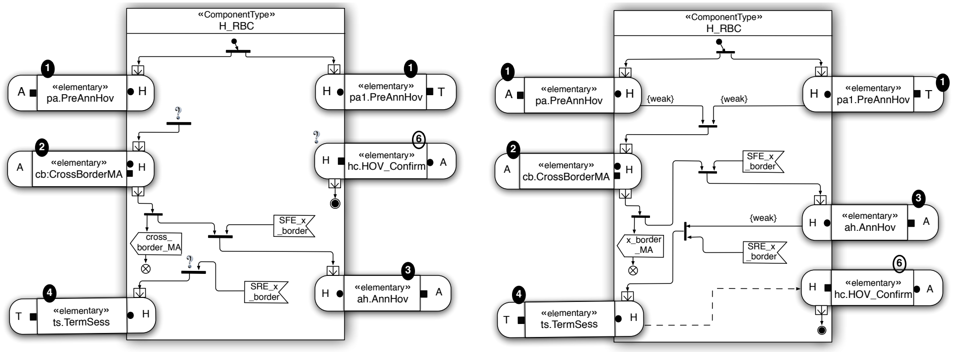
Figure 9.3: Flow-localized Chor. of Handover Process

- control flow lines $FL(x - y)$ connecting x to y where x, y represents any of localized N, CN, A and PN.

During the localization process one may also detect different realizability problems that may occur in a distributed realization that must be resolved. Different resolution strategies may be followed: either modify the flow-global specification, resolve in the flow-localized choreography or resolve during component design [KB10].

9.3.3 Strong, Weak, and Responding flows

The control flows can be classified into three different types: strong, weak and responding [KB10]. A strong control flow from $C1$ to $C2$ specifies that $C1$ will be completely finished before starting $C2$. In Fig. 9.2(c), actions are connected by strong control flows. A weak control flow from $C1$ to $C2$ is used if $C2$ has been initiated by a non-terminating role $r1$ in $C1$ while



(a) after steps 1 and 2 (a) after step 3

Figure 9.4: Component type derivation algorithm

another role $r2$ in $C1$ has not been completed i.e., weak flows imply some overlap between $C1$ and $C2$.

When the flows are localized to participating roles, there are at least two possibilities. In a first case, initiating flow of $C2$ is local to a component performing the terminating role of $C1$ (i.e. performing the last action). This flow is tagged with a property {strong}. In a second case, the flow between $C1$ and $C2$ are not completely localized to one component (called *interaction flows* [KB10]), and the initiating role of $C2$ either terminates $C1$ or sends the last message in $C1$. In this case, a coordination messages may be needed to ensure strong sequencing semantics. Use of coordination messages may however add a communication overhead. Such flows may be replaced as {weak} flows by connecting service roles that participates in $C1$ and initiates $C2$.

A completely localized control flow to a component for example from $r1$ in $C1$ to $r1$ in $C2$, may have corresponding external flows for example from $r2$ in $C1$ to $r2$ in $C2$. These external flow are called responding flows and a flow from $r1$ in $C1$ to $r1$ in $C2$ imply that $r2$ in $C2$ must be ready to respond once $r2$ has finished its part in $C1$.

9.4 Deriving Component Models

In general, component is defined as a unit of computation which is developed to be composed with other component [Szy02]. Component development is a process to produce a component, that includes both internal behavior and interfaces, to perform specific functionality. In this section, we will discuss our algorithm to derive the component based on both flow-global and flow-localized choreography. The functionality of a component is identified via the roles that this component will perform. The internal behavior and interfaces of the component will be derived based on the flow-localized choreography. If the behavior of the component functionality is dependent on other components, the interaction and responding flows will be used to capture these dependencies. The component type derivation algorithm is described in following steps: (1) identifying the roles that the component will have; (2) keeping the feature entities of the component type such as collaboration roles, local actions, control nodes and the flows that are local to the component; (3) deriving the component dependencies by adding localized interaction flows and responding flows; and (4) deriving the interfaces of the component type. The detailed discussion is in the following:

1. *Identifying the roles CT_R of a component type CT.*

   A component type contains set of roles assigned to it. For example in a component type RBC in a train control system in Fig. 9.1(a) $CT_R = \{rbc, h\_rbc, a\_rbc\}$, and in a component *h_rbc* in Fig. 9.1(c) $CT_R = \{H\}$.

2. *Keeping the feature entities associated with the component type CT.*

   (a) Mark each local actions LA_i and control nodes CN_i which are localized to CT. This is straight forward because in the flow-localized choreography these entities are bound to specific roles.

   (b) Mark each collaborations C_i in a flow-localized choreography where *a role in* $C_i \in CT_R$.

   (c) Mark each localized control flow $FL(x-y)_i$ where source and target, i.e., x and y, of a flow are localized to CT.

   (d) For each flows in step 2c containing a parameter node PN i.e flows like $FL(PN-y)_i$ or $FL(x-PN)_i$: replacing incoming PN i.e. PN in $FL(PN-y)_i$ with a receiving event, and replacing outgoing PN i.e. PN in $FL(x-PN)_i$ with a sending event.

   The resulting H_RBC after this step is shown in Fig. 9.4(a) where "?" symbols indicate inconsistent places.

3. *Deriving component dependencies by adding localized interaction flows and responding flows.*

   (a) For each interaction flows $FL(x-y)_i$ in the flow-localized choreography where target of the flow is bound to $R_i \in CT_R$: if the flow is not directly realizable according to [KB10] keep enforced strong sequence using coordination messages, otherwise traverse backward until there is a collaboration C_j where a role of the collaboration $R_j \in CT_R$, then replace the flow with $FL(R_j - R_i)$ and mark the property of this flow as {weak}.

(b) For each collaborations C_i selected in step 2b where *a collaboration role* $R_i \in CT_R$ is playing as non-initiating role (marked as a non-filled eclipse with a number in Fig. 9.4), identify the responding flows as following: In the flow-global choreography identify the equivalent role of R_i, traverse backward (consider all the possible backward flows if there are nodes like *merge or join* along the flow) to a preceded collaboration that has a role R_j belonging to component type CT, i.e., $R_j \in CT_R$. Then in the component type, add a responding flow from R_j to R_i, i.e., $FL(R_j - R_i)$. If R_j can not be found, add a responding flow from the initial node to R_i, i.e., $FL(N_{init} - R_i)$.

The resulting H_RBC is shown in Fig. 9.4(b) where responding flows are represented as dashed lines.

4. *Deriving the interfaces:* The interfaces of component can be categorized into two different types: interface behavior associated with collaborations (called *semantic interface* [JFS08]), and message passing interface associated with sending and receiving events. Due to the space limitation, details is not explained further.

9.5 Discussion and Related Work

Notations and semantics. Various notations have been proposed in the literature for specifying global requirements. Some examples are UML activity diagrams and interaction overview diagrams, Use Case Maps (UCM), the Process Definition Language (XPDL), the Business Process Modeling Notations (BPMN), and the Web Services Choreography Description Language (WS-CDL). The common concepts in most of these is that application behavior can be decomposed into several activities, and further into sub-activities. Different mathematical approaches has also been used to represent choreography as well as components behaviors, for example labeled transition system in [KP06], set of conversations in [BGG$^+$06], and activity traces in [QZCY07]. Derivation of components from global specification have been addressed by some works in [QZCY07, SB09, MH05]. Most work assume a manual derivation of components. Mixed initiatives [BF04], which is normal and unavoidable case in reactive system are not considered in most existing work. Most of these works also assume that the primitive activities are allocated to a single system component for example in terms of sending and receiving events. However we consider collaborations that encapsulate interactions as the basic building block of the behavior.

Flow-Global (F-G) and Flow-Localized (F-L). There are *interaction models* such as WS-CDL, and *interconnection models* such as BPMN for describing a choreography [Dec09]. *Interaction* models show interaction among the services, and *interconnection* models show the activities belonging to some role and interconnection to other roles specifying communication and dependencies. Our F-G choreography is like a *interaction model* where interaction among the services is specified and F-L choreography is like a mix of *interaction* and *interconnection* models.

Realizability issues. When a choreography is mapped to a set of components, the global behavior emerging from the composition of components should conform to the behavior specified in global choreography. Traditionally different equivalence mechanisms are used: *behavioral equivalence* in [SB09], *bisimulation* in [BGG$^+$06, KP06], and *trace equivalence* in [QZCY07]. Such techniques may also be used in our models. However we do the realizability checks during the localization process, in [KB10]. Interestingly we observe that the presence of {weak} responding flows indicate a potential realization problem, and should be resolved.

9.6 Conclusion

This paper has illustrated through a case study how UML activity diagram can be used to specify the choreographies of collaborative reactive services. Flow-global choreography are intended to be easily understandable by an end users or domain experts. Realizability problems are checked and resolved using in [KB10] during the mapping process from flow-global to flow-localized choreography. The flow-localized choreography is more detailed. An component derivation algorithm is proposed and elaborated through the case study - ERTMS. The derived components may then be stored in library and then be composed during systems design.

We plan to explore in detail how the derived component types can be reused and composed in a system design. Since the derived components contain information about external collaborations that may be used as contracts during composition, we believe that the resulting components can be composed with a high degree of interface compatibility.

References

[BF04] R. Bræk and J. Floch. Ict convergence: Modeling issues. In *In System Analysis and Modeling (SAM), 4th International SDL and MSC Workshop, pages 237–256, Ottawa, Canada.*, 2004.

[BGG$^+$06] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration conformance for system design. In *COORDINATION, volume 4038 of LNCS*, pages 63–81. Springer, 2006.

[CBvB07] Humberto Nicolás Castejón, Rolv Bræk, and Gregor von Bochmann. Realizability of collaboration-based service specifications. In *Proceedings of the 14th Asia-Pacific Software Engineering Conference*, pages 73–80, Washington, DC, USA, 2007. IEEE Computer Society.

[Dec09] Gero Decker. Realizability of interaction models. In *1st Central-European Workshop on Services and their Composition*, 2009.

[ert] FIS for the RBC/RBC Handover. http://www.era.europa.eu/Document-Register/Documents/SUBSET-039v2.3.0.pdf, Accessed November 2010.

[JFS08] S. Jiang, J. Floch, and R. Sanders. Modeling and validating service choreography with semantic interfaces and goals. In *IEEE International Symposium on Service-Oriented System Engineering, 2008*, pages 73 –78. IEEE Computer Society, 2008.

[KB09] Surya Bahadur Kathayat and Rolv Bræk. Platform support for situated collaborative learning. In *International Conference on Mobile, Hybrid, and On-line Learning, 2009. ELML'09.* IEEE Press, 2009.

[KB10] Surya Bahadur Kathayat and Rolv Bræk. From flow-global choreography to component types. In *System Analysis and Modeling (SAM)*, volume 6598 of *Lecture Notes in Computer Science*. Springer, 2010.

[KP06] Raman Kazhamiakin and Marco Pistore. Choreography conformance analysis: Asynchronous communications and information alignment. In *Proceedings of the 3rd International Workshop on Web Services and Formal Methods.*, volume 4184 of *LNCS*, pages 227–241. Springer, 2006.

[Kra08] Frank Alexander Kraemer. *Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks.* PhD thesis, Norwegian University of Science and Technology, 2008.

[MH05] Jan Mendling and Michael Hafner. From inter-organizational workflows to process execution: Generating bpel from ws-cdl. In *OTM 2005, LNCS 3762*, pages 506–515. Springer, 2005.

[QZCY07] Zongyan Qiu, Xiangpeng Zhao, Chao Cai, and Hongli Yang. Towards the theoretical foundation of choreography. In *Proceedings of the 16th international conference on World Wide Web*, pages 973–982. ACM, 2007.

[SB09] Gwen Salaün and Tevfik Bultan. Realizability of choreographies using process algebra encodings. In *Proceedings of the 7th International Conference on Integrated Formal Methods*, IFM '09, pages 167–182, Berlin, Heidelberg, 2009. Springer-Verlag.

[Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

Chapter 10

Paper 5

Title: Modeling Collaborative Learning Services - A Case Study

In: Proceeding of the 2011 International Conference on Collaboration Technologies and Systems (CTS 2011), May 23-27, 2011. Philadelphia, USA.

Publisher: IEEE Computer Society

Modeling Collaborative Learning Services - A Case Study

Surya Bahadur Kathayat, and Rolv Bræk
Norwegian University of Science and Technology (NTNU)
Department of Telematics
Trondheim, Norway
Email:{surya, rolv.braek}@item.ntnu.no

Abstract

This paper presents an approach to model collaborative learning activities. Interactions among collaborating participants in a collaborative activity are encapsulated in a special kind of building block. Such building blocks are then composed together in order to specify the ordering and causality among them resulting a learning activity-flow model. Using such activity-flow models, one can design a learning activity by applying various collaborative learning patterns such as Jigsaw and Pyramid. Our approach is illustrated using a case study of a city learning activity.

keywords: Collaborative Learning Services, Modeling, Patterns.

10.1 Introduction

Emerging information and communication technologies are supporting teaching-learning process in various ways. Learners are becoming more mobile and at the same time they are interacting and collaborating more with co-located or distributed learners having similar interests or learning objectives. Emerging technology is also opening opportunities for informal learning activities besides traditional class-room based learning. Technology is being accessible to more and more users and educational practitioners such as learners are becoming more active in creating or customizing technological solutions on their own according to their needs. However the challenge is to provide proper notations and tools supporting flexible, reusable and customizable collaborative learning activities designs.

Table 10.1: Collaborative services and associated roles

Service identified	**Collaborating entities**	**Description**
Login	User, Server	users to login to the system
Social Matching	User, Server	find the matched users
P2P Chat	User, User	peer to peer chat between users
Group Chat	User, Group	chat among group members
Group Discussion	User, Group	discussion among group members
Quiz	User, Group, POI	quiz about a point-on-interest
Positioning	User, Group	sharing location info to a group
Configuration	Teacher, Server	allows to configure a system

In this paper, we take the perspective that learning comes, not only from sitting at a desk and consuming content such as books, web pages and other materials, but also comes from being active and participating in collaborative activities with other learners and learning objects. Learning areas may be informal - such as in city wide collaborative learning as discussed in [CD09] where learners learn about a city by being in the city, and by interacting with other co-located or distributed learners. In such situations, learners collaborate and learn in social settings using groupware to support their activities.

We look into the ways to simplify the design and development of such collaborative learning activities (or interchangeably will be called services). In particular, we focus on modeling collaborative learning services and to automatically produce executables from there. Currently, IMS learning design specification [IMSa] can formally describe any design of learning activities i.e., teaching learning processes for wide range of pedagogical approaches. There are some editors and tools supporting IMS learning design specification, however several authors have pointed out that IMS learning design is insufficient (discussed in Sect. 10.6) in modeling group based and collaborative learning activities. We use UML [OMG09] for modeling such learning services.

By using the concepts of UML 2.x collaborations, it is possible to model the structure of collaborating entities in a service. UML activity diagram can then be used to specify their collaborative behavior i.e., how they interact with each other. Detailed interactions among interacting entities can be encapsulated into collaborative building blocks and such building

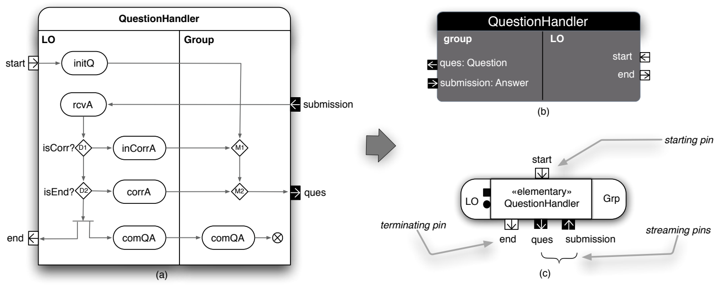


Figure 10.1: Question Handler Service Model

blocks can then be put into a library and reused to compose a new and larger learning service. Note that some of the services in a system runs in the background while others may interact with users. This means that user-interface concerns (the behavior of a user while interacting with a service via graphical user interfaces) plays significant role while designing a complete system specifications. Interestingly, using a similar approach as services, it is also possible to model and compose user-interface concerns on a high level of abstraction. From such models, service can be implemented in an automated way using code generation techniques.

The modeling style has a number of interesting properties for the development of the collaborative learning services. The building blocks lead to an incremental specification style, since they can be developed and analyzed in isolation, and serve as interfaces between experts of different domains. For example an user-interface expert may design the UI blocks and put into the library. Similarly, a service developer may develop service blocks and put into the library. Educational practitioners such as teachers may then compose a new service according to their needs putting together the available services and UI blocks in the library. Note that educational practitioners may use different collaborative learning patterns (c.f., Sect. 10.4) as guidelines while composing learning activities together.

The structure of the paper is following. Sect. 2 discusses a case study that will be used to illustrate the approach. Designing learning activities and user-interface concerns is discussed in Sect. 3. Design of collaborative learning patterns is discussed in Sect. 4. Implementation issues are discussed in Sect. 5. Related work is discussed in Sect. 6 and conclusion is given in Sect. 7.

10.2 A Case Study

City learning activity: New students, called users in the following, are just arrived in a city and are interested to collaboratively learn about different POIs in the city.

Detailed description: The Users log in to the system (optionally configured by a teacher or an instructor) using their GPS enabled handheld devices. Users may create their profiles specifying their preferences and interests. Based on their locations and profile information, the users may be able to participate in different groups and social activities and learn about different POI around the city. The user may also opt to learn individually and share the knowledge with other users thereafter. When a user is trying to find a particular point of interest (e.g., historic building in a city), a user may interact and collaborate with co-located or distributed other users having similar interests. During the collaboration, users may for instance chat, have a group discussion and may participate in a quiz about a particular point of interest. Table 10.1 shows some services that collaborating users may use during a learning activity [KB09]. Note that collaborating entities represent domain entities which play part in the service. They are also called roles in a service.

Some learning activities may be implemented by a elementary service i.e., those which can not be decomposed further, whereas some learning activities may be implemented as a composite service where a service is composed from smaller services. Some of the services shown in Table 10.1 are elementary, while some are composite. In the following section, we discuss the design and composition of elementary and composite services.

10.3 Designing Learning Activities

In this section, we start discussing the modeling of elementary collaborative activities (services) and then we discuss the modeling of associated UI concerns and the composition of UI and service models. We take a quiz service as an representative example in the following.

We use UML 2.x activity diagram [OMG09] to formally encapsulate the interactions among the collaborating entities in learning activity. Participating entities are represented as UML *ActivityPartition* elements and shown as swim-lanes for example in Figure 10.1. A collaborative activity is represented as special kind of building blocks having different types of input and output pins for instance streaming, starting and terminating types.[1] These

[1]An Initiating pin will start the called activity if it is not started yet. An Terminating

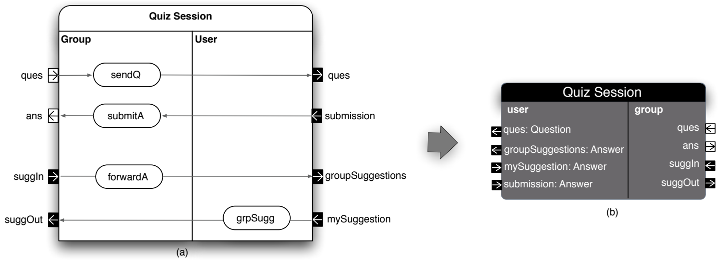


Figure 10.2: Figure 2. Quiz Session Service Model

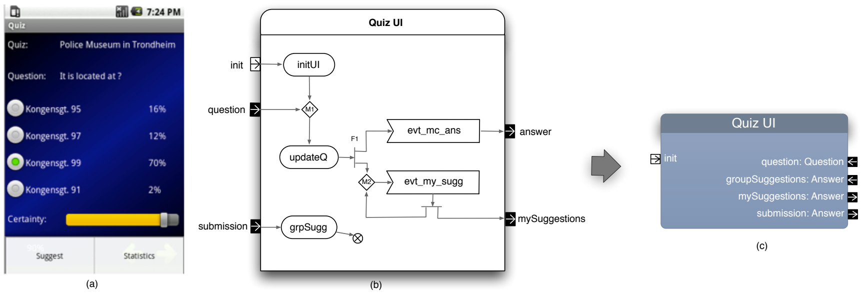


Figure 10.3: Quiz Session User Interface Model

pins are used to connect the building blocks together.

10.3.1 Designing Elementary Learning Activities

A quiz service is a composite service: it consists of two elementary services called *QuestionHandler* and *Quiz Session*. In a Question Handler service, a group and a learning object (LO) collaborate. A specification of a question handler service using UML 2.x activity diagram is shown in Figure 10.1(a).

pin will terminate the called activity. An Streaming pin can pass tokens while the called activity is active, i.e. a flow between two activities connected by streaming pins allows the called activities to interact without being stopped.

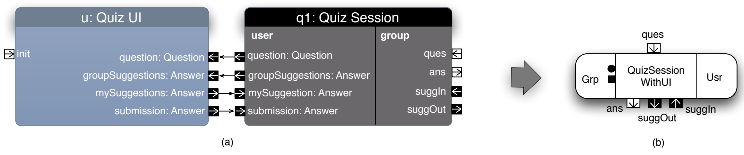
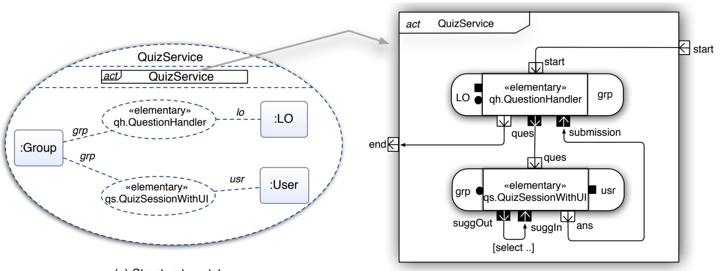


Figure 10.4: Composition of Quiz UI with Quiz Session Service Model



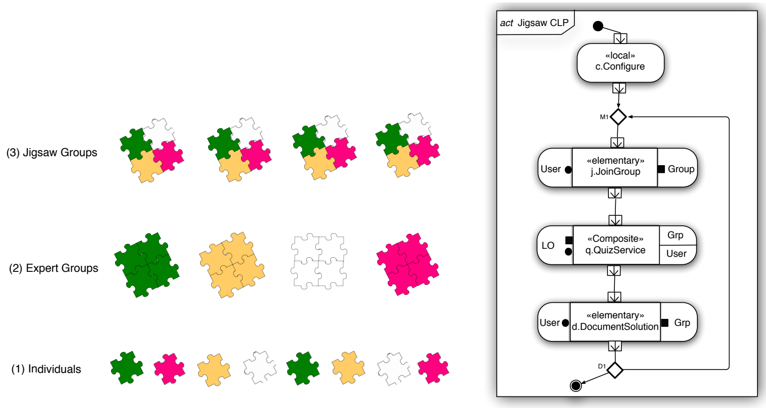
(a) Structural model (b) Bahavior model

Figure 10.5: Quiz Service Model as Composition of Quiz Handler and Quiz Session services

A *QuestionHandler* service is started with a *start* starting pin. Questions are then initialized in LO and sent to the group. These questions will be forwarded to the group members via streaming pin *ques*. When the group receives an answer via *submission* streaming pin, the group forwards it to LO. The correctness of the received answer is checked at LO and then accordingly either next question is sent to the group or completion of question-answer is indicated via *end* terminating pin.

Note that building block representations of *QuestionHandler* service are shown in Figure 10.1(b) and 10.1(c). Detailed interactions among collaborating entities are hidden in building blocks and they have pins at the boundaries which will act as interfaces or connecting points while composing two or more services together. The building blocks of type in Figure 10.1(b) are supported in our service engineering tool Arctis [KSH09] where one can see that the service pins are local to particular role (or collaborating entity). Building blocks of type in Figure 10.1(c) are more abstract representation (an extension to Arctis, proposed in [HKLB11]) where pins are not local to any participating entities, but are owned globally by a service itself. Starting and terminating service roles are represented by black filled circle and square box respectively.

Another sub-service of a quiz service is *Quiz Session* between group and users. The specification of this service is given in Figure 10.2(a) and the building block representation is in Figure 10.2(b). The service starts when a question is received at the group via *ques* starting pin. The received question is then forwarded to a user. The user may then suggest an answer to the group (via *mySuggestion* to *suggOut* pins) or receive suggestions from other members of the group (via *suggIn* to *groupSuggestions* pins). Finally the user may submit an answer (via *sumbission* to *ans* pins).



(a) Jigsaw Collaborative Learning Pattern (b) Choreography Model of Jigsaw Learning Activity

Figure 10.6: Jigsaw Collaborative Pattern and City Learning Activity

10.3.2 Designing User Interfaces Concerns

Note that users interact with *Quiz Session* service via graphical user-interfaces for example view the questions, submit suggestions and answers as shown in Figure 10.3(a). Such user-interface concerns can be encapsulated in a building block using UML activity diagram, using a same approach as we model services. Figure 10.3(b) shows the user-interface model that captures the user-interfaces concerns of a *Quiz Session* service. One can see in Figure 10.3(c) that there is only one activity partition (unlike in a service model where there are more than one partitions) representing that it is a local block.

10.3.3 Composing UI Concerns with Service Models

As shown in Figure 10.4(a), user-interface (UI) block *u.Quiz UI* and service block *q1.Quiz Session* are composed together by connecting their pins together. The pins can be connected together either manually or semi-automatically based on their name and associated data type. Refer [KKB11] for more details on designing comprehensive UI blocks for Java J2SE and Android platforms and composing UI blocks with service blocks.

Figure 10.4(b) shows the abstract representation of the composite block i.e., service and UI blocks composed together. In the following section, we assume that such abstract blocks are in the library and will be used to compose composite collaborative learning services.

10.3.4 Composing Learning Activities

Services building blocks can be composed together by connecting their pins together by specifying a flow between them. For example a composition of a *QuizService* which is composed from two elementary services *QuestionHandler* and *QuizSessionWithUI* is shown in Figure 10.5. Structural model, representing structure of the collaborating participants and the services they involved in, is shown in Figure 10.5(a). The service behavior model is shown in Figure 10.5(b). It starts from a starting pin *start* and a flow then goes to the starting pin of *QuestionHandler*. The question handler service will send the question via *ques* streaming pin to the starting pin of *QuizSessionWithUI* service. Users collaborate in the *QuizSessionWithUI* service, solve the problem and submit an answer. When *QuizSessionWithUI* service terminates, which is indicated by a terminating pin, a flow goes to incoming streaming pin of *QuestionHandler* service. As discussed in Sect. 3.1, the *QuestionHandler* service then checks the correctness of the answer and accordingly either send the new question or terminate the service which is represented by the flow line from terminating pin of *QuestionHandler* to terminating pin of a composite *QuizService*.

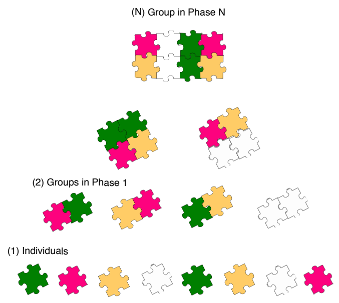
10.4 Learning Patterns

Collaborative learning patterns are defined as the formalization of good practices in structuring the sequence of collaborative (or not) learning activities or services [LPD04]. We therefore take them as guidelines while describing the composition i.e., the activity-flow of learning activities. In the following, we discuss *Jigsaw* and *Pyramid* collaborative learning patterns while composing a city learning activity.

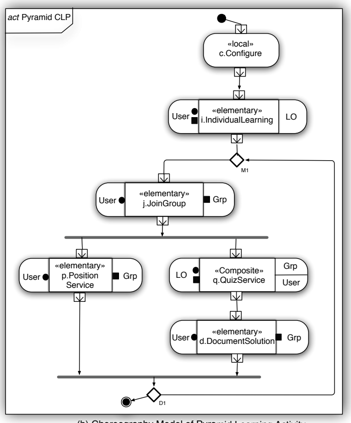
10.4.1 City Learning Activity with Jigsaw CLP

A Jigsaw collaborative learning pattern [ADH$^+$04] is shown in Figure 10.6(a). In Jigsaw collaborative learning pattern, *individuals* initially join *expert groups* where they collaborate and become experts on a particular subject or topic. (Note that different colored Jigsaw pieces represent individuals having different pieces of knowledge.) Thereafter, experts from different expert groups form a new group called *Jigsaw groups* where they share their knowledge to each other. In this way, all the users learn about all the topics to be learned.

A city learning activity based on the Jigsaw pattern can be organized as following: First, students will be divided into *expert groups* where group



(a) Pyramid CLP



(b) Choreography Model of Pyramid Learning Activity

Figure 10.7: Pyramid Collaborative Pattern and City Learning Activity

members learn about particular POI and become experts i.e., knowing detailed information about it. In each expert group, students collaboratively learn together - identify the location of the POI, solve the quizzes about that particular POI, make presentation and report together. Once students in a expert group solved their assigned tasks, they become experts and then they will be reassigned to *Jigsaw groups*. In Jigsaw groups, each member are experts on some particular POI as they come from their original expert groups. The task of each expert in Jigsaw group is to share their knowledge about the POI of which that they are experts and to learn about other POI from other experts. Once all the experts have shared their knowledge, all the students will have learned about all the POI in a city.

Figure 10.6(b) shows the learning activity-flow model as the composition of services supporting the city learning activity based on Jigsaw collaborative learning pattern. The users join in a group using *j.JoinGroup* service. Group members then collaborate, for example using *q.QuizService* and find a common solution about a particular POI and document it using *d.DocumentSolution* service. Thereafter, members in a group depart from their original expert group and join to a Jigsaw group using *j.JoinGroup* and group members collaborate and know about the point-of-interests shared by each members of the Jigsaw group. (It is assumed that nature of the quiz service will be the same in jigsaw groups and expert groups.) The process may continue until all the students know about all the point-of-interests to be learned which is indicated by last decision node D_1 in Figure 10.6(b).

10.4.2 City Learning Activity with Pyramid CLP

Pyramid collaborative learning pattern [ADH$^+$04] is shown in Figure 10.7(a) where each each student initially studies the problem individually. Thereafter they join to the larger group and share their knowledge. The smaller groups join to the bigger one and share their knowledge until all the users are in the same larger group and develop a shared knowledge.

A city learning activity based on the Pyramid pattern can be organized as following: Each individual student studies the problem i.e. locate and acquire more detail information about a particular point-of-interest(s). Thereafter some students form a group and share the knowledge about the POIs they have learned, compare their solutions and develop a common shared knowledge. After that, those smaller groups join in larger groups, collaborate and acquire the knowledge about the new POIs that other groups have initially learned. In this way, at the end all the students will develop a shared knowledge about all the point-of-interests to be learned in a city. Figure 10.7(b) shows the learning activity-flow model of a city learning activity as the composition of the services based Pyramid collaborative learning pattern. User individually learn themselves about a particular POI using *i.Individual Learning* service. Thereafter they join in a group using *j.JoinGroup* service and collaboratively learn about any POI using *q.QuizService* and document common shared solution using *d.DocumentSolution* service. At the same time, they can use other services such as *p.PositionService* which is shown in Figure 10.7(b), chat service and etc. The process may continue (i.e., groups may join to the another group and the collaborate) until all students are in a same group.

10.5 Implementations

We use a model-driven development (MDD) [Cas08, Kra08, BH93] approach for the development of collaborative learning services. Our development approach starts from abstract models which are close to the problem domain and are understandable by end users or domain experts. Such abstract models are transformed into more detailed models and so on until one can automatically generate application code. Our learning activity-flow models, which are described as the composition of services, represent models with highest level of abstraction. They represent the work-flow of learning activities. Such models uses the semantics of UML activity diagram with small notational extension which is provided through UML profile in [HKLB11]. These learning activity-flow models are platform independent and specify the pure functionality of a particular solution or an applications. Such models however may contain different types of design faults leading so called *realizability problems* in [KB10]. Therefore learning activity-flow models can be model-checked for such errors, identified errors can be resolved (if resolvable for example rules in [KB10]) or informed to the learning activity-flow designer about the potential point of problems which need to be manually resolved. After that the abstract learning activity-flow models can be automatically transformed into more detailed models using the rules in [KB10]. The resulting models can then be imported in to Arctis tool-suit [KSH09]. The Arctis tool can then be used to model-check for other realizability problems, and to generate application code for different platforms such as Java and Androids [KKB11, KSH07]

10.6 Related Works

Various standards and specifications for learning design are proposed and being used in the area of computer supported learning, in particular e-learning, mobile learning, and computer supported collaborative learning. IMS learning design [IMSa, IMSb] (grounded on EML [HMTK04]) is relevant work in the context of this paper i.e., modeling the work flow of learning activities. IMS learning design (LD) provides a framework to specify the ordering and work-flow among the learning activities in the form of unit of learning (UOL) which specifies who does what, when and with which facilities in order to reach the learning objectives. Note that creation of unit-of-learning also involves (besides the flow of activities) the bundling of all associated resources such as files, web references, learning materials, learning service configurations and etc.

Several authors have pointed out that IMD LD is insufficient to model collaborative learning activities, and accordingly propose extensions to IMS LD. Caeiro et. al. [RRN04] proposed the extension to meta-model of IMS LD by introducing the concepts of *community* to support collaborative activity. The *community* has local activities within it to support collaboration for multiple individuals. Hernandez et al in [HLVFAP$^+$06] proposed an extension to the IMS LD *service* descriptions defining a special type of service called *group service*. Authors in [YC07] however pointed the some of the limitations of the proposed preliminary concept of *group service* such as on the limited awareness and on the specification of privileged roles. Miao et. al. [MHHH05] also discussed some of the limitations of IMS learning design, in particular on modeling groups, complicated control flows and various forms of social interactions. There are some learning design editors

Table 10.2: Comparative summary of our work and IMS LD

	IMS LD	**Our Approach**
Development Approach	Top down	Bottom up
Semantics	formal (XML)	Semi-formal (graphical)
Notation	None	UML activity diagram
Learning Activities	Local activities	Local and collaborative activities
User Interfaces	No	Yes
Reusability	No	Yes
Workflow Design	Sequencing of local activities	Sequencing of local and collaborative act.
Who is it for	Developers/Teachers trained to IMD LD	Teachers knowing bases of work flows

such as: RELOAD [rel], CopperAuthor [cop], CoSMoS [Mia05], and MOT+ editor [mot]. RELOAD, CopperAutor and CoSMoS presumes that learning designer have sufficient knowledge about IMS learning design constructs and specifications. Collage tool [HLVFAP$^+$06] is a graphical tool, based on RELOAD, for authoring collaborative learning activities. MOT+ editor in addition provides some graphical representations for facilitating authoring tasks to some extent. LAMS editor [lam] uses a set of predefined (learning) activities in the library and a new activity is designed just by dragging and dropping such activities and connecting them together. It is however not compliant to IMS LD. Unlike most of the approaches, we use UML activity diagram which has intuitive and rich flow-constructs for modeling activities. For modeling collaborative learning activities, we encapsulate interaction among collaborating entities in collaborative building block and later these blocks can be composed together. This gives more flexibility compared to existing IMS LD based techniques, specially in composing collaborative activities together.

Our approach also allows multiple individuals to interact together in collaborative activities using a concept of session [Kra08]. Summary of the comparison of our work and IMS learning design related works is shown in Table 10.2. Unlike IMS LD based learning design, our approach provides the reusability of learning activity-flow models or unit of learning. We also support the modeling of user-interfaces in a similar ways as services or activities. Due to the graphical/visual nature of notations, we believe that our approach is intuitive for educational practitioners who has bases of workflows but need not be trained with formal semantics of IMS LD.

10.7 Concluding Remarks

We use UML activity diagram to model a work-flow of learning activities. With this, we aim to implement the concept of learning design which are not specifically intended for IMS learning design specifications. Such learning activities flow models capture requirements of educational practitioners and then one can go all the way down to the automatic generation of application code. Reusability and flexibility are the major benefits of our approach where services are designed and put into the library and can be reused, flexibly composed in other contexts while making a composite service. With the proposed approach, one can model (and then compose) user-interfaces and service concerns in a unified way. Our notations are intuitive for representing collaborative activities as well as UI blocks as one can see the information about the participating entities, their roles (starting or terminating or participating) and the interfaces (as connecting points) in a abstract service notation. Our approach is partially tool supported as well.

In the future work, we aim to transform our learning activity-flow models to IMS learning design (LD) compliant models. Since our models use UML semantics, there may be straight forward way to transform such UML models into IMS LD scripts, some related works are [IMSb, YC07]. For that extensions in IMS learning design may be necessary in order to support the concept of collaborative activities.

References

[ADH+04] J.I. Asensio, Y.A. Dimitriadis, M. Heredia, A. Martinez, F.J. Alvarez, M.T. Blasco, and C.A. Osuna. Collaborative learning patterns: assisting the development of component-based cscl applications. In *Proceedings of 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 218–224, 2004.

[BH93] Rolv Bræk and Oystein Haugen. *Engineering real time systems: an object-oriented methodology using SDL.* Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1993.

[Cas08] Humberto Nicolás Castejón. *Collaborations in Service Engineering: Modeling, Analysis and Execution.* PhD thesis, Norwegian University of Science and Technology, 2008.

[CD09] C. I. Canova and M. Divitini. Reflections on the role of technology in city-wide collaborative learning. *International Journal of Interactive Mobile Technologies (iJIM)*, 3, 2009.

[cop] CopperAuthor Website, http://www.copperauthor.org/. Accessed on December 2010.

[HKLB11] Fenglin Han, Surya Bahadur Kathayat, Hien Nam Le, and Rolv Bræk. Towards choreography model transformation by graph transformation. In *Proc. of 2nd IEEE International Conference on Software Engineering and Service Sciences*, Beijing, China, 2011.

[HLVFAP+06] D. Hernández-Leo, E.D. Villasclaras-Fernández, J.I. Asensio-Pérez, Y. Dimitriadis, I.M. Jorrín-Abellán, I. Ruiz-Requies, and B. Rubia-Avi. COLLAGE: A collaborative learning design editor based on patterns. *Educational Technology & Society*, 9(1):58–71, 2006.

[HMTK04] H. Hummel, J. Manderveld, C. Tattersall, and R. Koper. Educational modelling language and learning design; new opportunities for instructional reusability and personalised learning. *Int. J. Learn. Technol.*, 1:111–126, November 2004.

[IMSa] IMS. IMS Learning Design Version 1.0 Final Specification. http://www.imsglobal.org/learningdesign/.

[IMSb] IMS. IMS Learning Design Best Practice and Implementation Guide. http://www.imsglobal.org/learningdesign/.

[KB09] Surya Bahadur Kathayat and Rolv Bræk. Platform support for situated collaborative learning. In *International Conference on Mobile, Hybrid, and On-line Learning, 2009. ELML'09*. IEEE Press, 2009.

[KB10] Surya Bahadur Kathayat and Rolv Bræk. From flow-global choreography to component types. In *System Analysis and Modeling (SAM)*, volume 6598 of *Lecture Notes in Computer Science*. Springer, 2010.

[KKB11] Frank Alexander Kraemer, Surya Bahadur Kathayat, and Rolv Bræk. Unified modeling of service logic with user interfaces. *International Journal of Cooperative Information Systems (IJCIS)*, 20 (2):177–200, 2011.

[Kra08] Frank Alexander Kraemer. *Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks*. PhD thesis, Norwegian University of Science and Technology, 2008.

[KSH07] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann. Engineering support for uml activities by automated model-checking — an example. In *Proceedings of the 4th International Workshop on Rapid Integration of Software Engineering Techniques (RISE)*, November 2007.

[KSH09] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann. Tool support for the rapid composition, analysis and implementation of reactive services. *Journal of Systems and Software*, 82(12):2068–2080, 2009.

[lam] LAMS Project Website, http://lamsfoundation.org/. Accessed on February 2011.

[LPD04] D. H. Leo, J. I. Perez, Asensio, and Y. A. Dimitriadis. Ims learning design support for the formalization of collaborative learning patterns. In *Proceedings of the IEEE International Conference on Advanced Learning Technologies*, pages 350–354, Washington, DC, USA, 2004. IEEE Computer Society.

[MHHH05] Y. Miao, K. Hoeksema, H. U. Hoppe, and A. Harrer. Cscl scripts: modelling features and potential use. In *Proceedings of the 2005 conference on Computer support for collaborative learning: learning 2005: the next 10 years!*, pages 423–432. International Society of the Learning Sciences, 2005.

[Mia05] Y. Miao. Cosmos: Facilitating learning designers to author units of learning using IMS LD. In *Proceeding of the 2005 conference on Towards Sustainable and Scalable Educational Innovations Informed by the Learning Sciences: Sharing Good Practices of Research, Experimentation and Innovation*, pages 275–282, Amsterdam, The Netherlands, 2005.

[mot] MOT+ Editor Website, http://www.licef.teluq.uqam.ca/. Accessed on January 2011.

[OMG09] OMG. Uml 2.2 superstructure specification, February 2009.

[rel] RELOAD Project Website, http://www.reload.ac.uk/. Accessed on February 2011.

[RRN04] Manuel Caeiro Rodriguez, Luis Anido Rifon, and Martin Llamas Nistal. Towards ims-ld extensions to actually support heterogeneous learning designs. a pattern-based approach. In *Proceedings of the IEEE International Conference on Advanced Learning Technologies*, pages 565–569, Washington, DC, USA, 2004. IEEE Computer Society.

[YC07] D. Yu and X.M. Chen. Creating computer supported collaborative learning activities with IMS LD. In Julie Jacko, editor, *Human-Computer Interaction. HCI Applications and Services*, volume 4553 of *Lecture Notes in Computer Science*, pages 391–400. Springer Berlin / Heidelberg, 2007.

Chapter 11

Paper 6

Title: A Model-Driven Framework for Component-based Development.

In: 15th International Conference on System Design Languages, July 5th - 7th, 2011. Toulouse, France

Publisher: Springer (LNCS 7083).

A Model-Driven Framework for Component-based Development

Surya Bahadur Kathayat, Hien Nam Le, and Rolv Bræk
Norwegian University of Science and Technology (NTNU)
Department of Telematics
Trondheim, Norway
Email:{surya, hiennam, rolv.braek}@item.ntnu.no

Abstract

In this paper, we present a Model-Driven framework to support development of components. The framework addresses the following important issues: (1) how to reduce the cost of making component reusable, (2) how to efficiently ensure compatibility among components in a composition, (3) how to relate service composition to system composition The framework has three main models: (1) Service Models specifying the global cross-cutting functionalities; (2) Component Models, can be synthesized from service models or developed by composing existing components together, specifying the interfaces and behaviors of both elementary and composite component; and (3) System Design for developing system by composing components.

keywords:

11.1 Introduction

Traditionally component-based development is considered as an approach for software development in which software solutions are developed by assembling software parts defined as reusable components such that the desired system behavior is achieved through the collaboration of those parts [Szy02, CCL06, Lau06]. However, there are many different definitions of what a component is depending on the context in which a component performs. For example, [Szy02] defines a software component as a unit of composition with contractually specified interfaces and explicit context dependencies that can be deployed independently and be composed by third parties. On the other hand, in [CLM05], a component is defined as a run-time software component which is a dynamically bind-able package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at run-time. These definitions focus on the development or the execution of individual component without detailing how components are combined with others [CCL06]. In general, software components can be defined as reusable and composable software units which can be composed together with other software components to form new, more composite structures [Szy02, Lau06].

Component-based development, on one hand has potential to reduce cost and time to market. On the other hand, there is additional cost associated with making components reusable that must be justified by cost reductions in later re-use. Central issues are the granularity of components, how to efficiently define and specify component interfaces so that components can be connected and deployed together, and how to specify the behaviors of composite components [CCL06, Jis04].

In this paper, we present a service-oriented development approach leading to components with behaviored interface contracts derived directly from service models, at little or no extra additional costs. The approach helps to overcome some serious problems that have plagued component oriented development by:

- Reducing the cost of developing component interfaces for reuse.
- Improving the quality of interfaces by including a precise description of interface behaviors.
- Defining interfaces as reusable entities (contracts) that can be validated separately.
- Reducing compatibility checks on links between components to static contract checks.
- Supporting a granularity of components ranging from a few actions to complex sub-systems.
- Supporting composition of distributed components as well as local components.

Our approach is based on collaborative services [KB10]. We use the term *service* as a partial system functionality where two or more entities, i.e., components, collaborate and achieve some goal [BCLR10]. A component may participate in several services in different contexts. A service can involve roles played by different components. Thus there is a need for composition both in the service dimension and in the system dimension. Our development approach leads to components with interfaces for composition in both dimensions.

The rest of the paper is as following. In Section 11.2, we present our Model-Driven framework for component development which is illustrated by a Train Control System (ERTMS) [ert]. Section 11.3 discusses the composition of component. How a system is developed by composing components is discussed in Section 11.4. Section 11.5 discusses the related work. Conclusions are is given in Section 11.6.

11.2 Model-Driven Framework for Component Development

In this section, we give an overview of our collaboration-based Model-Driven framework that supports the developments of components. The framework is shown in Figure 11.1 and has three main models: (1) Service Models for specifying the structure and behavior of cross-cutting services and interfaces; (2) Component Models for specifying the interfaces and local behaviors of both elementary and composite component; and (3) System Design for developing system by composing components. The details of the framework are described below and illustrated by a case study called a Train Control System - based on the new European Rail Traffic Management System (ERTMS) [KLB10, ert].

11.2.1 Service models

We consider services as partial system functionality where two or more entities collaborate and achieve some goals. We distinguish between elementary services and composite services. Elementary services can not be decomposed further while composite service is composed from smaller services. We use UML 2.x collaborations to specify the structure of collaborating entities in a service and activity diagram to specify their behavior. Figure 11.2 defines the structure of the Train Control System (ERTMS) using UML 2.x collaborations.

While moving in a physical geographical region, a Train must always be supervised by a Radio Block Center (RBC) whose responsibility is to monitor and control all train movements within its region. The Train Control System is decomposed into three collaboration uses that refer to separately defined collaborations each having an associated activity diagram:

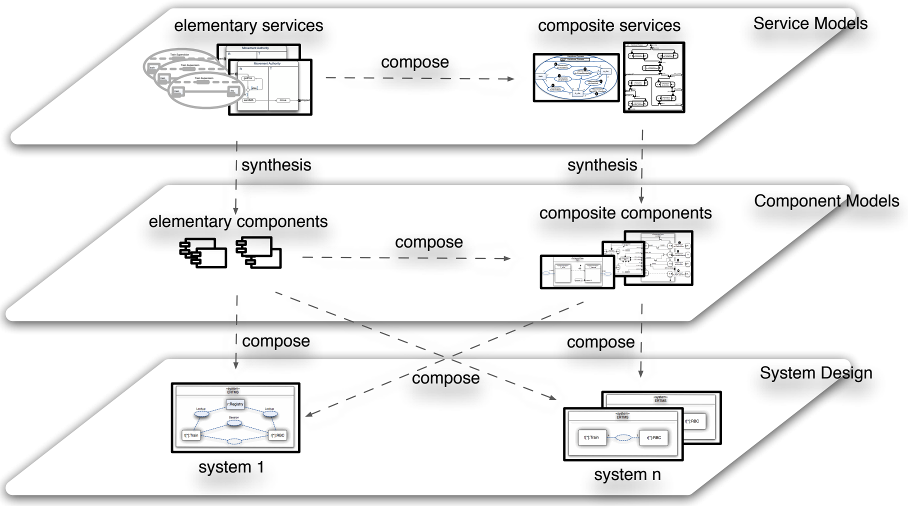


Figure 11.1: The Collaboration Based Framework

- *Movement supervision:* this composite collaboration monitors the location of the Train via the *PositionReport* service, and controls the safe traveling distance of the Train via the *MoveAuthority* service.

- *Communication Supervision:* this collaboration monitors and manages the communication between the Train and the RBC. The *Communication Supervision* is a composite service which is composed from collaboration uses referring to smaller collaborations that are defined separately, i.e., the elementary services *EstSess* and *TermSess*, which represents the establishment and termination of communication sessions between a Train and a RBC, respectively. The roles of the collaboration uses are bound to the roles of the enclosing collaborations. For example the roles *train*, and *rbc* of the *EstSess* collaboration are bound to the role cT and cR of the *Communication Supervision* collaboration uses. Note that the numeric identifiers associated with collaborations uses are used just for ease of referencing within this paper.

- *Handover supervision:* during its journey, the Train may travel across more than one geographical region; this means the supervision of the Train movement is handled by more than one RBC. The handover is performed by the collaborative activity *HandoverSupervision*. When the handover supervision is completed, the Train will switch back

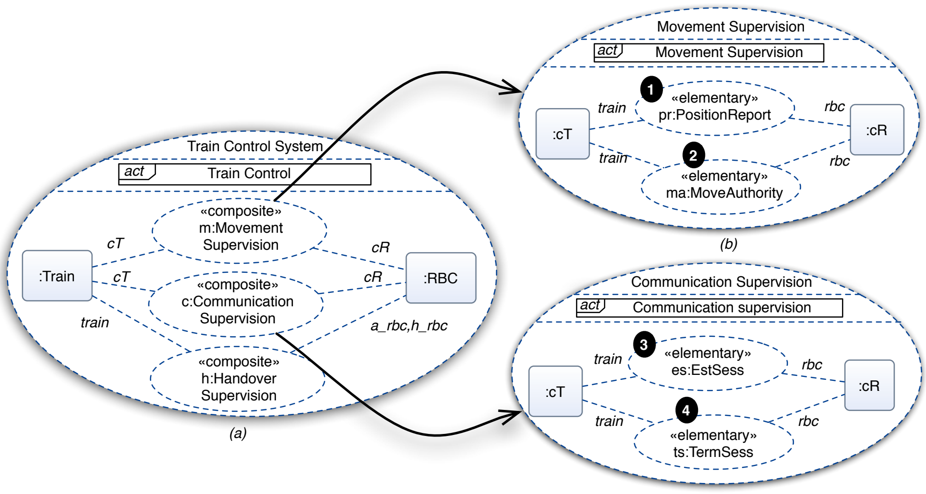


Figure 11.2: Structural Model of Train Control System

to normal movement supervision (i.e., the Train will now be under supervised by the new RBC). During the handover process, the current RBC plays a handing over role (*h_rbc*) and the neighbor RBC where the train is approaching plays accepting role (*a_rbc*).

The collaboration structure only describe a structure of the roles of components, e.g., *train* and *rbc*; but not how these interact with each other in a system. The interaction behavior of elementary service roles can be specified using the token flow semantics of UML activity diagrams, for example using the swim-lane notation of [KKB11]. In this paper, we assume that elementary services are available in the repository as building blocks. They can be reused in the behavior specification (called *choreography models*) of a composite service.

For composite services, the choreography is very useful for discussion with end-users and to capture requirements in a very abstract way and to specify the intended global behavior [KLB10, KB10]. The choreography of a collaboration is specified using UML activity diagrams, illustrated in Figure 11.3 for the collaborations represented in Figure 11.2. One can see that the choreography of a composite service specifies an ordering of the behaviors defined by the collaborations referenced by the collaboration uses. Note that sub-collaborations in *Handover Supervision* in Figure 11.3(c) are composite collaborations with choreography defined in Figure 11.4. The choreography models specify the following aspects:

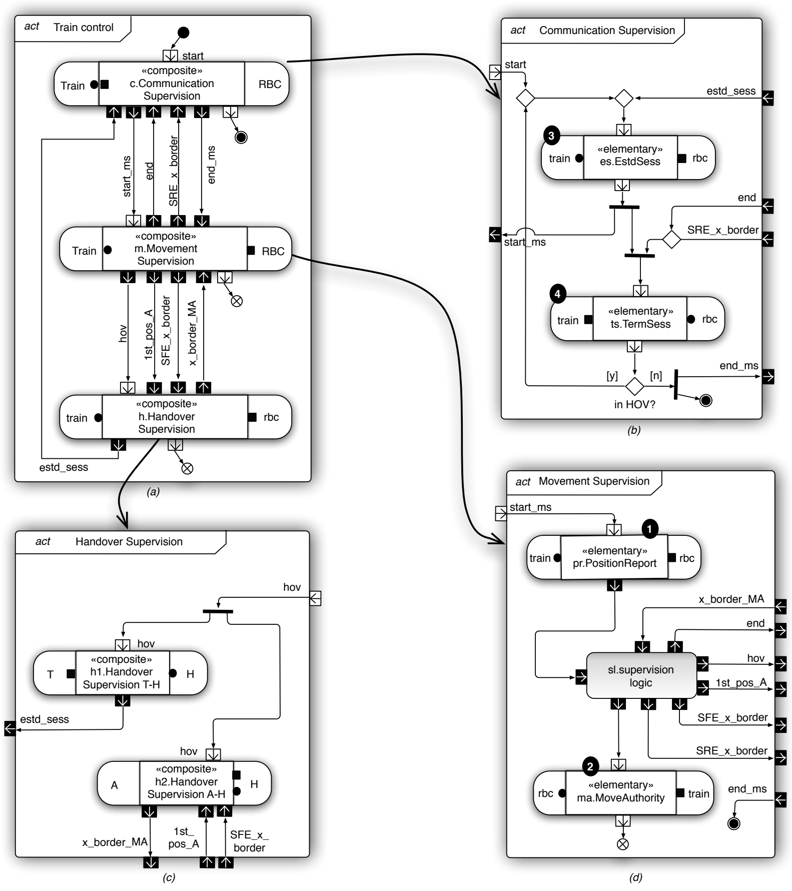


Figure 11.3: Choreography of the Train Control System

- *Participating roles.* The participating roles are represented as partitions of collaborative actions.

- *Initiation and termination.* The initiating and terminating roles in a collaboration are indicated by black filled circle and square boxes respectively. Note that this notation is not the part of standard UML, but needed for analysis and may be provided by additional profiling. For example, in Figure 11.3(a), the Train will initiate and terminate the *Communication Supervision* collaboration with the RBC, while in the *Movement Supervision* collaboration, the initiating role is the Train and the terminating role is the RBC.

- *Execution order of collaborations.* As shown in Figure 11.3(a), the

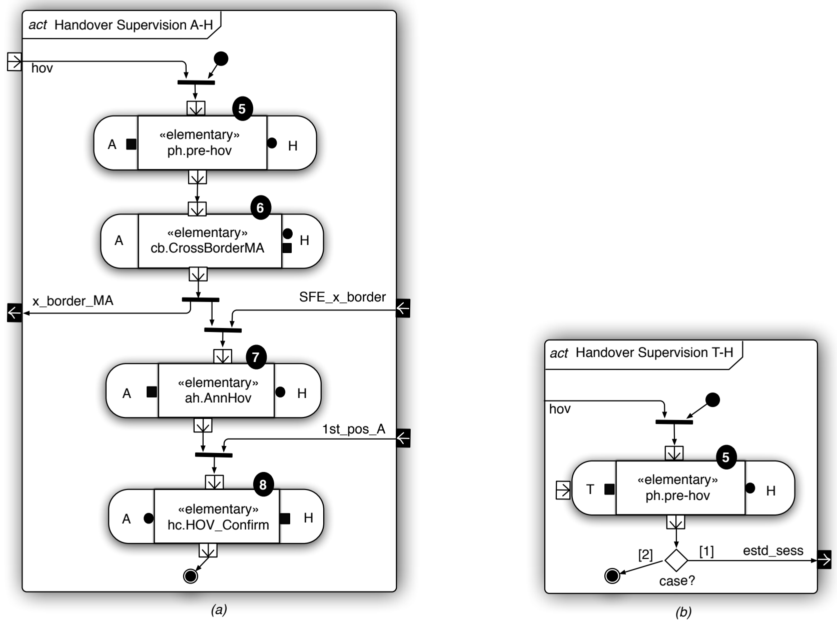


Figure 11.4: Choreography of the Handover Supervision Collaboration in the Train Control System

Movement Supervision is only started when the communication between the train and the RBC has been established and a token is emitted on the *start_ms* pin.

- *Collaboration interactions.* The choreography model also has the capacity to specify how collaboration activities interact while they are active by means of streaming pins. For example when the Train reaches the region border, the movement supervision will issue a token via the *end* streaming output pin to the communication supervision to terminate the communication session.

- *Local actions.* As shown in Figure 11.3(d), the interaction between collaborations *PositionReport* and *MoveAuthority* can also be affected by some local activity *supervisionlogic* performed by the *rbc*.

The choreography model does not explicitly specify the behavior of individual components, for example, the *train* or *rbc*. However, based on this level of specification, the behavior of components as well as their interfaces can be synthesized using a projection technique as explained in

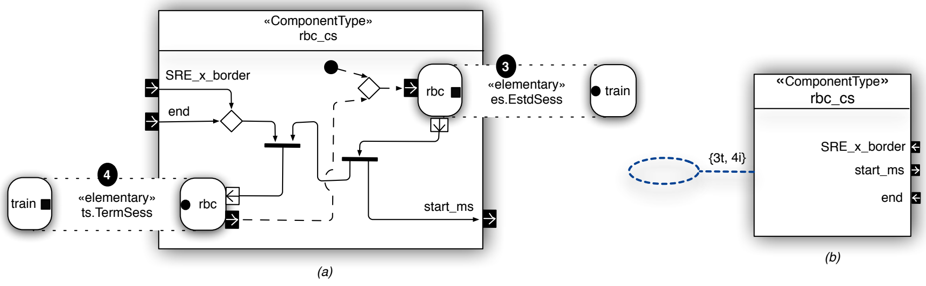


Figure 11.5: The RBC Component in Communication Supervision Service

[KLB10, KB10, BCLR10]. Elementary services may result in elementary components while composite service models result in composite components. The next section explains the resulting component models.

11.2.2 Component models

In this section, the component model is presented with focus on defining the component interfaces and the internal behaviors of the components.

Figure 11.5 shows the communication supervision component *rbc_cs* of the RBC derived from the choreography. Another component the *train_cs* is shown in Figure 11.6. These component models have the following properties:

- *Interfaces:* Components have two types of interfaces: (1) semantic interfaces for inter-component interaction and, (2) local interfaces pins for intra-component interactions. We use the term semantic interface differently from [BF04, JFS08] where semantic interfaces have been defined using state-machines defining the observable behavior at the service interface. A novelty introduced here is to directly use the collaboration and activity diagram to define the semantic interfaces both statically and dynamically. In this way, the semantic interfaces become reusable building blocks with pins for local composition. For example, in Figure 11.5(a), *rbc_cs* component has two semantic interfaces defined by the collaborations *EstSess* and *TermSess* responsible for establishing and terminating communication session with the Train. While monitoring the communication session, the *rbc_cs* component may interact with other local components using the streaming pins. The *rbc_cs* component has two input streaming pins *SRE_x_border* and *end*; and one output pin *start_ms*.

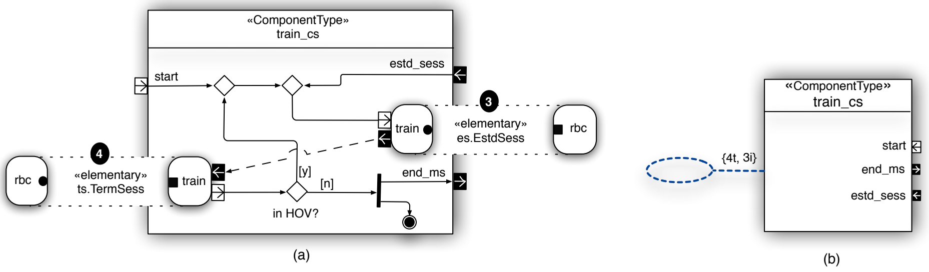


Figure 11.6: The Train Component in the Communication Supervision Service

- *Internal behavior:* The internal behavior of a component is represented by the flows which link actions performed by the component. There are two different types of flows: initiating flows and responding flows. Initiating flows in Figure 11.5(a) are directly projected from the choreography models shown in Figure 11.3(b) using techniques presented in [KLB10, KB10]. The responding flows capture the global flows that are external to the component (c.f [KLB10] for details). Responding flows in a component specify when a particular role should be ready to respond to the external initiating role in a particular collaboration. Responding flows are represented by dashed lines. In Figure 11.5(a), there is a responding flow from the service role *rbc* in *ts.TermSess* to the role *rbc* in *es.EstdSess* to represent that when the *rbc_cs* component is finished playing its role in *ts.TermSess* service, it should be ready to respond in the *es.EstdSess* service in order to be ready to handle the handover activity.

Such component models can be represented externally as black boxes by hiding the internal details and showing just the information about interfaces as in Figure 11.5(b) and 11.6(b). The name and type of the pins corresponding to local interfaces is retained. The semantic interfaces are abstractly represented by a set $\{ns\}$ where n is a reference [1] to an interface and s is a character having values of (1) i representing that component plays initiating role; (2) t representing that component plays a terminating role; and (3) p representing that the component plays a participating role in a collaboration.

Semantic Interfaces of a component define virtual sockets, plugs and cables for wiring components together. As long as the plugs and sockets

[1]In this paper we simply use numeric references.

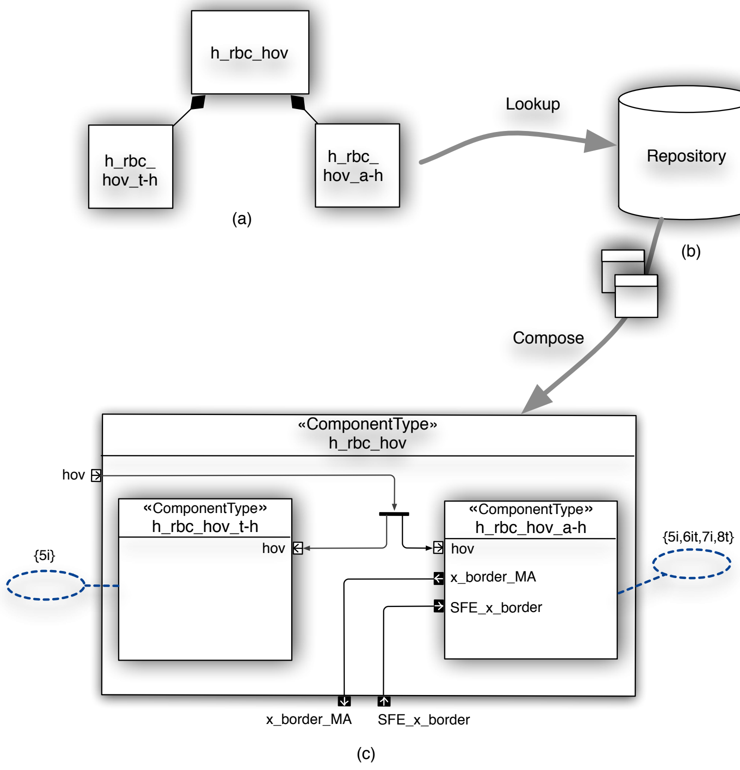


Figure 11.7: Handingover RBC Component

of a cable are compatible with the plugs and sockets of the components it links, we can rest assured that the wiring is valid.

11.3 Component Composition

This section discusses how to define a composite components locally. For illustration purposes, we select a RBC component from the Train Control System [ert]. We assume that the inner parts of a composite component are in a repository and can be connected together via pins (as local interfaces) at their boundary. A component composition involves the following steps:

- *Specify the roles to be provided.* For example, the roles to be provided by the handing over RBC component *h_rbc_hov* in the Train Control System can be specified as in Figure 11.7(a) to consist of two elementary roles representing handover interactions with the Train (*h_rbc_hov_t-h*) and the accepting RBC (*h_rbc_hov_a-h*).

- *Find components providing the roles* in a component library. This can be done manually by the composer based on the requirements at hand or recommended/selected based on the information available in the service choreography models from which the roles (or components) have been synthesized. Selected components to provide the roles for handing over RBC (*h_rbc_hov*) are shown in Figure 11.7(b).

- *Compose the selected components together* using the pins at the boundary of the selected components. The composition can be done manually by the composer or based on the information available in the service choreography. Pins may also be connected based on the data types that they use. Pins that are not connected to any other pins are made transparent to the boundary of a composite component. For example composition of inner components in *h_rbc_hov* is shown in Figure 11.7(c). This is done according to the choreography model of the handover supervision service in Figure 11.3(c). Note that pins *x_border_MA* and *SFE_x_border* are made available to the component (*h_rbc_hov*) boundary as local interfaces.

In similar way as the *h_rbc_hov*, a RBC component can be composed as shown in Figure 11.8. The role structure is shown in Figure 11.8(a). Note that inner role of a component may itself be composite i.e. may be composed from other inner-inner roles. For example the *h_rbc_hov* and *a_rbc_hov* roles of the handing over RBC component are composed together to make a composite handing over RBC role (*rbc_hov*). Finally the *rbc_hov*, *rbc_cs* and *rbc_ms* are composed together to make a composite RBC component, as shown in Figure 11.8(c).

In a similar way, the other components of the Train in a Train Control System can be composed. Note that the composed components can be placed back into the repository so that they can be reused in other compositions.

11.4 System Design

This is the last composition step where system level components (i.e. not the inner roles of a component) are composed together in order to make a system. In this step, components are composed together mainly using semantic interfaces.

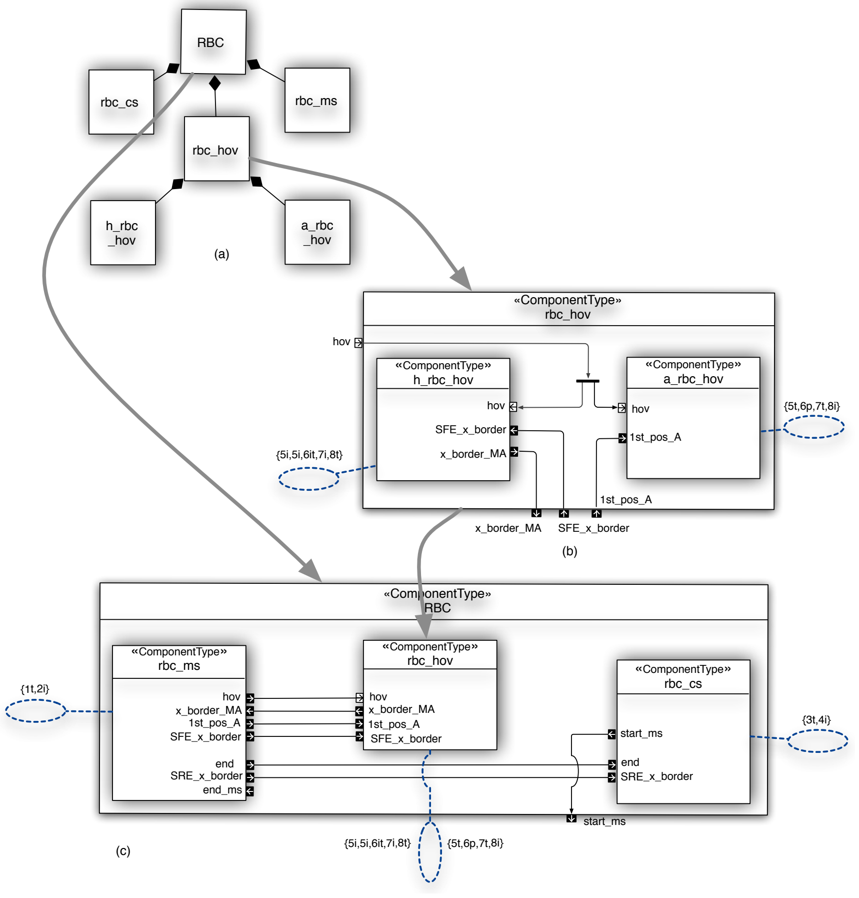


Figure 11.8: Composition of a RBC Component

Figure 11.9 shows a system design model for the Train Control System. Note that the Train and RBC component types interact via collaborations (shown by dotted eclipse symbol with associated semantic interfaces) and pins (shown by solid line connecting local interface pins). In the cases where collaborating components have pins at the interfaces, such pins will be replaced by send and receive events. For example in Figure 11.9, there will be a *start-ms* sending event at the RBC component and corresponding receive event at the Train component.

Compatibility among components across semantic interfaces are ensured by the following steps:

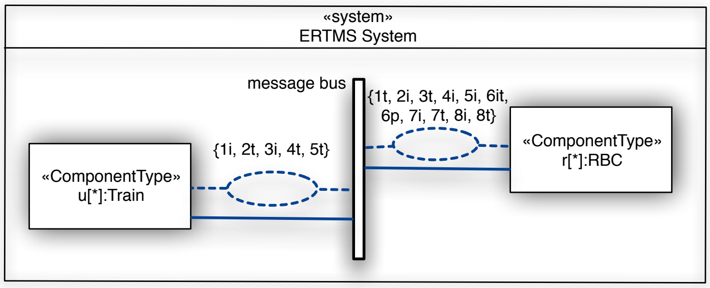


Figure 11.9: System Design Model for Train Control System

- Each semantic interface is analyzed separately for internal consistency.
- Each component is analyzed separately to make sure it is consistent with all its semantic interfaces.
- When two components are linked we check that the linked roles are complementing roles of the same semantic interface; this assures that the link is consistent, but not that the ordering of roles using different links are compatible.
- We ensure compatibility in the ordering of roles by considering either the local ordering within each component against a global choreography, or by considering the responding flows of a component against the initiating flows of linked components.

The local interfaces are currently represented just as pins, and thus as a kind of static interface. The concept of external state machine, *esm*, used in the Arctis tool [KSH09] may be used to define the external behavior across local interfaces, but this has not be elaborated here.

In a system, there may be many instances of the same service running concurrently with roles assigned to different system components. Each system component may play roles in different services, and different components may be able to play different combinations of the roles. This variability is provided through the component and system composition steps.

11.5 Related Works

A large body of work on component based development is based on the approach that first components (and their internal details) are designed and then static interfaces are defined. For example IDL based interfaces in CORBA and WSDL based interfaces in web-services. This means that additional work is needed to make interfaces to the components. In our approach, we start from the collaborative service models and derive components with external interface contracts. Since we produce interface contracts and components from the same source, reusability comes almost for free. Moreover the interface contracts have behavior that can be validated once and for all without considering particular components. Given the validated interfaces, checking the consistency of links among components is a matter of simple static matching.

There are many related works focusing on the issues of component composition [Lau06, AIJQ06, PKBGS08]. In these approach, existing components are usually assumed to be compatible and connectable as long as their static interfaces match. There is however no standard way of defining semantic interfaces. For example, in [Lau06], composite components are created by connecting existing components via a connector. In their approach, existing components, which are assumed to be compatible, are connected together either in sequential or parallel patterns. In our approach, we provide two types of interfaces to a component: local interfaces and semantic interfaces. The semantic interfaces define a behavior contract among collaborating components by encapsulating the collaborative behavior.

In our approach, components (along with their semantic interfaces) may be automatically synthesized from collaboration or service models [KLB10, KB10, LK11]. External dependencies to a component are maintained by a special kind of flows called *responding flows* [KB10]. The responding flows are also useful to detect any problems with emergent behavior, also called realizability problems, as briefly described in [KB10]. More details on the realizability of choreographies can be found in [CBvB07]. Another body of related work [BGG$^+$06, QZCY07, SB09] deals with the relation between choreographies and their realization in terms of distributed components, so called *orchestrations*. Several of these use a similar projection technique as we do, but without mapping external flows to responding flows within the components. The idea to keep collaborations as interface contracts in the derived components seems to be original in our work.

Moreover, new composite components can be designed by composing existing components from a repository. A system can be designed in a similar approach i.e. by composing components together. Components can also be dynamically linked at run-time with efficient runtime checks on the compatibility in terms of semantic interfaces.

11.6 Concluding Remarks

In this paper, we have presented and discussed a Model-Driven framework, which is based on the concept of collaborative services, to support the development and composition of components. We use UML collaborations to model the structure of collaborating entities in a service. Global behavior, called choreography, is specified using UML activity diagrams. From collaborations with choreography we get semantic interfaces that are building blocks and can be put into a library for reuse. We are also able to synthesize components equipped with semantic interfaces ready to be put into a repository. Given a library of semantic interfaces and components one can design a new component and eventually a system, according to the requirements at hand, by composing existing components together. Two types of interfaces, semantic interface and local interfaces, are used to compose the components together.

References

[AIJQ06] J. P. A. Almeida, Maria-Eugenia Iacob, Henk Jonkers, and Dick Quartel. Model-driven development of context-aware services. In *Distributed Applications and Interoperable Systems*, volume 4025/2006 of *Lecture Notes in Computer Science*, pages 213–227, Berlin, 2006. Springer Verlag.

[BCLR10] Rolv Bræk, Humberto Nicolás Castejón, Hien Nam Le, and Judith E. Y. Rossebø. Policy-based service composition and recommendation. In *Service Intelligence and Service Science: Evolutionary Technologies and Challenges*. IGI Global, 2010.

[BF04] R. Bræk and J. Floch. Ict convergence: Modeling issues. In *In System Analysis and Modeling (SAM), 4th International SDL and MSC Workshop, pages 237–256, Ottawa, Canada.*, 2004.

[BGG+06] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration conformance for system design. In *COORDINATION, volume 4038 of LNCS*, pages 63–81. Springer, 2006.

[CBvB07] Humberto Nicolás Castejón, Rolv Bræk, and Gregor von Bochmann. Realizability of collaboration-based service specifications. In *Proceedings of the 14th Asia-Pacific Software Engineering Conference*, pages 73–80, Washington, DC, USA, 2007. IEEE Computer Society.

[CCL06] Ivica Crnkovic, Michel Chaudron, and Stig Larsson. Component-based development process and component lifecycle. In *Proceedings of the International Conference on Software Engineering Advances*, Washington, DC, USA, 2006. IEEE Computer Society.

[CLM05] Sergio De Cesare, Mark Lycett, and Robert D. Macredie. *Development Of Component-based Information Systems (Advances in Management Information Systems).* M. E. Sharpe, Inc., Armonk, NY, USA, 2005.

[ert] FIS for the RBC/RBC Handover. http://www.era.europa.eu/Document-Register/Documents/SUBSET-039v2.3.0.pdf, Accessed November 2010.

[JFS08] S. Jiang, J. Floch, and R. Sanders. Modeling and validating service choreography with semantic interfaces and goals. In *IEEE International Symposium on Service-Oriented System Engineering, 2008*, pages 73 –78. IEEE Computer Society, 2008.

[Jis04] Dan Laurentiu Jisa. Component based development methods: comparison. In *Proceedings of the 5th international conference on Computer systems and technologies*, pages 1–6, New York, NY, USA, 2004. ACM.

[KB10] Surya Bahadur Kathayat and Rolv Bræk. From flow-global choreography to component types. In *System Analysis and Modeling (SAM)*, volume 6598 of *Lecture Notes in Computer Science*. Springer, 2010.

[KKB11] Frank Alexander Kraemer, Surya Bahadur Kathayat, and Rolv Bræk. Unified modeling of service logic with user interfaces. *International Journal of Cooperative Information Systems (IJCIS)*, 20 (2):177–200, 2011.

[KLB10] Surya Bahadur Kathayat, Hien Nam Le, and Rolv Bræk. Automatic derivation of components from choreographies - a case study. In *International conference on Software Engineering*, Phuket, Thailand, 2010.

[KSH09] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann. Tool support for the rapid composition, analysis and implementation of reactive services. *Journal of Systems and Software*, 82(12):2068–2080, 2009.

[Lau06] Kung-Kiu Lau. Software component models. In *Proceedings of the 28th international conference on Software engineering*, ICSE ’06, pages 1081–1082, New York, NY, USA, 2006. ACM.

[LK11] Hien Nam Le and Surya Bahadur Kathayat. A framework to support the development of collaborative components. In *Proceedings of the 9th Workshop on System/Software Architecture*. Springer, LNBIP, 2011.

[PKBGS08] An Phung-Khac, Antoine Beugnard, Jean-Marie Gilliot, and Maria-Teresa Segarra. Model-driven development of component-based adaptive distributed applications. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC'08, pages 2186–2191, New York, NY, USA, 2008. ACM.

[QZCY07] Zongyan Qiu, Xiangpeng Zhao, Chao Cai, and Hongli Yang. Towards the theoretical foundation of choreography. In *Proceedings of the 16th international conference on World Wide Web*, pages 973–982. ACM, 2007.

[SB09] Gwen Salaün and Tevfik Bultan. Realizability of choreographies using process algebra encodings. In *Proceedings of the 7th International Conference on Integrated Formal Methods*, IFM '09, pages 167–182, Berlin, Heidelberg, 2009. Springer-Verlag.

[Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

Chapter 12

Paper 7

Title: Towards Choreography Model Transformation via Graph Transformation.

In: The 2nd IEEE International Conference on Software Engineering and Service Sciences (ICSESS 2011). July 15 17, 2011. Beijing, China

Publisher: IEEE.

Towards Choreography Model Transformation via Graph Transformation

Fenglin Han, **Surya Bahadur Kathayat**, Hien Nam Le, Rolv Bræk, and Peter Herrmann
Norwegian University of Science and Technology (NTNU)
Department of Telematics
Trondheim, Norway
Email:{simon.han, surya, hiennam, rolv.braek, herrmann}@item.ntnu.no

Abstract

This paper presents a Model-Driven method to develop collaborative systems. We use UML collaborations to capture the system requirements and architecture of collaborative system. The system behaviors are specified by two choreography models: an abstract flow-global and a detailed flow- localized choreography. These choreography models are both described by UML activity diagrams. A graph-based transformation approach for carrying out the transformation from the flow-global to the flow-local choreography is the core contribution of this paper. Our approach is illustrated using a case study of the European Rail Traffic Management System.

keywords: Choreography Model; Model Transformation; Graph Transformation; Collaborative Service.

12.1 Introduction

Model-Driven Development (MDD) is an approach supporting the software development process by creating models on different levels of abstraction and platform independence. First, one develops more abstract models specifying the pure functionality of a particular solution or an application domain but hiding aspects of the later realization. These models can be transformed into models incorporating more implementation details. Based on the refined models, application code can be generated ranging from system skeletons to complete, deployable products for different platforms. MDD is

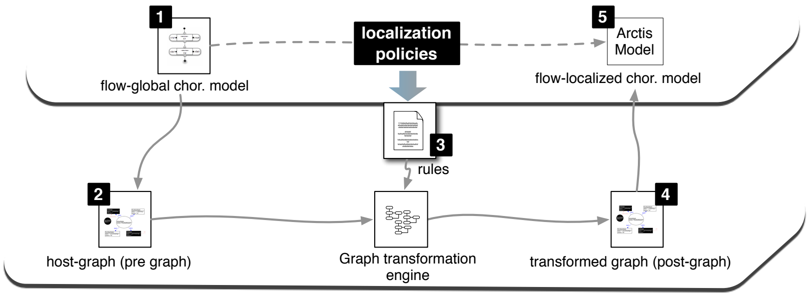


Figure 12.1: Overall Approach

considered effective when the transformation from the abstract to the detailed models can be done with a high degree of automation. This makes it easy to keep consistency between the two models. In addition, the developer can utilize the comprehensibility and the generality of the platform-independent more abstract model as well as the fine-grained semantics and the mature structuring mechanisms of the more detailed one.

In this paper we discuss a model-driven development method for distributed, reactive and collaborative services. A collaborative service is defined as a partial system functionality in which two or more components collaborate to achieve a common goal [KSH09, CBvB07]. UML 2.x collaborations are used to describe the structure of collaborating participants, while UML activities specify the collaborative behavior of interacting participants. We consider the behavior models (called choreography models) at two levels of details and use graph-based model transformation to derive detailed implementable models from global abstract models.

There are other model transformation techniques that connect different abstract levels of models. However, either the two levels of models are not associated effectively or the more abstract model is not giving enough information for the more detailed model which makes the model transformation not a necessarity. Further, Model transformation also needs to be flexible, extensible and as understandable for composers as possible. We believe that we fulfill the above requirements of models and model transformation. Figure 12.1 delineates the overall MDD approach:

- A flow-global choreography model (label 1 in Figure 12.1) abstracts from most technical issues of a distributed system seeking to specify the desired global behavior in terms as close to the problem domain as possible. It is intended to be understandable by end-users and experts of a specific domain. Thus, the flow-global choreography focuses on the global interaction, i.e., avoiding the details and resolutions of coordination problems that may occur at the level of a distributed realization.

- A flow-localized choreography specification (label 5 in Figure 12.1) is used to define global behavior in sufficient detail to allow extensive analysis and to automatically synthesize the behavior of isolated system parts and to generate the application code. The flow-global choreography can be transformed to the flow-localized choreography by using a set of transformation rules and policies (see label 3 in Figure 12.1). From flow-localized choreography models, implementation code can be generated using existing tools like Arctis [KSH09].

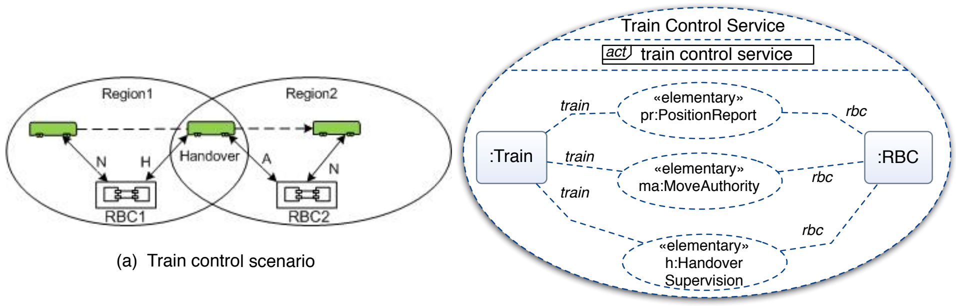
- The focus of this paper is on the model transformation from a flow-global to a flow-localized choreography using graph transformation techniques. The localization policies are implemented by applying graph transformation rules (label 3 in Figure 12.1) to the host-graph model of a flow-global choreography model (label 2) which is then converted to a post-graph model of a flow-localized choreography specification (label 4).

There are several advantages of using graph transformation to support the choreography model transformation. First, the flow-global models, which can be re-used in different system scenarios, are generally designed and stored in a repository [KB10]. Thus, the associated graph models of the flow-global choreography models are also pre-defined and can be re-used. Second, it is not necessary that the full flow-global model and the flow-global graph model must be available in order to be transformed to flow-localized model. This is due to the possibility to transform partial flow-global choreography models.

In the following, we introduce the structure and choreography models by a train control system scenario in Section 12.2. A survey of the task for the model transformation is provided in Section 12.3. Model transformation using the graph transformation approach is presented in Section 12.4. Section 12.5 discusses the related work and is followed by concluding remarks in Section 12.6.

12.2 Architecture and Choreography Models

In this section, we discuss the related models contributing to our MDD approach: *1)* The architecture model to capture the system requirements


Region1
Region2
Handover
N
H
A
N
RBC1
RBC2
Train Control Service
act train control service
:Train
:RBC
train
train
train
rbc
rbc
rbc
«elementary» pr:PositionReport
«elementary» ma:MoveAuthority
«elementary» h:Handover Supervision

(a) Train control scenario

(b) Structural model

Figure 12.2: Structure Model of the Train Control System.

and to structure the system architecture; *2)* the flow-global choreography model to specify the high-level global behavior including the ordering and causality among sub-services and service roles in a composite service; and *3)* the flow-localized choreography model to specify detailed behavior so that application code can be generated. We present a summary of these models using an example of the European Rail Traffic Management System (ERTMS).

12.2.1 Architecture model

The UML collaboration composite structure are used to specify the structure (i.e., roles and interactions) of distributed collaborative entities and the architecture of the system. Figure 12.2 (a) shows an example of a train control scenario which is described as following:

A train must always be supervised by a radio block center (*RBC*). The *RBC*'s responsibility is to monitor and control all train movements in a particular region. Guided by its current *RBC*, the train keeps on moving and sends its position reports to the *RBC* and the *RBC* validates the received position information of the train. Moreover, the *RBC* issues successive movement authorities (*MA*) to the train, which specifies a safe distance that the train can travel. In addition, in some situations, the train may also travel across several regions covered by different *RBC*s. This means the supervision of the train movement can be handled by more than one *RBC*. When the train crosses a border between two regions, the *RBC* of the current region will hand over the control to another one, i.e., Handover process.

The train control scenario descriptions and requirements are captured and presented by our structural model as shown in Figure 12.2 (b) which is

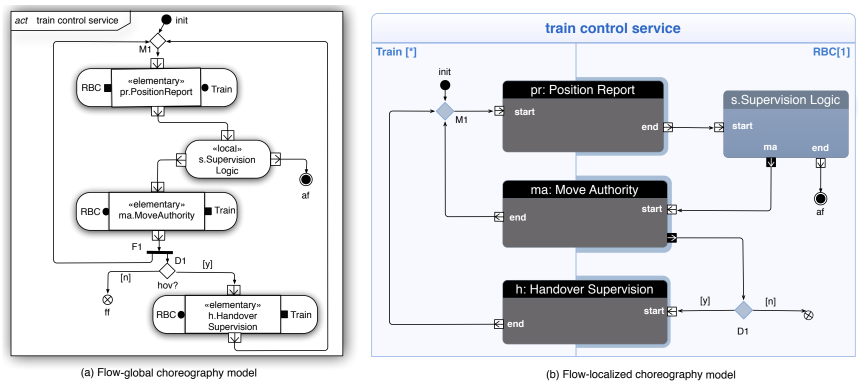

act train control service
init
M1
RBC
«elementary» pr.PositionReport
Train
«local» s.Supervision Logic
af
«elementary» ma.MoveAuthority
F1
D1
[y]
[n]
hov?
ff
«elementary» h.Handover Supervision
(a) Flow-global choreography model
train control service
Train [*]
RBC[1]
pr: Position Report
start
end
s.Supervision Logic
ma
ma: Move Authority
h: Handover Supervision
(b) Flow-localized choreography model

Figure 12.3: Flow-Global model and Flow-Localized model(derived as Arctis model)

a UML collaboration structure.

The *TrainControlService* system has two main parts *Train* and *RBC* which take part in different collaborative sub-activities. In Figure 12.2(b), the *Train* and *RBC* participate in three collaborative sub-activities: *PositionReport*, *MoveAuthority*, and *HandoverSupervision* which take responsibility for the activities: (1) *PositionReport* reports the current train position to the RBC; (2) *MoveAuthority* sends the safe travel distance from the RBC to the train; and (3) *HandoverSupervision* transfers the train control supervision to the new RBC if the train travels to a different region.

12.2.2 Choreography models

The choreography specification of a train control service is given in Figure 12.3.

12.2.2.1 Flow-global choreography model

The flow-global specification shown in Figure 12.3 (a) is defined by UML activities connecting actions (in particular, call behavior actions) by flows that are not assigned to any particular role in the collaboration. Actions may either specify the behavior of a collaboration or a local activity. Flows may contain intermediate control nodes (e.g. start, stop, choice, merge, fork and join) defining the ordering and causality among the actions. The control nodes are not assigned to any particular component as well.

We believe that this is the right level of abstraction to discuss the intended behavior with end-users and other stake holders since global choreography models specify the global behavior more clearly than interaction diagrams or other diagrams. This is the case as they not only give an object-oriented view of collaborative services and depict their interactions, but also show the causality and execution order of collaborative services. They also hide implementation details of distributed systems like the location of components. A flow-global choreography model is also a useful first step in the formalization of the requirements of a system.

Figure 12.3 (a) specifies the flow-global choreography behavior of the train control service. When the train starts its journey, it will first contact the *RBC* which is in charge of its current region. Afterwards, the train reports its current position to the *RBC*. This operation is specified by the *PositionReport* collaboration. Thereafter, the *RBC* validates the received position information of the train via the *SupervisionLogic* local activity. If the information about the location of the train is correct, the *RBC* will issue successive movement authorities (MA) to the train, i.e., by *MoveAuthority* collaboration. If the train is crossing regions, the *HandoverSupervision* collaboration will be invoked. From such an abstract choreography specification, we want to derive a flow-localized choreography model from which deployable application code can be generated.

12.2.2.2 Flow-Localized choreography model

The flow-localized specification is also defined by UML activities connecting actions, that either represent the behavior of a collaboration or a local activity, by flows and intermediate control nodes that are localized to the participating distributed entities represented by roles. Flows between nodes which are not localized to the same system parts thereby imply communication and transfer delays. This means that the flow-global choreography model abstracts from several design issues that needs to be addressed when transforming it to a flow-localized choreography model. Note that the pins in flow-global choreography model are owned by collaborations itself. In the flow-localized model, these pins need to be bound to a particular service roles. This is done in the way defined in Section 12.3.

The flow-localized choreographies are modeled by UML activities using Arctis [Kra08, KBH09], our tool-set to develop component based collaborative systems. The activities are provided with a formal semantics [KH10] which allows for the application of model checkers to detect design errors [KSH09]. Further, application code for different platforms such as Standard Java, Android and Sun Spots can be automatically generated from the Arctis building blocks. In the train control scenario, this is the code for the train and RBC components.

A screen shot of the Arctis model of the train control system is shown in Figure 12.3(b). There, the collaborative and the local activities are mapped to Arctis building blocks (the dark gray respective blue boxes with pins). Control nodes are localized to components represented by Activity partitions. The extra gray border of the collaborative actions (*pr*, *ma* and *h*) at the *RBC* part represent that the service roles bond to this part are multiple-session.

12.3 Flow Localization

Based on the model provided above, we outline the overall transformation process and our design intent. First, we recall the causal relationship between sequential collaborative activities. Second, we define a localization policy based on the causal properties.

12.3.1 Causality Relationship

In order to localize the flows and control nodes between actions in a flow global choreography, we first need to find out the causality relationship. As described in [CBvB07, KB10], the following causal relationships between any two sequential connected actions C_1 and C_2 can occur:

- *Strong causality:* The terminating role of C_1 and the initiating role of C_2 belong to the same system component. Then the coordination between C_1 and C_2 can be executed locally in this component.

- *Non-causal causality:* The terminating role of C_1 and the initiating role of C_2 belong to different components and also none of the non-terminating roles in C_1 participates in the component of the initiating role of C_2. This means that local ordering between actions of C_1 and C_2 is not possible.

- *Weak causality:* The terminating role of C_1 and the initiating role of C_2 belong to different components but there is a non-terminating role in C_1 and an initiating role in C_2 which are in the same component. Here, one can coordinate the non-terminating role of C_1 and the initiating role of C_2 locally but has to be aware that both collaborations run in parallel for a while.

12.3.2 Localization Policy

In the simplest case where there are no control nodes between actions (direct-flow), the flow between C_1 and C_2 (C_2 comes after C_1) are localized as following:

- Localize *strong* flows to the role that initiates C_2 and terminates C_1.
- In case of *weak* flows, in which the terminating component in C_1 is different from the initiating component in C_2, localize the *weak* flow to the role that initiates C_2 by adding a streaming pin to the non-terminating role in C_1. Otherwise use send and receive messages to resolve the flow, which we also call *enforced* strong sequencing [KB10]. Adding an extra streaming pin to a building block in the flow-localized level will change the internal behaviors of the actions. For the modification of internal behavior, we prepared a set of transformation rules for waving extra behavior into building blocks without effecting the original functional design. This will not be addressed in this paper due to the space limitation.
- $Non-causal$ flows can only be maintained by using enforced strong sequencing using send and receive events (cf. Section 12.4).

If there are one or more intermediate control nodes between actions C_1 and C_2, one must consider all possible flow-paths passing them. This means that each control node can be part of several paths. We use the following notation to represent the path and path property for a control node:

$$Path ::= (sourceNode) \overset{causality}{\to} (targetNode)$$

where $sourceNode$ and $targetNode$ are either pseudo nodes (such as an initial node or an activity final node which are represented by $\star$) or collaboration role identifiers (represented by collaborationId.roleId). The arrows show the flow direction between those collaborative activities with a causality relation above. In the flow-global choreography model in Figure 12.3 (a), there are totally eight paths:

- $P_0 : \star \overset{na}{\to} pr.Train;$
- $P_1 : pr.RBC \overset{strong}{\to} s.RBC;$
- $P_2 : s.RBC \overset{strong}{\to} ma.RBC;$

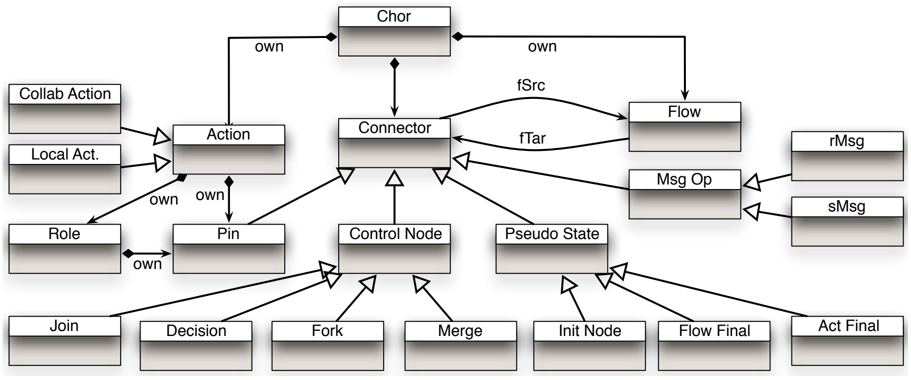


Figure 12.4: Meta-model for the choreography graph models.

- $P_3 : ma.Train \overset{strong}{\rightarrow} pr.Train$;
- $P_4 : ma.Train \overset{na}{\rightarrow} \star$;
- $P_5 : ma.Train \overset{weak}{\rightarrow} h.RBC$;
- $P_6 : h.Train \overset{strong}{\rightarrow} pr.Train$;
- $P_7 : s.RBC \overset{na}{\rightarrow} \star$;

The abbreviation *na* means that there is no causality relationship available. P_0, P_4 and P_7 are dangling paths where one end of the path is a pseudo state node, the pseudo state and control nodes along the path will be localized to the component of the activity node, e.g., the initial node and decision node D_1 will be localized to *Train*. In order to localize the remaining control nodes M_1, D_1 and F_1 and paths P_1, P_2, P_3, P_5 and P_6, we need to consider the path properties that each control node is involved in:

- M_1: $P_6(strong), P_3(strong)$;
- F_1: $P_3(strong), P_5(weak)$;
- D_1: $P_5(weak)$;

M_1 fulfills *strong* causality for both involved paths and is therefore localized to *Train*. In contrast, F_1 and D_1 contain *weak causal* paths such that we need to find suitable breaking points along the involved paths, i.e., P_5 containing both F_1 and D_1. To achieve that, we have to assign a breaking priority level to all the nodes on the path. The priority level is defined by the combination of causality properties in Table 12.1. Here, the columns and rows are the causal relationship of a path through a node and based on that we define the priority order for selecting breaking point as follows:

$$6 > 5 > 4 > 7 > 3 > 2 > 1$$

Table 12.1: Localization priority order and policy matrix for control node.

property	strong	weak	non-causal	all
strong	1	2	3	-
weak	-	4	5	-
non-causal	-	-	6	-
all	-	-	-	7

In the *TrainControlSystem* specification, D_1 is involved in $P_5(weak)$ and has priority level 4 according to Table 12.1. Similarly, F_1 is involved in $P_3(strong)$ and $P_5(weak)$ and has priority level 2. According to our policy, D_1 is selected as breaking node as it has higher breaking priority level than F_1. In this case D_1 is broken at the incoming edge. In the subsequent refinement steps, F_1 in Train is not necessary as D_1 is linked by a streaming pin from the *ma.MovementAuthority* service (according to the weak causality localization) and one can ignore a fork node if it has only one outgoing leg. After the transformation and refinement the flow-localized graph model is exported in the form of Arctis building blocks as shown in Figure 12.3 (b).

12.4 Graph-based Model Transformation

This section discusses how graph transformation techniques can be used in order to perform model transformation from the flow-global choreography model to the flow-localized model. We will tailor the graph models, graph transformation rules and implementation in the following parts.

12.4.1 Graph model definition

The meta-model for the choreography graph model (also called *type graph* in the following) is shown in Figure 12.4 in terms of UML class diagrams. A choreography meta-graph model has three main entities: *Actions*, *Connectors*, and *Flows*. An action can be either a *local activity*, i.e., an activity

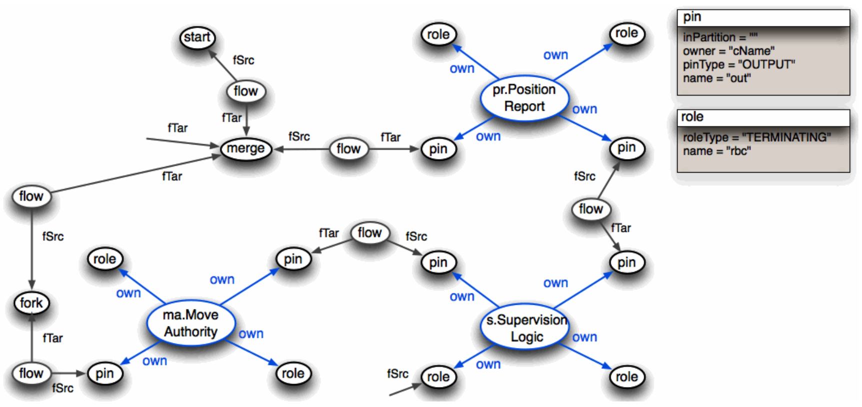


Figure 12.5: Graph model of the part of choreography graph model of Train control scenario

which is performed by one specific role; or a *collaborative activity*, i.e., an activity which is carried out by more than one roles. Connectors represent the mechanism to connect Actions, e.g., how collaborations connect to other collaborations or local activities There are also three types of connectors: *Pin*, *Control Node* and *Pseudo State*: Pins are connection points which are associated with Roles or Actions. Control Nodes include join, fork, merge and decision nodes. Pseudo Nodes include initial nodes, activity final nodes and flow final nodes. Connectors can also send message nodes (*sMsg*) or receiving message nodes (*rMsg*). There are three types of graph edges: *own* specifying that one node is owned by another; *fSrc* modeling the source node of the flow; and *fTar* specifying the flow target.

Based on this type graph, two types of graph models can be defined corresponding to the flow-global and flow-localized choreography models. Figure 12.5 illustrates a flow-global graph model of the corresponding flow-global choreography model of the train control system in Figure 12.3(a) without the Handover activity. Note that there are two dangling edges in the graph model: *fTar* to merge node and *fSrc* to role node. These edges are connected to the Handover activity if required.

12.4.2 Graph Models of Transformation rules

Transformation rules can be visualized and are mainly composed of two graph parts: the pre-pattern subgraph, expressing what to replace, and the post-pattern subgraph describing the replacement. A transformation condition can also be used to define conditions or constraints describing how

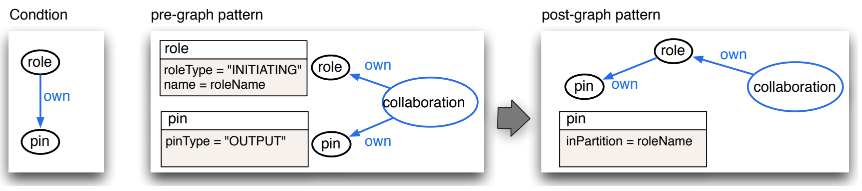


Figure 12.6: Pin localization rule graphs

and under which condition the graph production can be applied (such as a negative application condition NAC introduced in [Tae04]). In the following, we introduce the graph models of transformation rules and policies by selecting some representative rules: graph models of the pin location rule and the direct flow localization rule.

Graph Models of Pin Localization Rule: Figure 12.6 illustrates the graph transformation rule for pin allocation. Pins in the pre-pattern of the graph model are owned by collaboration nodes (shown in Figure 12.6(a)). They are localized, i.e., connected to roles in the post-pattern of the graph (shown in Figure 12.6(b)). Note that attributes and their values to the graph nodes can be used, checked, or modified. For example in Figure 12.6, *roleType* and *pinType* attributes are checked in the *role* and *pin* node in the pre-pattern of the graph model. Similarly, the *inPartition* attribute of the pin node is modified (or assigned a value) in the post-pattern of the graph.

Graph Models of Direct flow localization rules: Figure 12.7 shows the pre-pattern and post pattern of the rules which localize the direct flow having *weak* causality and *non-causal* causality properties. Figure 12.7(a) shows the pre-pattern graph model of the direct flow in which pin allocation rules have been performed. If the flow has *weak* causal property, then the corresponding post-pattern is shown in Figure 12.7(b). In this case, a new output streaming pin is added to a collaborative activity *Ci*. An example of this type is P_5 in the case study. In the case where a direct flow-path has a *non-causal* property, the flow-path is resolved using send and receive message nodes as shown by the post-pattern graph model in Figure 12.7(c).

12.4.3 Implementations

An Eclipse plug-in has been developed to create graph models of flow-global choreography models and to import the post-graph (as a result of transformation) into Arctis (a service engineering tool developed in our research group) [KSH09]. As a system for graph transformation engine, we use the Attributed Graph Grammar System (AGG) [Tae04]. AGG offers a high flex-

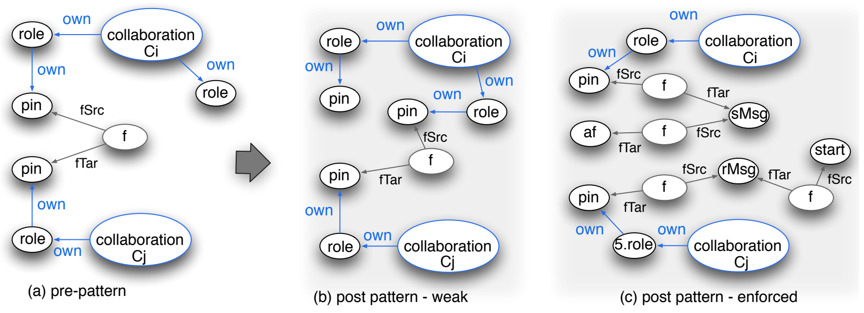


Figure 12.7: Direct flow localization rule.

ibility in creating the visual definition of graph models. Further, it provides Java APIs facilitating its integration to the Arctis tool which is also Java-based. AGG has also a facility to enable the verification and correctness of models during transformation.

The AGG graph transformation engine takes the graph model of a flow-global choreography (partly depicted in Figure 12.5) as well as the rules introduced in Section 12.4.2 as inputs and produces the post-graph model which corresponds to a flow-localized choreography of the train control system.

12.5 Related Work

A comparison of approaches and tools that use graph transformation techniques as intuitive mechanisms for capturing model transformations is provided in [VAB$^+$08]. In [KCBL10], Kerkouchea et al. propose an approach for transforming UML state-chart and collaboration diagrams to Colored Petri nets using graph transformation techniques. In contrast, the authors of [REC11, VAB$^+$08] suggested to map activity diagrams into communicating sequential processes (CSP) processes.

Unlike these approaches, both our abstract and detailed models are based on UML activity diagrams. Moreover, our approach envisages collaborative building blocks encapsulating the interaction between different components. In addition, UML activities can be defined hierarchically by hiding an activity as a call behavior action of another one. We use AGG as graph transformation engine and our post graphs can be directly imported to the Arctis tool for further analysis, synthesis and code generation.

Unlike [RR09, JWEG07], we currently miss the formal proof that our graph-based transformation is correctness-preserving. Due to the formal semantics of Arctis [KH10], however, which can also be used for the flow-global choreography models, the correctness verification can be provided as temporal logic-based refinement proofs [Lam94] which is intended for the close future.

Ideally, one can choose any graph transformation tool to validate or implement our approach. Some candidates besides AGG were ATOM3 [LV02], VTMS [LLMC06] and C-SAW [ZLG05]. As mentioned above, we chose AGG due to its high flexibility and its Java-compliance.

12.6 Concluding Remarks

In this paper, we presented a Model-Driven Development approach to support the engineering process of distributed collaborative systems. The global behavior and realization of the distributed systems are captured and synthesized by two different types of choreography models: flow-global and flow-localized, which are specified by UML activity diagrams. The transformation between these two choreography models are performed with the support of graph transformation techniques. The approach is used within the EU-funded project CESAR for the cost-effective development of safety-relevant embedded system [Ces11]. As future work, we plan to refine model transformation policies and test them with larger and more complex system models. As mentioned above, we will further prove the correctness of the graph transformation rules formally.

References

[CBvB07] Humberto Nicolás Castejón, Rolv Bræk, and Gregor von Bochmann. Realizability of collaboration-based service specifications. In *Proceedings of the 14th Asia-Pacific Software Engineering Conference*, pages 73–80, Washington, DC, USA, 2007. IEEE Computer Society.

[Ces11] Cesar project website. http://www.cesarproject.eu/. accessed jan 2011, 2011.

[JWEG07] Praveen K. Jayaraman, Jon Whittle, Ahmed M. Elkhodary, and Hassan Gomaa. Model composition in product lines and feature interaction detection using critical pair analysis. In *MoDELS*, pages 151–165, 2007.

[KB10] Surya Bahadur Kathayat and Rolv Bræk. From flow-global choreography to component types. In *System Analysis and Modeling (SAM)*, volume 6598 of *Lecture Notes in Computer Science*. Springer, 2010.

[KBH09] Frank Alexander Kraemer, Rolv Bræk, and Peter Herrmann. Compositional service engineering with arctis. *Telektronikk*, 105(2009.1), 2009.

[KCBL10] Elhillali Kerkouche, Allaoua Chaoui, El-Bay Bourennane, and Ouassila Labbani. A UML and colored petri nets integrated modeling and analysis approach using graph transformation. *Journal of Object Technology*, 9(4):25–43, 2010.

[KH10] Frank Kraemer and Peter Herrmann. Reactive semantics for distributed uml activities. In John Hatcliff and Elena Zucca, editors, *Formal Techniques for Distributed Systems*, volume 6117 of *Lecture Notes in Computer Science*, pages 17–31. Springer Berlin / Heidelberg, 2010.

[Kra08] Frank Alexander Kraemer. *Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks.* PhD thesis, Norwegian University of Science and Technology, 2008.

[KSH09] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann. Tool support for the rapid composition, analysis and implementation of reactive services. *Journal of Systems and Software*, 82(12):2068–2080, 2009.

[Lam94] Leslie Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16:872–923, May 1994.

[LLMC06] László Lengyel, Tihamér Levendovszky, Gergely Mezei, and Hassan Charaf. A visual control flow language. In *Proceedings of the 5th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems*, pages 30–35, Stevens Point, Wisconsin, USA, 2006. World Scientific and Engineering Academy and Society (WSEAS).

[LV02] Juan de Lara and Hans Vangheluwe. Atom3: A tool for multi-formalism and meta-modelling. In *Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering*, FASE'02, pages 174–188, London, UK, UK, 2002. Springer-Verlag.

[REC11] Houda Hamrouche Raida Elmansouri and Allaoua Chaoui. From uml activity diagrams to csp expressions: A graph transformation approach using atom3 tool. *IJCSI International Journal of Computer Science Issues*, 8(2):368–374, 2011.

[RR09] Vahid Rafe and Adel Rahmani. Towards automated software model checking using graph transformation systems and bogor. *Journal of Zhejiang University - Science A*, 10:1093–1105, 2009.

[Tae04] Gabriele Taentzer. *AGG: A Graph Transformation Environment for Modeling and Validation of Software*, volume 3062 of *Lecture Notes in Computer Science*, chapter 35, pages 446–453. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[VAB$^+$08] Dániel Varró, Márk Asztalos, Dénes Bisztray, Artur Boronat, Duc-Hanh Dang, Rubino Geiß, Joel Greenyer, Pieter Van Gorp, Ole Kniemeyer, Anantha Narayanan, Edgars Rencis, and Erhard Weinell. Applications of graph transformations with industrial relevance. chapter Transformation of UML Models to CSP: A Case Study for Graph Transformation Tools, pages 540–565. Springer-Verlag, Berlin, Heidelberg, 2008.

[ZLG05] Jing Zhang, Yuehua Lin, and Jeff Gray. Generic and domain-specific model refactoring using a model transformation engine. In *Volume II of Research and Practice in Software Engineering*, pages 199–218. Springer, 2005.

Chapter 13

Paper 8

Title: Collaboration-based Model-Driven Approach for Business Service Composition.

In: Handbook of Research on E-Business Standards and Protocols: Documents, Data and Advanced Web Technologies. Ejub Kajan, Frank-Dieter Dorloff, Ivan Bedini (Eds). IGI Global.

Publisher: IGI Global

Collaboration-based Model-Driven Approach for Business Service Composition

Surya Bahadur Kathayat, Hien Nam Le, and Rolv Bræk
Norwegian University of Science and Technology (NTNU)
Department of Telematics
Trondheim, Norway
Email:{surya, hiennam, rolv.braek}@item.ntnu.no

Abstract

We argue that business processes can be modeled in the same way as collaborative business services and therefore use an approach developed for such services. We consider business services that are collaborative and crosscutting in nature: several participants may collaborate in a business service to achieve its goal and a participant may take part in several different business services, playing different roles in each. A framework to support the development and composition of such business services is the main focus of this Chapter. We use UML collaborations for modeling the structure of roles involved in a business service/process, and activity diagrams for specifying the global behavior performed by the roles. From these models, reusable components realizing the roles can be automatically synthesized and such components can then be composed together in order to make different systems that meet the requirements of business services.

keywords: Service, Service Composition, UML Collaboration, UML Activity Diagrams, Chorepgraphy, Orchestration, Semantic Interface.

13.1 Introduction

A business process is commonly defined as an ordered execution of a combination of related activities whose goal is to achieve a desired effect for customers [OCK+11, vdAHW03, CST10].

A business process may have active and/or passive participants. These participants can be people, computer systems, devices or tools [CST10]. Passive participants only respond to external requests and never take independent initiatives on their own. Active participants can take initiatives to perform actions and send requests to other participants. Often each participant takes part in several different business processes, so in general, the behavior of business processes are composed and combined from partial participant behaviors. This is sometimes referred to as the crosscutting nature of business processes [BKM07]. Due to the heterogeneous and dynamic nature of business processes, there are many challenges related to business process modeling and analysis [vdAHW03].

We consider that business processes are collaborative and crosscutting in nature, i.e., several participants may collaborate in a business process to achieve its goal; and a participant may take part in several different business processes, playing different roles in each. We will use a general definition of service which defines a service as *"an identified functionality aiming to establish some desired goals/effects among collaborating entities"* [BCLR10]. On a suitable level of abstraction, we see that business services and business processes are very similar. Therefore we assume that business processes and business services can be modeled in the same general framework.

Our approach is based on model-driven development where a workflow of business services is specified using UML notations, in particular UML collaborations and activities. We use collaborations to specify service structure and consider a web-service as two-party collaboration in which one part is playing a requester (initiating) role and the other is playing a responder (participating) role.

The framework constitutes a systematic approach to specify, analyze and compose collaborative business services, with automatic synthesis of components that realize services and service roles. Component types that are synthesized from service models [BCLR10] can be stored into a repository and re-used to compose systems having different capabilities. In some situations it may be desirable that components (or component types) can first be composed into composite components and then into systems. In this case the complete system behavior emerges from the composition of the components.

In summary, the proposed approach provides the following novel contributions:

1. Modeling and designing business services on a global level in such a way they can be easily analyzed and re-used.

2. A method to compose existing business services to create new business services that not only meet customers requirements but also have the capacity to deal with the dynamic situations of services or users.

3. A method to synthesize components from service models and to compose them in order to provide different system design models. Components may be encapsulated and retained for later reuse.

Using this approach, we aim to provide the following benefits to the business service developer:

- **Abstraction and reusability:** Business process design normally occurs at early state of system development. Business process developers need to work with customer requirements. We provide abstract notations and mechanisms for services and their composition. We encapsulate collaborative interactions among service participants in reusable building blocks so that they can be reused in different contexts with little or no additional cost.

- **Flexibility:** Business process developers may compose new composite services just by dragging/dropping and connecting service building blocks together.

- **Correctness by construction:** Individual and composite services can be model-checked once and reused without having to recheck internal correctness. Correctness of links between components is ensured by checking compliance to contracts defined as collaborations.

- **Automation and cost-reduction:** We provide automation for the proper technical implementation of business processes. We provide mechanisms to automatically synthesize components and their implementation from choreography models. This helps towards cost-reduction and speed of system development.

The rest of the chapter is organized as following. The next section *State of the Art* describes the comparison with related works and motivation for the chapter. The *Background Concepts* section summaries the basic concepts of the business service composition framework. Thereafter the composition of services and components in the framework are explained in section *Composing Business Services* and *Orchestration Models*. The *Component Composition Pattern* and *System Design* sections address how existing service components are composed into service systems. The *Implementation* section discusses the implementation issues of the proposed approach. Finally *Conclusions* are given. A *City Guide* service will be used to illustrate concepts and techniques of the framework throughout this chapter.

13.2 State of the Art

Unlike web-services where a service provider provides available services upon request from the service requester, we consider that individual service participants may take initiatives on their own. Sometimes such independent initiatives may be initiated simultaneously and collide. This is called mixed-initiative problem [BF04], which is normal and unavoidable case in complex reactive system but largely ignored in most existing works on business services. Our work provides mechanisms to detect and resolve such problems early in the service design and composition.

There is a large body of work in the design (modeling and composition) of business services [PTDL08, DS05, Jis04]. Most of this assumes that the business services are provided by a single system component as a local activity. However we consider business services as collaborative activities and encapsulate collaborative interactions in reusable building blocks so that a service developer can reuse such building blocks in different contexts. This provides flexibility in composition and potentially reduces the cost.

In the web services domain, different languages such as BPMN, BPEL, WS-CDL, and UML activity diagrams are used for the composition of business processes. BPMN provides graphical notations for specifying business process workflows. BPEL and WS-CDL are XML-based languages for orchestration and choreography respectively. They have different semantics: BPEL is focusing on the local view of one business participant; WS-CDL is focusing on global behavior specifications. According to [BO10, PBA$^+$08], UML activity diagrams and BPMN have almost equal expressive power for the business process specification. There is a body of work for mapping BPMN to UML and vice versa using UML profiles. We use UML collaborations to specify the structure of roles participating in a service. We use UML activity diagrams (AD) to specify the global behavior associated with a collaboration, the so-called choreography, as well as the local behavior of roles, the so-called orchestration. We use UML activity diagrams to model both orchestration and choreography specifications in a unified way.

In the domain of semantic-web technologies, concepts like ontologies are being used to compose or assist the composition of services where services (and their composition) are specified using languages such as such as OWL-S, DAML-OIL and WSDL-S. These are text-based approaches and intended to add semantics to the services in terms of constraints, properties, policies, goals and etc. Our approach of service composition is intended for the higher-level service composers who can just drag, drop and connect the services together in order to make a composite service. The service developer need not be a technical expert but rather a domain expert. There is also a body of work on business process composition based on AI planning where composition of services is seen as a planning problem and the plan is generated from a set of services in order to reach a business process goal.

In general, service composition strategies are classified as static composition, semi-static and dynamic composition depending on the time of composition and service binding time [Lau06]. Static composition is also called design time composition where a service composer discovers, binds and assembles services during service development. In dynamic service composition, a composition plan is generated at runtime based on the requesters service request. Our framework is concerned with design time composition. However, we provide mechanisms to support dynamic discovery and linking.

Some related approaches are summarized in Table 13.1. Most of these describe choreography where activities are localized to service participants [MH05]. We call these *flow-localized* models. We also provide support for more abstract choreography models that are closer to the problem domain called *flow-global* choreography models. Furthermore we automatically synthesize the component types from the abstract choreography specifications. Most of the approaches do not really take into account the structure of collaborating entities in a service. They are just considered as set of roles in [KP06, BGG$^+$06, QZCY07]. We define the structural model of a service using UML collaboration diagrams that provides specific information such as which participant uses which service and interacts with which other participants, etc. They also support service composition by means of collaboration uses and role bindings.

In the reactive system domain, languages such are MSC, UCM, and Interaction Overview Diagrams are used for choreography specification. There is also a large body of work based on formal approaches such as labeled transition system (LTS) in [KP06], set of conversations in [BGG$^+$06], and activity traces in [QZCY07]. We use UML activity diagrams and their semantics [OMG09, KH10] for specification and composition of services. This enables us to define complete behavior that can be directly realized, unlike the partial or fragmented scenarios normally provided by sequence diagrams and MSCs. Compared to these approaches, specification using UML activity diagram is more abstract and complete, as discussed in [KB10].

Different techniques have been used to check realizability of choreography specifications. In [SB09] behavioral equivalence between the LTS of choreography and a LTS of a parallel composition of orchestrations is used. [BGG$^+$06, KP06] uses the bi-simulation equivalence for this. Trace equivalence is used in [QZCY07]. In our work we do not explore global state spaces, but provide rules for analyzing the choreography directly based on flow-analysis.

The main features that distinguish our approach from the other approaches discussed above are the following:

- It is an engineering approach indented for practical service and system development.
- It uses the same notations, UML collaborations and activity diagrams, for choreographies as well as orchestration.
- It supports compositional development of collaborative services involving active objects and mixed initiatives.
- It supports automatic synthesis of the orchestrators with interface contracts.

13.3 Background Concepts

The proposed *Business Service Composition Framework* is shown in Figure 13.1. The framework has two main model categories: service models and design models. The service models include structural models and choreography models. Structural models specify the structure of roles participating in a service as well as the involved sub-services represented by collaboration uses. Choreography models specify the global behavior including the ordering and causality among sub-services and local actions. The design models define design solutions as structures of inter-connected parts. The behavior of parts is called orchestration.

13.3.1 Structural Models

At the early service modeling stages it makes little sense to identify the implementation level components that will participate in providing the service. One should rather focus on identifying components representing domain entities, such as users and user groups that are involved in the service [BH93]. It is important to identify the properties and behavior that those components shall have, and specify these as service roles. UML 2.x collaborations are well suited to describe the structures of collaborating

Table 13.1: Summary of the related works

	[KP06]	[BGG$^+$06]	[MH05]	[QZCY07]	[SB09]	**Our work**
Partici-pant	roles	roles	services	roles	peers	roles
Partici. struct.	set of roles	set of roles		set of roles	UML comm. dia.	UML collab. dia.
F-L Chor not.			UML AD		UML comm. dia.	UML AD
Chor. se-man-tic	LTS	set of conver-sation		activity traces	LTS	UML AD, [KH10]
Orches. nota-tion		process traces	UML AD, BEPL			UML AD
Orhes. se-man-tics	LTS	process (ax-ioms, rules)	UML AD, BEPL	activity traces	LOTOS process	UML AD seman.
Synthe-sis	no	no	no	yes	yes	yes
Confor-mance	bi-simulation			trace equiva-lence	behavior equiva-lence	by syn-thesis
Impl.		no	yes	no	no	yes
Comm.	async	async	sync/async	async	sync/async	async
Mixed Init.	yes	no	no	no	no	yes
Data	yes	no	yes	no	no	yes

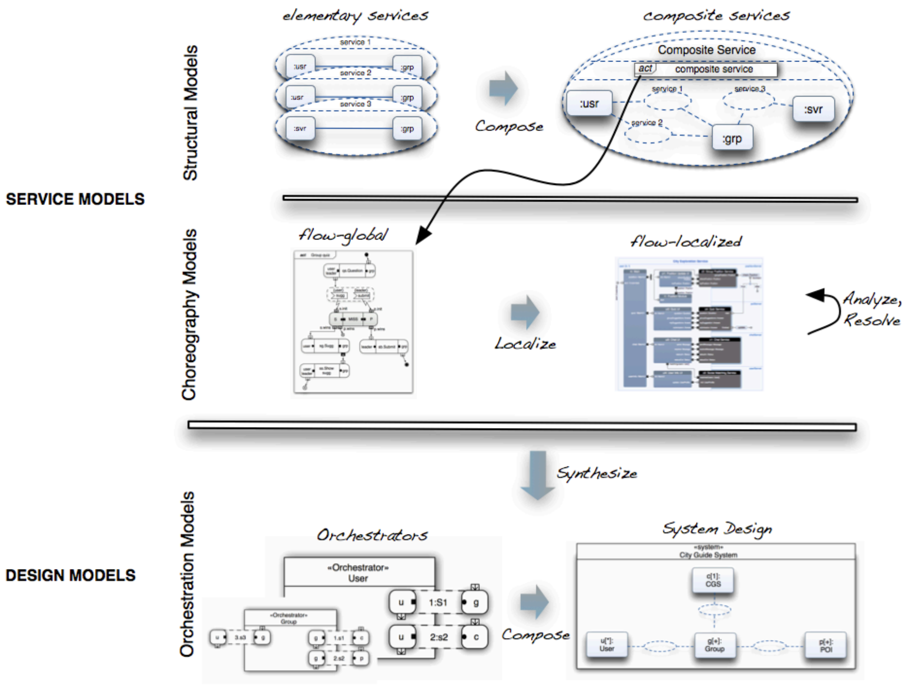


Figure 13.1: Business Service Composition Framework

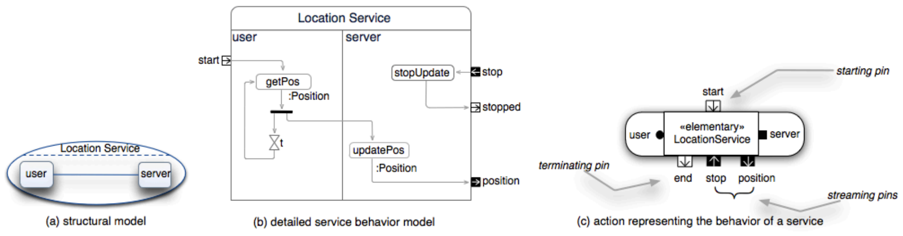


Figure 13.2: Structural and behavioral models of the Location Service

entities (as service roles) and to provide a container for service behavior [BCLR10, KKB11, KLB10].

We distinguish between elementary and composite services. Elementary services cannot be decomposed further, while composite services are composed from other smaller services. When services are composed into a composite service, roles of the sub-services are bound to roles of the composite service.

We consider an elementary service called *LocationService* as a representative example for illustration purpose. The *LocationService* is shown in Figure 13.2 and the desired goal of the service is to allow a user to continuously update the current location information to the server. How this service is modeled and how it will be composed with other services is discussed in the following.

Figure 13.2(a) shows the structural model of the *LocationService* defining the service roles user and server. Being an elementary service, it is not further decomposed into collaboration uses.

13.3.2 Choreography Models

UML Activity Diagrams are well suited to represent the global behavior i.e., the choreography of a composite service. The execution ordering of elementary sub-collaborations can be described using activity diagrams where actions are *CallBehaviorActions* that invoke the activity defining the behavior of a collaboration use. One can specify the *choreography* or global ordering of the collaboration uses of a composite collaboration without localizing all control flows to roles [CBvB07]. This is called a *flow-global* choreography. This allows to specify the ordering intended by users without needing to handle the details of localization and the resolution of coordination problems that must be handled in a distributed realization. Such flow-global choreography can be transformed into more detailed choreography called *flow-localized* choreography where all control flows are localized to partitions representing service roles and all coordination problems are resolved. Therefore the specification is sufficiently detailed to allow extensive analysis and to automatically synthesize the behavior of component types providing orchestration.

The choreography of the *LocationService* is defined in flow-localized form in Figure 13.2(b). The participating roles user and server are represented as partitions in the diagram using a swim-lane notation. A service may interact with other services or the environment via different types of pins. The *LocationService* is initiated by a triggering token entering via the *start* pin in the *user* partition. Each participant may also have local operations (also called *local actions*) and interactions with other participants. For example, in the *user* partition, there is one local operation named *getPos* that periodically reads the geographical coordinates of the user. This information will be send to the server partition via the flow to the *updatePos* local action. The *LocationService* can be terminated at the server side via a streaming pin *stop*. Emitting an output token at the *stopped* pin indicates termination of the *LocationService*. This service model can be stored in the repository as a reusable building block that encapsulates the interactions [KSH09].

Figure 13.2(c) shows the abstract representation of the *LocationService* building block behavior as an action. This representation is used in flow-global choreography models (discussed in *Composing Business Services* section). Note that in Figure 13.2(c), the pins *start*, *end*, *stop*, and *position* are not localized or associated to any partition (unlike the pins in Figure 13.2(b)). In addition, the initiating and terminating roles in a global service building block are indicated by black filled circle and square boxes respectively. For example, in this case, the *user* is an initiating role and the *server* is a terminating role. This provides useful information for analysis and ordering purposes.

13.3.3 Component Orchestration

Orchestration models can be synthesized from the choreography models using projection techniques. Orchestration denotes the local behavior of each participant and is needed to completely define component behaviors for implementation. An orchestrator may have well defined interfaces and internal behavior including the information about the services it participates in and their ordering. This means that the internal behavior may consists of sub-service role behaviors.

Orchestrators (alternatively called components in the following) can be represented as reusable components and be composed together to make a system design, as shown in Figure 13.1. System design models contain

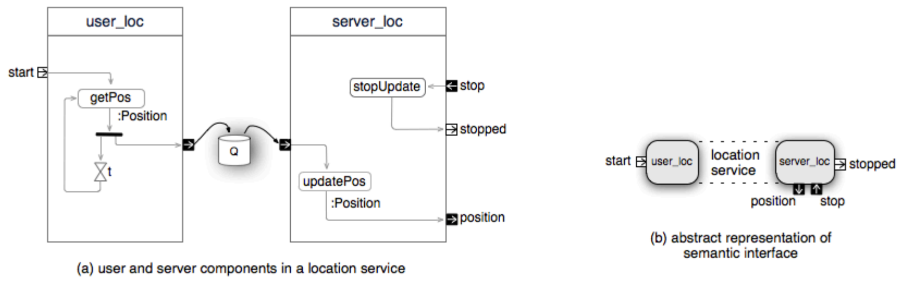


Figure 13.3: Component models and semantic interface corresponding to the Location Service

information like multiplicity of the orchestrator types and other system specific information.

In the case of elementary services, we have used a simple projection technique for component orchestration. That means orchestration models are derived by tearing apart partitions, and replacing flows crossing partition boundaries by streaming output pins and streaming input pins. For example Figure 13.3(a) shows the *user_loc* and *server_loc* components that are generated from the *LocationService* model in Figure 13.2(b). The pins owned by the user and server partitions in the flow-localized choreography remain mapped to the *user_loc* and *server_loc* components respectively. The interactions across the partition boundary are replaced with input and output-streaming pins linked by a queue that models a communication medium. Note that all such interactions among the service components can also be encapsulated in a special type of interface called *semantic interfaces*. The semantic interface has been defined in [JFS08] using state machines defining the observable behaviors at the component interfaces. It is refined in [KLB11] where a collaboration with associated activity diagram is directly used to define a semantic interface. In this way a semantic interface becomes a reusable building block with associated pins for composition. Note that semantic interfaces may also be stored in a repository for future re-use.

Similar to the abstract representation of services, we also make use of an abstract representation of the semantic interfaces. For example, the *LocationService* defines a semantic interface as shown in Figure 13.3(b). Each of the roles will be bound to different system components as illustrated in Figure 13.1.

13.4 Composing Business Services

This section will discuss how to reuse and compose existing business services to create new business services to meet the demand of users. To illustrate our approach a *CityGuide* service is used as a representative example. The *CityGuide* service is a composite service i.e. it is composed from a set of sub-services. This service will be deployed at the tourist office server in a city to provide services for tourists visiting the city. The tourists make use of their mobile devices to connect to the server and download the *CityGuide* service for their visits. The *CityGuide* service is composed from the following three groups of services:

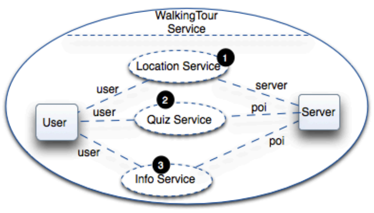
- *Core services.* These services may be running all the time in a system. All other services may depend on core services. A *LocationService* is an example of a core service and when the *LocationService* ends, depended services cannot be provided.

- *Supporting services.* These services help the tourists when they walk around the city. They are built on top of core services. A typical example of a supporting service is a *WalkingTour* service. Based on the situation, preferences and interests of tourists, the *WalkingTour* service recommends the best walking route through a list of points of interest (POI) in the city such as historical buildings, museums, temples and churches. Note that while tourists walk around the city, the *LocationService* is running in the background.

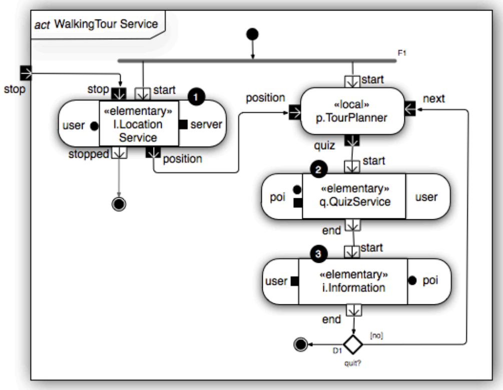
- *Added-value services.* Based on the situation, preferences and requests from the tourists, the system may provide different recommendation services such as recommendation about close-by restaurants via a *RestaurantService.* Such recommendation services can be composed with the *Supporting* services such as the *WalkingTour* service to enhance the quality of the *CityGuide* service to the users. (This will be discussed to illustrate component composition in *System Design* Section).

13.4.1 Structural Model of a Composite Service

The structural model of composite services is represented using UML collaborations. Sub-services in a composite service are represented as *collaboration uses* where roles of the collaboration uses are bound to the roles of



(a) Structural Model of WalkingTour Service



(a) Flow-global Choreography Model of WalkingTour Service

Figure 13.4: Structure and choreography models of the Walking Tour service

the enclosing collaboration. For example, Figure 13.4(a) shows the structural model of a *WalkingTour* service where one can see that the *user* and *server* roles in the *LocationService* are bound to the *User* and *Server* roles of the *WalkingTour* service respectively. Similarly the *user* and *poi* roles of the *QuizService* and Information service are bound to the *User* and *Server* roles of the *WalkingTour* service.

13.4.2 Choreography Model of a Composite Service

The choreography of a composite service specifies the ordering of actions that are either calling the behavior of sub-service, or are local actions. As the behaviors of sub-services are represented as actions, connecting actions together specifies the ordering and causality among them.

The purpose of the flow-global choreography is to define the intended global execution ordering on a high level. It avoids the detail of localization and also resolution of coordination problems that must be handled in a distributed realization. Note that pins that connect to building blocks or control element in a flow-global choreography are not localized to any participating roles. Figure 13.4(b) shows the flow-global specification of the *WalkingTour* service. When it starts, the *LocationService* and *TourPlanner* services start concurrently. Once a TourPlanner receives the location information of a user via the LocationService, it is checked whether the user is close enough to any particular point of interest (POI). If so, a *QuizService* is started where the user participates in a quiz with that particular POI. After a tourist has answered all the questions, a report along with other relevant information is given to the tourist via the *Information* service. The user continues until the end of the walking tour. Note that pins that connect to a service building block or a control elements such as $F1$ and $D1$ in Figure 13.4(b) are not localized to any participating roles.

The purpose of the flow-localized choreography is to allow extensive analysis and to support automatic synthesis of component types. Figure 13.5 shows the flow-localized choreography model of the $WalkingTour$ service. It is different from the flow-global choreography in the following respects:

- In the flow-localized model, every connected pin associated with a service building block must be assigned to a particular role. This means that at this level of modeling, one can precisely specify how a service is started, terminated or being triggered in a distributed system.
- In the flow-localized model, non-collaborative (local) activities and control nodes such as fork, join, merge or decision must also be localized, i.e., assigned to one service role. This may involve mapping global choices onto one or more local choices and communication depending on the composite situation of business processes.
- Control flows in the flow-localized model are derived from the flow-global model. However, the distributed localized flows may have semantic differences compared to the global control flows [KB10] (explained in next section).

In the flow-localized specification of the $WalkingTour$ in Figure 13.5, fork node $F1$ and decision node $D1$ are localized to the user. This means that the flows: (a) from $D1$ to the input streaming pin next of $p.TourPlanner$ building block (which is localized to the server), and (b) from $F1$ to the starting pin *start* of $p.TourPlanner$ building block are interaction flows, i.e. not localized to the same role.

In general, a flow-localized model helps to make the flow-global specification complete in terms of the distributed realization. The rules for transforming flow-global choreography models into flow-localized models are summarized in the next section.

13.4.3 Summary of Choreography Model Transformation Rules

The flow-global choreography can be transformed to the flow-localized choreography by using a set of transformation rules, called localization rules [KB10,

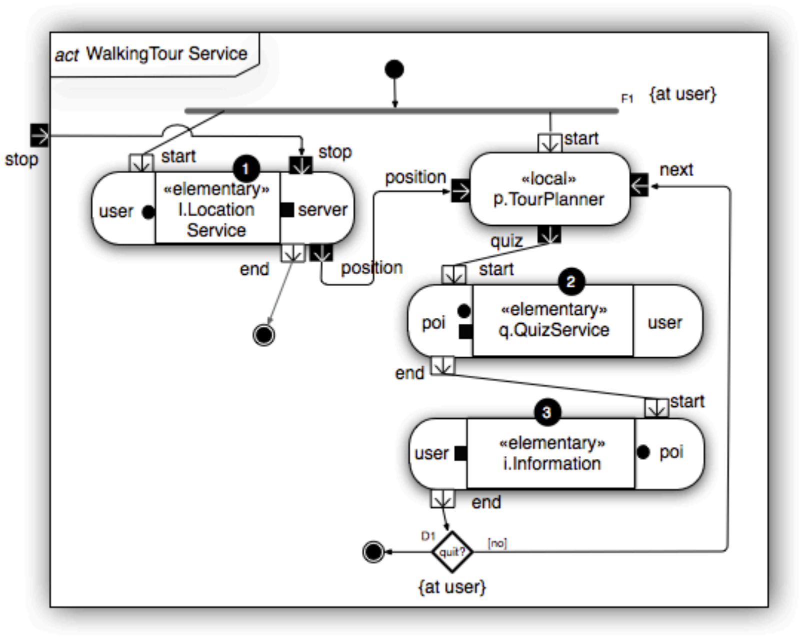


Figure 13.5: Flow-localized choreography model of Walking Tour Service

HKLB11]. Interestingly one can also check for errors or flaws in the flow-global choreography model during the localization process. Such flaws may result not only in profit loss in a business services but also lead to system failure. Therefore it is essential to detect possible problems and resolve them early during the design of choreography models.

The localization rules can be classified into the following three categories: (1) Pin localization, (2) Direct flow localization, and (3) Indirect flow localization with intermediate control nodes. These are discussed in the following:

Service pins in a flow-global choreography model are localized in such a way that starting pins are localized to initiating roles; terminating pins are localized to terminating roles. For example the *start* and *end* pins of the *LocationService* are localized to the *user* and *server* roles respectively.

Direct flows means that there are no intermediate control nodes between services C1 and C2. For example a flow from *QuizService* to *Information* service in Figure 13.4 is a direct flow. In general, direct flows can have following possibilities:

- Strong sequence localization: this is a case where the terminating role of C1 and the initiating role of C2 are localized to the same component. It implies that all the roles in C1 will be completely finished before C2 is started.

- Weak sequence localization: this is a case where the terminating role of C1 and the initiating role of C2 are bound to different components but the initiating role of C2 is participating in C1. In this case there is a possible overlapping between C1 and C2 and the flow is indicated with the weak property.

- Enforced strong sequence localization: in this case the initiating role in C2 and the terminating role in C1 are not localized to the same component. Strong sequencing can then be enforced by introducing an interaction between the terminating role of C1 and the initiating role of C2 using send and receive events. This type of flow is indicated with enforced property.

In the case of indirect flows one needs to localize all the intermediate control nodes that are present between action nodes Ci and Cj. We say that two actions Ci and Cj are linked by a *flow-path* from Ci to Cj through a number of intermediate control nodes each linked by direct *flow-steps*. Flow-steps and the intermediate control nodes are localized according to the rules defined in [KB10, HKLB11]. Normally all the flow-steps and intermediate control nodes in a flow-path should be localized to one component. However there may be cases where intermediate flow-steps and control nodes may be part of several flow-paths i.e. localized to different components. This is an indication of a realizability problem and needs to be resolved [CBvB07, KB10]. For example if a decision node is not localized to one single component (this represents a non-local choice) then there may be a mixed-initiative problem [KSH09, KB10]. In mixed-initiative cases, autonomous components may take initiatives independently and nearly simultaneously, resulting in a collision. Mixed initiatives may also be explicitly modeled in flow-global choreography using interruptible regions and interrupting events. Therefore interruptible regions and interrupting events in flow-global choreography models need special attention, especially when all interrupting events are not local to the same component. In such cases, additional coordination will be needed to resolve which initiative to select. The resolution strategy will depend on the problem at hand and is difficult to solve in a general way. In stead one may develop a library of alternative solutions to select from. One way to handle such situations is to assign primary and secondary priorities to the conflicting partners and let an initiative from the primary side be accepted in all cases. For the secondary side, this means that it must be prepared to receive a primary initiative even after it has issued an initiative itself, and obey the primary one [KSH09, KB10].

13.5 Orchestration Models

Given a flow-localized choreography model, the next step is to synthesize orchestration models for each service component. Orchestration models are derived using a projection of the flow-localized choreography to include only local actions performed by the component, c.f. [KLB10]. Information about the collaborations it participates in and the ordering imposed by external activity flows are retained in order to support automatic realization as well as compatibility checks during subsequent composition of components into working systems.

Figure 13.6 shows the orchestration models of two components, *user_walkingtour* and *server_walkingtour* used in the *WalkingTour* service. The orchestrators or component models specify the following aspects:

- A component is a container that contains orchestration of the service roles it performs. The component type *user_walkingtour* for example orchestrates the service roles *user_loc*, *user_q*, and *user_inf*. Similarly the *server_walkingtour* component orchestrates activities associated with the service roles *server_loc*, *poi_q*, and *server_inf*.

- A component contains the pins, local activities and control nodes that are associated with the contained service roles. For example, in Figure 13.5, the fork nodes $F1$ and $D1$ are assigned to the user role, therefore, $F1$ and $D1$ nodes are placed in the in the orchestration model of *user_walkingtour* component. Similarly the *p.TourPlanner* local activity is placed in the *server_walkingtour* component.

- Interaction flows that are either needed because of the localization of control nodes or to enforce strong sequencing are mapped to local flows and pins. In the *user_walkingtour* component, there are two interaction flows that are mapped to local flows: (1) from the fork node $F1$ to the *start.TourPlanner* streaming pin; and (2) from decision node $D1$ to the *next.poi* output streaming pin. We call the pins at the component boundary as *local interfaces.*

- Pins and local flows may also be added to link actions using a *weak* sequencing semantics [CBvB07, KB10]. Such cases are not present in the given example.

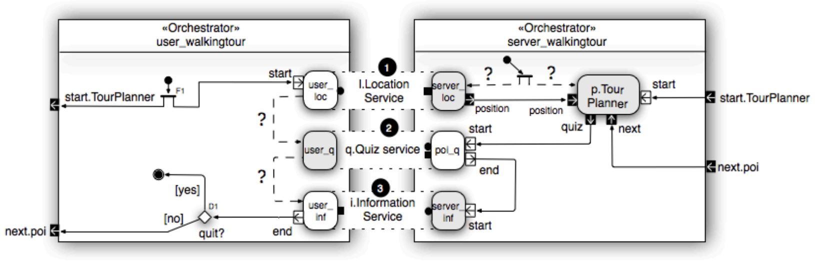


Figure 13.6: Service components of the Walking Tour Service

- The component also has a special type of flows, called *responding flows*, which capture the external flows that will initiate collaboration where the component is participating, but not initiating. In Figure 13.6, the responding flows are marked by the symbol ?. In the following section, we will discuss the composition of components and inner-components within a component and the use of responding flows.

- A component interacts with other components via collaborations that the component participates in. For example, the *user_walkingtour* component participates in three different collaborations: *LocationService*, *QuizService* and *Information* service. Each of these collaborations defines *semantic interfaces* to a component [KLB11].

13.6 Component Component Patterns

Not all of the service roles in a collaborative service may have the capacity to initiate or terminate the service, e.g., via *initiating* or *terminating* pins. This means that if there is no initiating pin in a role, the component will have no clue when the component should start the role. Similarly if there is no terminating pin, we do not know when the role will terminate. This creates a problem, especially when a role is to be composed with other roles. For example, when one wants to make a user component *user_walkingtour* by composing a location user role *user_loc* with quiz user role *user_q* as shown in Figure 13.6, one needs a way to connect them together since there is no terminating pin in the *user_loc* role and no initiating pin in *user_q* role. For this reason the responding flows are marked with a question mark in Figure 13.6.

In order to solve this kind of problems, we introduce a general composition pattern that ensures that each service role will have pins to coordinate

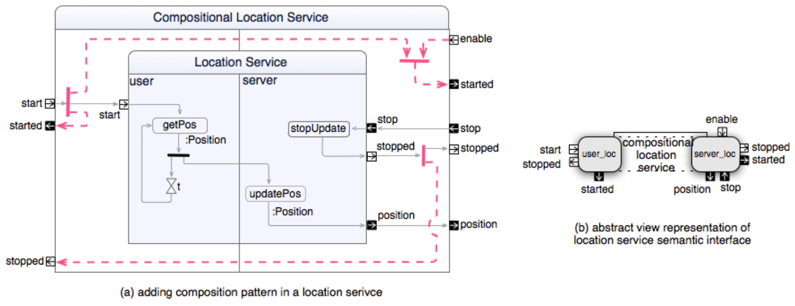


Figure 13.7: Service composition pattern

the enabling, start and termination of collaborative actions. The main novelty of this composition pattern is that, the pre-designed functional behavior of the service is unchanged and additional co-ordination can be added in an enclosing activity indicated by dashed flows in Figure 13.7. The added coordination information includes the following:

- In the initiating role, add a fork before the start pin and send a token to non-initiating roles to indicate that the service activity have been started (in *user_loc* component in Figure 13.7(a)).
- In the terminating role, add a fork after the terminating pin and send token to non-terminating roles to indicate that the service have been stopped (in *server_loc* component in Figure 13.7(a)).
- In the participating role, add a pin to enable the role in responding flows. This is used for the responding flows in the *user_q*, *user_inf* and *server_loc* roles in Figure 13.8.

These adjustments introduce additional interactions between the service roles, especially for coordination purposes. The corresponding abstract view representation the *LocationService* is shown in Figure 13.7(b). The service roles within a composite component can be composed by using pins, and the interaction flows needed to enforce strong sequencing are in place. Figure 13.8 shows the *user_walkingtour* and *server_walkingtour* component models. A responding flow from the *started* pin of the *user_loc* role to the *enable* pin of the *user_q* role represents the possibility of some overlapping behavior between the service roles. A responding flow from the end pin to the enable pin can be used to specify that the succeeding role is enabled only after the preceding role is ended. A responding flow from the initial node (for example in *server_walkingtour* component in Figure 13.8) specifies

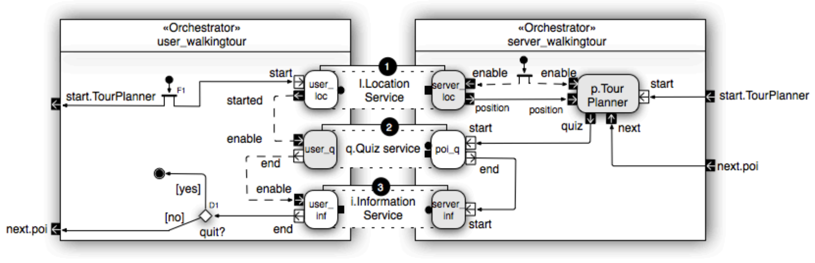


Figure 13.8: Orchestration models of service components of Walking Tour Service

that services are initially enabled when a composite component starts i.e. they can accept the messages in the succeeding services.

There are other benefits of this component composition pattern. A component can for example register itself to the events, such as *start* or *termination*, of other components. This type of mechanism (listening to different events) is very useful in object-oriented programming and in general software-engineering discipline.

Note that the component models can be represented as abstract building blocks by hiding the internal details and showing just the information about interfaces. For example the abstract building block representation for the user component of the *WalkingTour* service in Figure 13.9(a) is shown in Figure 13.9(b). The name and type of the pins corresponding to *local interfaces* is retained. The *semantic interfaces* are abstractly represented by a set ns where n is a reference to an interface (in this chapter we simple use numeric references) and s is a character type whose has values of (1) i representing that component plays initiating role; (2) t representing that component plays terminating role; and (3) p representing that component plays participating role in a component.

13.7 System Design

In order to enable precise and complete definition of service behavior, a service model will normally only consider the roles and behavior involved in a single service. In a system there may be many instances of the same service running concurrently with roles assigned to different system components. There may also be many different types of services provided by the system. In general the system components may play roles in different services, and different components may be able to play different combinations of roles.

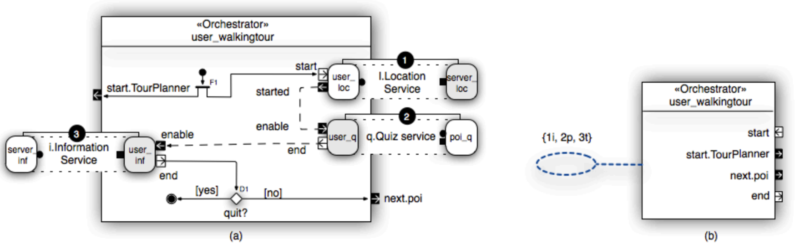


Figure 13.9: Detailed and Black box representation of user component of Walking Tour Service

Consequently there is a difference between system design and service modeling. In system design one needs to develop components that can play different combinations of roles, and one needs to design the overall system as a composition of such components. If necessary the resulting system will enable a dynamic structure where services are combined in ways that are not easily described in service models.

In this section, we demonstrate how our framework can support system design. For illustration purposes, we consider a *CityGuide* service in which a tourist (in a *WalkingTour* service discussed above) wants to subscribe to a *RestaurantService* while he is using a *WalkingTour* service. We assume that user and server components for the *WalkingTour* service and *RestaurantService* are available in the repository.

13.7.1 Component Composition during System Design

During system design, one need not specify a complete system behavior. In stead the complete system behavior emerges from the composition of the components themselves. This means that smaller components can be composed together into larger and composite ones. We now assume a library of component types including *user_walkingtour*, *user_restaurant*, *server_walkingtour*, and *server_restaurant*. The following steps are used:

- Define the part structure i.e., inner structure of a component. For example part structures for the composite user component is shown in Figure 13.10(a).

- Identify the roles (inner roles) to be composed from a component library. In this example, the user uses the *WalkingTour* and *Restaurant* services; therefore corresponding service roles will be selected as shown in Figure 13.10(b).

- Compose the selected roles together using the pins of the selected roles. Pins may be connected based on the data type that the pin owns or they can be connected manually. Some additional control nodes may be added to compose the selected roles together such as fork nodes F1 in Figure 13.10(c). Pins that are not connected to any other pins within the boundary of the component are made transparent to the component boundary as *local interfaces*. For example the *next.poi* and the *start.TourPlanner* pins in the *user_cityguide* component are made transparent to the component border.

Using these steps, the resulting composition of the *user_cityguide* component is shown in the Figure 13.10. In a similar way, the *server_cityguide* component can be composed. Note that composed composite component type can also be represented as building block and put into the library and reused.

13.7.2 Completeness of System Design

In simple cases where a system provides just one service the system design may be defined by a single service model. In more complex cases one need a separate system design activity. For example, a top-level system composition model of a city guide service is shown in Figure 13.11. Note that the User and Server component types (corresponding to *user_cityguide* and *server_cityguide* respectively) interact via collaborations. In the cases where collaborating component has input or output pins at the interfaces, such pins will be represented by a local interface. Moreover, system composition model may specify multiplicity of the parts. For instance, *u[*]:User* implies multiple instances of a User component (and its inner parts) in a system.

A system model consisting of all the service component types is shown in Figure 13.11. In this particular example, we have a Server component that provides *WalkingTour* service: it plays the initiating role in the Location-Service, the participating role in the *QuizService* and the *Session* services, and the terminating role in the *Information* service. A registry component is participating in the *Lookup* service. In a similar way, one can see that an user component is playing the initiating role in the *LocationService*, the participating role in the *QuizService*, the terminating role in the *Information* service, both the initiating and the terminating role in the *Lookup* and the *Session* services. (c.f. Appendix for detailed information about *Lookup* and *Session* services).

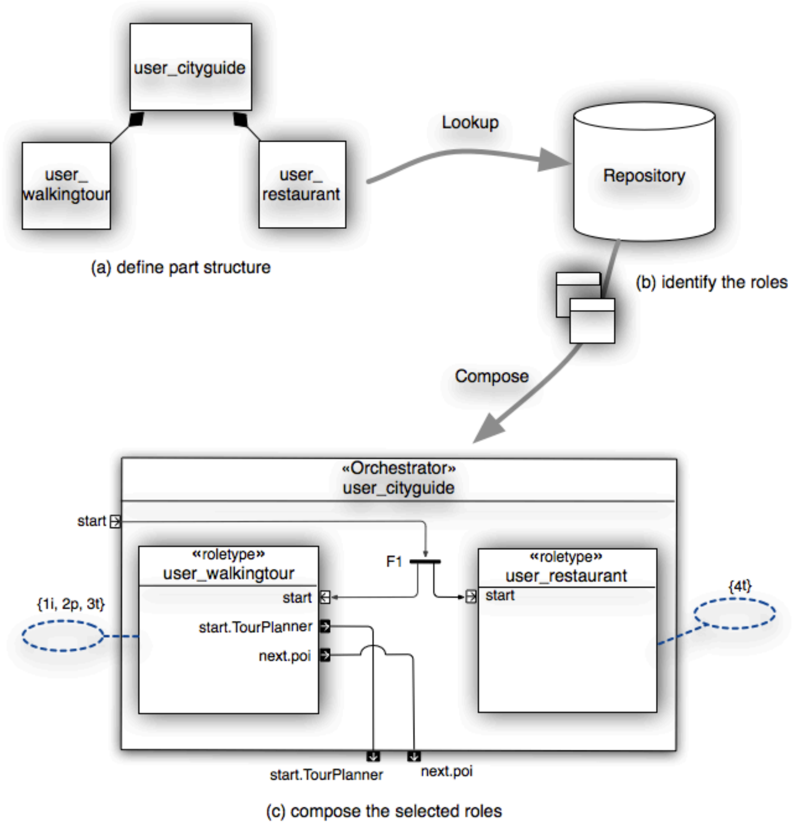


Figure 13.10: Composition of components at system design level

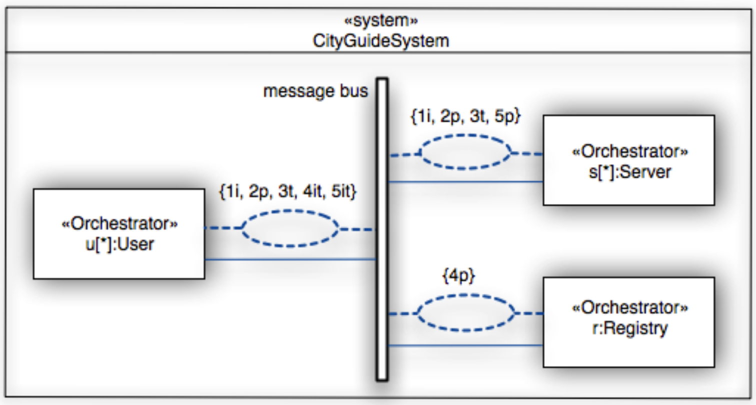


Figure 13.11: System design model of the CityGuide service

13.8 Implementations

We discuss four aspects of the implementation of the approach discussed so far: (1) support for flow-global choreographies, (2) implementation of model transformation rules, (3) support for flow-localized choreographies, and (4) component synthesis and component composition issues.

13.8.1 Flow-global Choreography and UML Profile

The flow-global choreography model uses two features that are not supported in the standard UML 2.x activity specification [OMG09]: assignment of pins to a service itself instead of to participating roles, and the dots and square used to indicate initiating and terminating roles. Those features can be defined using an UML profile as shown in Figure 13.12. The core element in this profile is the service stereotype which extends the UML 2.x action elements. A service contains one or more parts (extending the UML *ActivityPartition* element) and has one or more input pins (extending the UML *InputPin* element) as well as output pins (extending the UML *OutputPin* element). Parts are specified with the Part stereotype, input pins with InputPin and output pins with OutputPin. The type of a part is defined by a *ActivationType* enumeration that uses the literals INITIATING, TERMINATING and PARTICIPATING. Here, INITIATING expresses that the role initiates the service, TERMINATING couches that the role terminates the service, while PARTICIPATING refers to roles that neither initiate nor terminate the service. The types of input and output pins are defined by the *InputPinType* and *OutputPinType* enumerations having various literals. The literals INPUT and OUTPUT represent starting or terminating pins of an action (i.e., a service in our case). The literals INPUT STREAM and OUTPUT STREAM typify that an action may receive or send tokens via those pins without starting or stopping the action. Finally, the literal types INPUT MULTIPLE and OUTPUT MULTIPLE denominate one of multiple starting or terminating pins of an action.

The proposed UML profile can be defined using any standard UML tool or model editor. We used the Eclipse plug-in of the Papyrus UML tool. There is no graphical editor yet fully supporting the modeling of flow-global choreography models, however one can use the tree view based UML model editor for example the UML editor of the Eclipse UML project MDT.

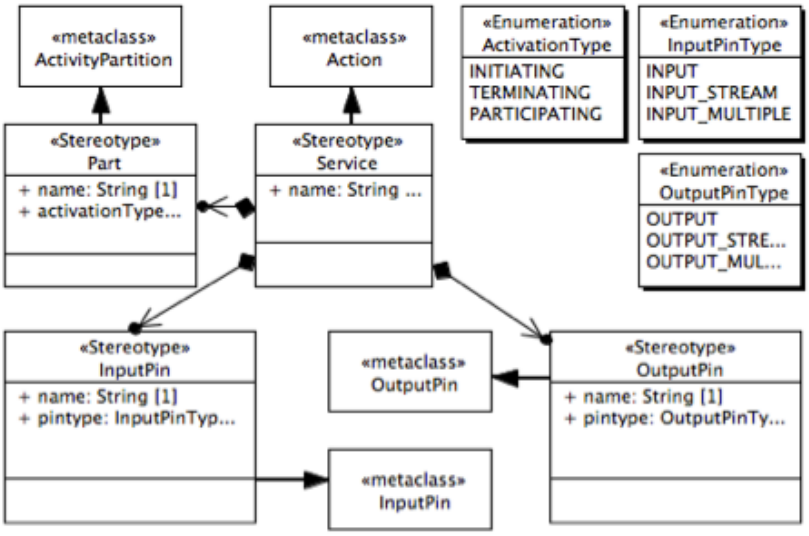


Figure 13.12: UML Profile for supporting flow-global choreography service notations

13.8.2 Model Transformation

The proposed model transformation rules are implemented using graph transformation techniques. Implementation details can be found in [HKLB11]. Flow-global choreography models are represented as pre-graph models. Transformation rules are also represented as graph models. A graph transformation engine takes pre-graph models as inputs and applies rules (graph models) and produces post-graph models corresponding to flow-localized choreography models.

13.8.3 Flow-localized Choreography Model and Arctis Tool

We can import flow-localized graph models in to the Arctis tool [KSH09]. Arctis is a tool for engineering reactive distributed software systems from choreography models and these models are semantically equivalent to our flow-localized choreography models. In particular, Arctis supports a high degree of reuse by offering domain specific libraries of building blocks. Such a building block specifies a sub-functionality that often can be reused in several applications.

Arctis also uses UML collaborations and activities to model the structure and behavior of a system. Arctis uses the same swim-lane based notation of UML activities where service roles are represented as partitions. Note that the service roles of the flow-global choreography model are bound to the corresponding partitions in the Arctis collaboration. Arctis also allows system analysis by means of model checking. Furthermore, application code for different platforms such as Standard Java, Android and Sun Spots [KSH09a] can be generated from the Arctis models.

13.8.4 Component Synthesis and Composition

A complete component synthesis algorithm and component composition techniques are not implemented yet and planned as further works. We do not see any fundamental problems preventing their implementation. Implementation of the synthesis algorithm will be fairly straightforward as both choreography and component models are represented by UML activity diagram. Furthermore, existing techniques [KH07, KSH09a] can be used for transforming UML activity diagrams (of component models) into state machine based models and from there to application code.

13.9 Concluding Remarks

In this chapter, we have presented and discussed a framework to support the development and composition of business services. We use UML collaborations and activity diagrams to specify the structures of collaborating entities of business services and their global behaviors (called choreography models) respectively. In the case of elementary services, the interactions among participating roles in a service are specified using the swim-lane based form of UML activity diagrams. In case of composite services, choreography models are specified by connecting existing building blocks together by specifying their ordering and causality.

The *choreography models* are used to capture the global collaborative business service behavior involving two or more participants, while the *orchestration models* are used to define individual components behaviors for implementation. Given a library of orchestration models, i.e. component types, we discuss their composition. We discuss a component pattern that supports the local composition of components.

Finally, we address system design as composition of existing components to handle different requirements from users. At the system design level, the multiplicity of service instances, and components is also taken into account. Depending on the situation and requirements at hand, a service/system composer may decide the balance between service composition and system composition.

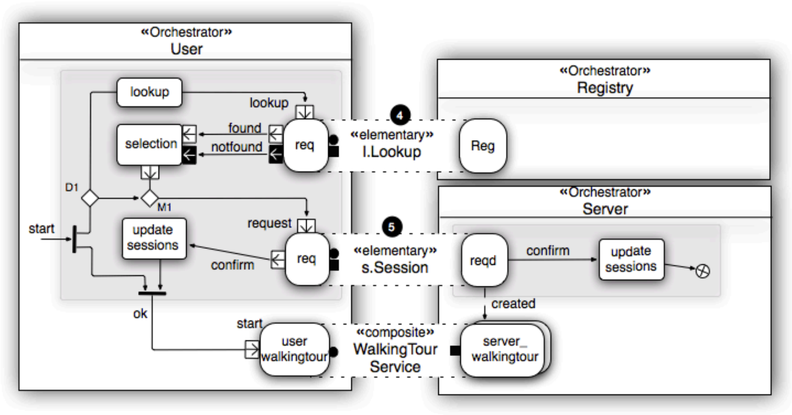


Figure 13.13: Lookup and session support in system design

Appendix

Service Lookup and Session Management

In the following, we outline how the instances can be created and managed in a framework. For illustration purposes, we assume that many users may use the WalkingTour service simultaneously. This means that there will be many sessions of WalkingTour service roles at the server side, as shown in Figure 13.13.

Before requesting a session setup, the requester component i.e., the *user_walkingtour* should know whom to request for the session. We may use a generic *Lookup* service for that as shown in Figure 13.13. The registry contains information about the services, associated roles, and relationship among the services. In our case we assume that a *Registry* will return the identity of a *Server* component that contains the requested role i.e. *server_walkingtour*. The *User* will then do session request using the *Session* service. Once a session setup is confirmed, the *WalkingTour* can be started and the role instances in the *WalkingTour* service will interact with each other. Figure 13.13 also shows how the service roles of *Lookup* and *Session* services can be composed with other service roles within the *User* and *Server* components.

Key Terms and Definitions

Service: Service is partial system functionality where two or more entities collaborate to achieve a goal.
Service Composition: the composition of smaller services in order to make a composite service.
UML Collaboration: UML collaborations define a structure of roles and collaboration uses. They support service composition in terms of collaboration uses and role bindings.
UML Activity Diagrams: UML activity diagrams define behavior in terms of actions that are linked by token flows.
Choreography: Global behavior is called choreography. Choreography is specified using UML activity diagrams. We have used two levels of choreography models: flow-global and flow-localized.
Orchestration: Orchestration defines the local behavior of a component. We use UML activity diagrams to specify orchestration models.
Semantic Interface: A two party collaboration defining interface behavior.

References

[BCLR10] Rolv Bræk, Humberto Nicolás Castejón, Hien Nam Le, and Judith E. Y. Rossebø. Policy-based service composition and recommendation. In *Service Intelligence and Service Science: Evolutionary Technologies and Challenges.* IGI Global, 2010.

[BF04] R. Bræk and J. Floch. Ict convergence: Modeling issues. In *In System Analysis and Modeling (SAM), 4th International SDL and MSC Workshop, pages 237–256, Ottawa, Canada.*, 2004.

[BGG$^+$06] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration conformance for system design. In *COORDINATION, volume 4038 of LNCS*, pages 63–81. Springer, 2006.

[BH93] Rolv Bræk and Oystein Haugen. *Engineering real time systems: an object-oriented methodology using SDL.* Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1993.

[BKM07] Manfred Broy, Ingolf H. Krüger, and Michael Meisinger. A formal model of services. *ACM Transactions on Softwere Engineering and Methodology*, 16, February 2007.

[BO10] Dominik Birkmeier and Sven Overhage. Is bpmn really first choice in joint architecture development? an empirical study on the usability of bpmn and uml activity diagrams for business users. In George Heineman, Jan Kofron, and Frantisek Plasil, editors, *Research into Practice Reality and Gaps*, volume 6093 of *Lecture Notes in Computer Science*, pages 119–134. Springer Berlin / Heidelberg, 2010.

[CBvB07] Humberto Nicolás Castejón, Rolv Bræk, and Gregor von Bochmann. Realizability of collaboration-based service specifications. In *Proceedings of the 14th Asia-Pacific Software*

Engineering Conference, pages 73–80, Washington, DC, USA, 2007. IEEE Computer Society.

[CST10] Artur Caetano, António Rito Silva, and José Tribolet. A method for business process decomposition based on the separation of concerns principle. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 79–85, New York, NY, USA, 2010. ACM.

[DS05] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *Int. J. Web Grid Serv.*, 1:1–30, August 2005.

[HKLB11] Fenglin Han, Surya Bahadur Kathayat, Hien Nam Le, and Rolv Bræk. Towards choreography model transformation by graph transformation. In *Proc. of 2nd IEEE International Conference on Software Engineering and Service Sciences*, Beijing, China, 2011.

[JFS08] S. Jiang, J. Floch, and R. Sanders. Modeling and validating service choreography with semantic interfaces and goals. In *IEEE International Symposium on Service-Oriented System Engineering, 2008*, pages 73 –78. IEEE Computer Society, 2008.

[Jis04] Dan Laurentiu Jisa. Component based development methods: comparison. In *Proceedings of the 5th international conference on Computer systems and technologies*, pages 1–6, New York, NY, USA, 2004. ACM.

[KB10] Surya Bahadur Kathayat and Rolv Bræk. From flow-global choreography to component types. In *System Analysis and Modeling (SAM)*, volume 6598 of *Lecture Notes in Computer Science*. Springer, 2010.

[KH07] Frank Alexander Kraemer and Peter Herrmann. Transforming collaborative service specifications into efficiently executable state machines. In Karsten Ehring and Holger Giese, editors, *Proceedings of the 6th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007)*, volume 7 of *Electronic Communications of the EASST*. EASST, 2007.

[KH10] Frank Kraemer and Peter Herrmann. Reactive semantics for distributed uml activities. In John Hatcliff and Elena Zucca, editors, *Formal Techniques for Distributed Systems*, volume 6117 of *Lecture Notes in Computer Science*, pages 17–31. Springer Berlin / Heidelberg, 2010.

[KKB11] Frank Alexander Kraemer, Surya Bahadur Kathayat, and Rolv Bræk. Unified modeling of service logic with user interfaces. *International Journal of Cooperative Information Systems (IJ-CIS)*, 20 (2):177–200, 2011.

[KLB10] Surya Bahadur Kathayat, Hien Nam Le, and Rolv Bræk. Automatic derivation of components from choreographies - a case study. In *International conference on Software Engineering*, Phuket, Thailand, 2010.

[KLB11] S. B. Kathayat, H. N. Le, and R. Bræk. A model-driven framework for component-based development. In *Proc. of the 15th International Conference on System Design Languages of the SDL Forum Society*. Springer, 2011.

[KLB12] Surya Bahadur Kathayat, Hien Nam Le, and Rolv Bræk. *Handbook on Reserach of E-Business Standards and Protocols: Documents, Data and Advanced Web Technolgies.*, chapter Collaboration-based Model-Driven Approach for Business Service Composition., pages 1–21. IGI Global, 2012.

[KP06] Raman Kazhamiakin and Marco Pistore. Choreography conformance analysis: Asynchronous communications and information alignment. In *Proceedings of the 3rd International Workshop on Web Services and Formal Methods.*, volume 4184 of *LNCS*, pages 227–241. Springer, 2006.

[KSH09a] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann. Model-driven construction of embedded applications based on reusable building blocks: an example. In *Proceedings of the 14th international SDL conference on Design for motes and mobiles*, SDL'09, pages 1–18, Berlin, Heidelberg, 2009. Springer-Verlag.

[KSH09b] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann. Tool support for the rapid composition, analysis and implementation of reactive services. *Journal of Systems and Software*, 82(12):2068–2080, 2009.

[Lau06] Kung-Kiu Lau. Software component models. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 1081–1082, New York, NY, USA, 2006. ACM.

[MH05] Jan Mendling and Michael Hafner. From inter-organizational workflows to process execution: Generating bpel from ws-cdl. In *OTM 2005, LNCS 3762*, pages 506–515. Springer, 2005.

[OCK$^+$11] Jeyeon Oh, Nam Wook Cho, Hoontae Kim, Yunhong Min, and Suk-Ho Kang. Dynamic execution planning for reliable collaborative business processes. *Inf. Sci.*, 181:351–361, January 2011.

[OMG09] OMG. Uml 2.2 superstructure specification, February 2009.

[PBA$^+$08] Daniela C. C. Peixoto1, Vitor A. Batista1, Ana P. Atayde1, Eduardo P. Borges1, Rodolfo F. Resende, and Clarindo Padua. A comparision of bpmn and uml 2.0 activity diagrams. In *Proceedings of the VII Simposia Brasileiro de Qualidade de Software*, 2008.

[PTDL08] Michael P. Papazoglou, Paolo Traverso, Scharam Dustdar, and Frank Leymann. Service-oriented computing: A research roadmap. *International Journal of Cooperative Information Systems*, 17(02):223+, 2008.

[QZCY07] Zongyan Qiu, Xiangpeng Zhao, Chao Cai, and Hongli Yang. Towards the theoretical foundation of choreography. In *Proceedings of the 16th international conference on World Wide Web*, pages 973–982. ACM, 2007.

[SB09] Gwen Salaün and Tevfik Bultan. Realizability of choreographies using process algebra encodings. In *Proceedings of the 7th International Conference on Integrated Formal Methods*, IFM '09, pages 167–182, Berlin, Heidelberg, 2009. Springer-Verlag.

[vdAHW03] W.M.P. van der Aalst, A. H. M. Ter Hofstede, and M. Weske. Business process management: A survey. In *Proceedings of* *the 1st International Conference on Business Process Management, volume 2678 of LNCS*, pages 1–12. Springer-Verlag, 2003.

Chapter 14

Paper 9

Title: Analyzing Realizability of Choreographies Using Initiating and Responding Flows.

In: ACM 2011 Conference on Model-Driven Engineering, Verification, and Validation (MoDeVVA). October 17, 2011. Wellington, New Zealand.

Publisher: ACM.

Analyzing Realizability of Choreographies Using Initiating and Responding Flows.

Surya Bahadur Kathayat, and Rolv Bræk
Norwegian University of Science and Technology (NTNU)
Department of Telematics
Trondheim, Norway
Email:{surya, rolv.braek}@item.ntnu.no

Abstract

Choreographies are used to define and analyze the global collaborative behavior of reactive systems while orchestration are used to define local behavior of components for realization. A number of realizability problems need to be addressed when going from a global choreography to distributed realization. This paper focuses on the analysis of choreographies in order to detect and resolve such realizability problems. UML activity diagram are used for choreography specification as well as for distributed realizations using a distinction between *initiating* flows local to a component and *responding* flows that represent flows external to a component. These concepts provide a *new, simpler, and computationally efficient* way to detect and to some extent resolve most known realizability problems.

keywords: Choreography, Realizability, Collaborative Services.

14.1 Introduction

The work presented here considers the relationship between global service behavior models (called *choreography models* in the following) and their realization in terms of distributed components behaviors (called *orchestrations* in the following). It is desirable that service engineers (or domain experts who are not service developers or programmers) can work mainly on the level of choreography, focusing on the intended collaborative behavior rather than on the detailed orchestration. A choreography should therefore fully define the global behavior so that component orchestrations may be

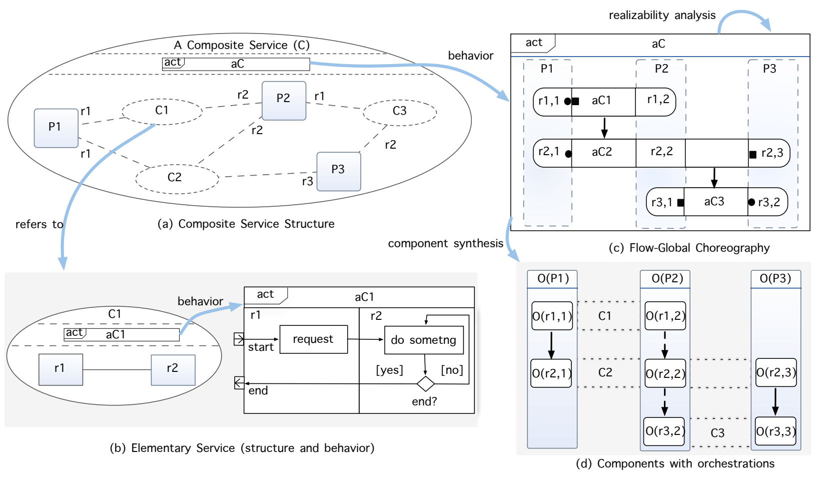


Figure 14.1: Overall Approach

automatically synthesized thereby enhancing the correctness of the realization. The overall approach addressed in this paper illustrated in Fig. 14.1, seeks to enable this.

Most approaches to choreography assume that basic activities are allocated to one component. However in many cases the basic building blocks are actually collaborations involving several participants. Therefore we allow collaborations as building blocks in choreography. Collaborations encapsulate interactions and allow to define choreography on a higher level of abstraction than interactions. The corresponding orchestrations will therefore be a temporal ordering of role behaviors as illustrated in Fig. 14.1(d).

Our work rests on two key enablers: *(1)* The use of UML 2.x collaborations to structure the service dimension and to identify the roles to be used in composition, as shown in Fig. 14.1(a); *(2)* The use of UML 2.x activity diagrams to completely define and analyze the cross-cutting service behavior i.e. *choreography* illustrated in Fig. 14.1(c) as well as *orchestration* in Fig. 14.1(d).

The *choreography* uses model elements of standard UML 2.x activity diagrams with associated semantics. Semantically these diagrams define behavior as a temporal ordering of actions, where an action either may be local to a role or represent a collaborative behavior involving several roles. In the following it is assumed that the choreography of a composite service C is defined by activity aC as illustrated in Fig. 14.1(c), similarly the choreography of an elementary service C_1 is defined by aC_1 as illustrated in Fig. 14.1(b).

Each role in a choreography is mapped to a corresponding role orchestration that is linked within the orchestration of a component by local *initiating flows* (indicated by solid lines) and *responding flows* (indicated by dotted lines in Fig. 14.1(d)).

When a global choreography is mapped to distributed role orchestrations and any of the role orchestrations need to handle behavior that has not been specified for the corresponding role in the choreography specification, this is considered a *realizability problem*.

In a choreography, the activity flows that cross partition boundaries does not model the possibility that messages may be reordered by the medium, nor that all flows between a pair of components may share a common medium and input buffer at the receiving end, which normally is the case in distributed realizations. In orchestration models we assume an arbitrary transfer delay, and a shared buffer for each component. This models the delays that are inevitable in a distributed realization using an underlying communication medium such as an asynchronous message bus. If this buffer is a FIFO queue, messages will be taken from the queue in strictly the same order as they arrive. If this arrival order is the same as the reception order specified by the activity flows, everything is fine. But it may happen that the arrival order differs from the specified reception order either due to different sending orders or due to reordering by the communication medium, which means that the distributed realization will generate reception orders that differ from the order specified in the choreography. This is the fundamental cause of many realization-problems.

We note here that the same kind of problems may occur when other types of diagrams such as interaction diagrams are used for choreography specification. Such unspecified behavior is sometimes referred to as *implied scenarios*. Implied scenarios and other realizability problems have been extensively studied in numerous publications (c.f. Sect. 14.5). In this paper we use a novel approach to the identification of such problems and how they may be resolved based on flow analysis of the choreography models using the concepts of initiating and responding flows. This enables a simple, efficient, and general detection mechanism that is easily integrated with the orchestration synthesis.

The rest of the paper is organized as following. Direct realization and realizability problems are discussed in Sect. 14.2. Realizability of control and object flows, and control nodes and paths is discussed in Sect. 14.3 and Sec. 14.4. Related works and conclusion is given in Sect. 14.5.

14.2 Realization and Realizability Problems

14.2.1 Notational Conventions

In order to simplify and clarity the presentation, the following notations and conventions will be used in the following:

- The composite service structure is represented as a UML collaboration where participating entities are represented as *roles*. Sub-services in a composite service are represented as *collaboration uses*. We use the term *role uses* for the roles of collaboration uses in order to distinguish them from the roles of a composite collaboration. The roles of a composite collaboration are in turn considered as role uses in the following composition steps when uses of the composite collaboration are composed.

- Each collaboration use C_i has a structure of *role uses* $r_{i,j}$. The subscripts of the *role uses* $r_{i,j}$ are used to identify collaboration uses in dimension i and roles (parts) in dimension j. For example in Fig. 14.1(c) and Fig. 14.1(d), the role use $r_{i,j}$ represents the role in C_i that is played by part P_j.

- A collaboration use C_i will have role uses $r_{i,1}, r_{i,2}, \ldots r_{i,n}$ and a choreography defined by aC_i. These role uses will be bound to roles $P_1, P_2, \ldots P_n$ of the enclosing collaboration and finally to the components of a system. In a distributed realization the behavior specified for each role use must be realized by a local *role use orchestration* denoted $O(r_{i,j})$. A *role orchestration* $O(P_j)$ is composed from the role use orchestrations $O(r_{i,j})$ bound to it.

- In the choreography for a collaboration C, the action aC_i refers to the behavior of collaboration use C_i. The action[1] has partitions (indicated by solid lines) representing the roles, as shown in Fig. 14.1(c). This notation was first proposed in a slightly different form in [CBvB07] and in its current form in [KB10]. The initiating and terminating roles are indicated using dots and squares attached to the partition. This

[1]An action can have different types of pins: *input pins*, represented by an unfilled pin with incoming arrow and the flows ending on input pins are called *initiating flows*; *output pins*, represented by an unfilled pin with outgoing arrow, will terminate the called activity; *streaming pins*, represented by filled incoming and outgoing pins, can pass tokens while the called activity is active.

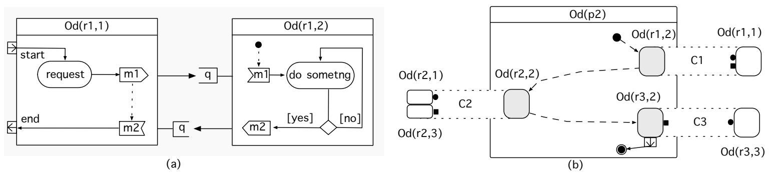


Figure 14.2: Role Orchestrations with Responding Flows

information is not part of UML, but needed for analysis on the level of choreography and can be provided by additional profiling [HKLB11].

14.2.2 Direct Realization

A direct realization of an elementary service choreography aC_i with partitions $r_{i,1}, r_{i,2}, \ldots r_{i,n}$ is obtained by directly projecting the aC_i to local orchestrations $Od(r_{i,j})$ for each partition in the following way:

- insert a send event action on all outgoing flows towards a partition boundary,
- insert a receive event action on all incoming flows from a partition boundary,
- replace all actions not performed by partition $r_{i,j}$ by no-operation,
- mark all flows not local to $r_{i,j}$ as *responding flows.* In Fig. 14.2(a), the responding flow from m_1 to m_2 specifies that the component $Od(r_{1,1})$ must be ready to the receive message $m2$ after the message $m1$ is sent.
- mark all control nodes (decision, merge, fork, join) not local to $r_{i,j}$ as responding nodes

Using projection to derive orchestration is a common technique used by several authors [QZCY07]. As far as we know, the idea to classify external flows as responding flows, and to use them for realizability analysis is novel in our approach. Responding flows are necessary in the orchestrations to ensure that a responding role is enabled and ready to participate in a collaboration activity whenever the collaboration activity may be started.

The orchestration for choreography aC_1 from Fig. 14.1, is two elementary role orchestrations that are connected by an asynchronous communication medium and an input buffer for each role as shown in Fig. 14.2(a). Observe that in $Od(r_{1,2})$ there is a responding flow to enable the orchestration and the receive event action $m1$. In $Od(r_{1,1})$, there is a responding flow from sending event m_1 to receive event m_2. Note that these orchestrations have inter-role interfaces represented by send and receive actions, and intra-role interfaces represented by pins.

A direct realization of a composite service choreography aC defined over *CallBehaviorActions* $aC_{i(i=1,...n)}$ is then derived in the same general way, but now the local orchestration $Od(P_j)$ of a role P_j will include actions representing local orchestrations $Od(r_{i,j})$, such as represented in Fig. 14.2(b) for role orchestration P_2 in Fig. 14.1.

We now define direct realizability of a global choreography aC_i in terms of a relation between the behavior of roles $r_{i,j}$ in aC_i and the behavior of directly realized orchestrations $Od(r_{i,j})$ in a direct realization $Rd(aC_i) = Od(r_{i,1}) \parallel Od(r_{i,2}) \parallel \ldots \parallel Od(r_{i,n})$.

Definition 1: *Direct realizability:* If the temporal ordering of actions in $r_{i,j}$ in aC_i is exactly the same as the possible temporal ordering of actions in $Od(r_{i,j})$ when composed in $Rd(aC_i)$ ignoring the inserted send and receive actions, we say that $r_{i,j}$ is directly realizable in aC_i. If all roles in aC_i are directly realizable, we say that aC_i is directly realizable.

Some authors [KP06, BGG$^+$06, QZCY07] define realizability in terms of trace equivalence or global state space equivalence. We have chosen to define it as a relationship between roles and composition of role orchestrations in order to allow for the additional orderings caused by looser coupling as long as the result is stuck-free and achieves the same effects.

We assume now that each collaboration use in the global choreography is internally correct and *stuck-free*, and demand that the composition of role orchestrations is stuck-free too. The notion of being stuck-free eliminates the following design errors: *1. deadlocks* - a component waits for the message in non-final state that will never be sent, *2. unspecified receptions* - a component receives a message for which there is no transition to consume it in its current state, and *3. orphan messages* - a special kind of unspecified reception where the execution context for which this message was intended does not exist anymore.

It follows from this that if aC_i is stuck-free and directly realizable then $Rd(aC_i)$ is stuck-free.

This can now be extended to the next level of composition in a straight forward way. We assume that each activity aC_i defining the choreography of an elementary collaboration C_i is directly realizable and consider the activity aC defining the choreography of the composite collaboration C. Then if the action ordering of P_j in aC is exactly the same as the action ordering possible in $Od(P_j)$ when composed in $Rd(aC)$ ignoring the inserted send and receive actions, we say that P_j is directly realizable in aC. If all roles in aC are directly realizable, we say that aC is directly realizable.

It follows from this that if aC is stuck-free and directly realizable, then $Rd(aC)$ is stuck-free.

14.2.3 Realization Problems

Realization problems occur because of additional action orderings introduced or allowed by the looser coupling in the orchestration compared to the choreography. This is not problematic as long as all the action orderings specified in the choreography can be performed (ignoring the queue actions) and no role orchestration $O(P_j)$ need to handle behavior that has not been specified for the corresponding role P_j in aC.

One necessary condition for direct realizability is that the internal ordering of each role orchestration $Od(r_{i,j})$ can remain unchanged in the role orchestrations $Od(P_j)$. There shall be no side-effects of the composition that will require that one $Od(r_{i,j})$ must be modified to handle interactions belonging to another $Od(r_{k,j})$. This accords with sound principles of modularity and compositionality. We say that each $Od(r_{i,j})$ keeps its *integrity* under the composition. As we shall see later, it may happen that the integrity of some $Od(r_{i,j})$ is violated by a global flow, which means the global flow is not directly realizable. This does not necessarily mean it cannot be realized, it only means that a realization must provide a solution, normally by means of some additional behavior.

Another necessary condition is that no additional interactions are needed in the orchestration. This means that it must be possible to localize pins and control nodes, such as decisions, forks and joins, without adding interactions. It also means that interruptible activity regions involving more than one role cannot be directly realized.

Rather than generating the global behaviors of $Rd(aC)$ and aC and comparing them role-wise to check if aC is directly realizable we shall seek to identify necessary and sufficient conditions for direct realizability that can be determined by analyzing initiating and responding flows directly. These results builds on earlier work in [CBvB07], but enables different analysis technique and covers more cases than in [CBvB07].

It is important to note that whether a choreography is directly realizable or not depends not only on the ordering defined by the choreography, but also on the characteristics of the underlying communication channel and also the type of input buffering at each component. We consider the following types of communication channels: (a) *out-of-order delivery channel* where where message sent from a source to a given destination may be received in different order than they were sent, (b) *in-order delivery* channel where messages are received in the same order they were sent. In both cases, it is assumed that there is no message loss. Regarding the input buffering at each component, we distinguish three cases: *1. no-reordering* - where each component has a single FIFO buffer, *2. reordering between sources* - where each component has a separate FIFO buffer for each source and locally determine from which source the next message should be consumed, *3. full reordering* - where the component may reorder the messages freely.

14.3 Realizability of Control and Object Flows

We first consider just one sequential composition step, written: $aC_1; aC_2$. We then discuss the realizability of sequential flows.

In $aC_1; aC_2$, according to the activity diagram semantics, aC_1 shall be completely finished before aC_2 is started. In a distributed realization the global ordering must be provided by means of local flows. In the following we discuss the different causality properties that local flows can provide. This classification based on [CBvB07].

Localized causality: This causality is ensured when the role P_j that terminates aC_1 is the role that also initiates aC_2. Localized causality has the property that it satisfies the *strong* sequencing specified in the activity diagram for the global choreography. We mark the initiating flow in this case as {*strong*} to indicate that the strong sequencing property holds. All roles P_k that play a responding role $r_{2,k}$ in aC_2 and also participate in aC_1 with a role $r_{1,k}$ will have a local responding flow linking $r_{1,k}$ to $r_{2,k}$. These responding flows are also marked as {*strong*}.

We note that no overlap is possible between two roles directly linked by {strong} flows. Strong flows imply that the linked role orchestrations keep their integrity and therefore a {*strong*} flow from aC_1 to aC_2 is directly realizable. An example of localized causality is the flow between the aC_1 and aC_2 collaborations in Fig. 14.3(a). Localized causality is ensured in this case when the flow is localized to the P_1 role.

No causality: If the role that initiates aC_2 is not participating in aC_1, the sequential ordering specified in the global choreography cannot be realized, since there is no way to ensure a sequential ordering between aC_1 and aC_2. The global flow as well as the local initiating flow to aC_2 is marked as {*non-causal*} in this case. This means that the roles $r_{1,j}$ and $r_{2,j}$ of any role P_j participating in both aC_1 and aC_2 will be linked by {*non-causal*} responding flows and may overlap arbitrarily.

A {*non-causal*} flow between role orchestrations implies a possible overlap that may violate the integrity of both role orchestrations, and therefore

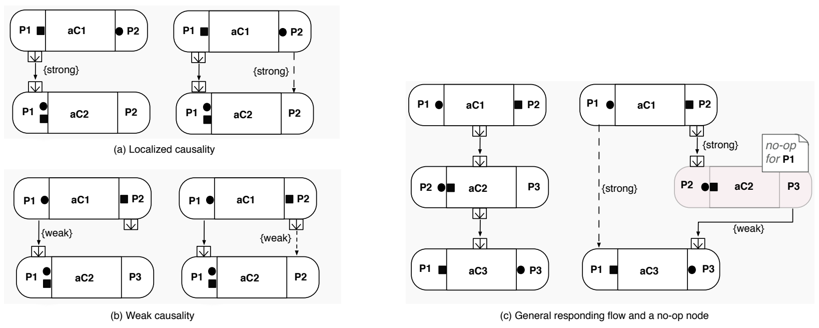


Figure 14.3: Causality Properties and Responding flows examples

is not directly realizable.

If it is the intention that the events in aC_1 and aC_2 should be arbitrarily interleaved, the solution is to change the global choreography to use a parallel fork instead of using a sequence between aC_1 and aC_2. If sequential ordering is the intention, one needs to add at least one interaction to ensure at least weak causality (see below) or to change either aC_1 or aC_2 or both so that the flow at least ensures weak causality.

Weak causality: Weak causality means that the role P_j playing the initiating role $r_{2,j}$ in aC_2 participates with a non-terminating role $r_{1,j}$ in aC_1. This means that $r_{1,j}$ is completely finished when $r_{2,j}$ and hence aC_2 starts, but other roles participating in aC_1 need not be finished. This corresponds to the weak sequencing normally assumed in interaction diagrams, but deviates from the strong sequencing of UML activity diagrams. There are two possible options:

- *Enforced strong sequence.* This means to add at least one interaction from the terminating role $r_{1,i}$ of aC_1 to the initiating role $r_{2,j}$ of aC_2, or to modify either aC_1 or aC_2 to include this interaction. This will make the composition directly realizable, but it will introduce the additional message overhead. This strategy has been used in earlier work on protocol synthesis, such as [KHB96].

- *Weak sequencing.* This means to allow partial overlap between events in aC_1 and aC_2. As long as this does not harm the integrity of any roles, it may be fully acceptable and considered as a direct realization option. In this case the initiating flow and the corresponding responding flows are marked $\{weak\}$. An example can be seen in Fig. 14.3(b) between aC_1 and aC_2.

If some role P_k participates with a role $r_{1,k}$ in aC_1 and a responding role $r_{2,k}$ in aC_2, there will be a $\{weak\}$ responding flow from $r_{1,k}$ to $r_{2,k}$ in P_k.

This means that messages may be sent to $r_{2,k}$ (by another role in aC_2) before $r_{1,k}$ is completely finished. If there is any possibility that these messages may be received before $r_{1,k}$ is completely finished, they may interleave with remaining events in $r_{1,k}$, causing unspecified ordering between send/receive events in $r_{1,k}$ and $r_{2,k}$, and damage the integrity of $r_{1,k}$ and $r_{2,k}$. *Note that apart from {non-casual} flows role integrity can be damaged only between roles linked by a {weak} responding flow.* We also note that the amount of possible overlap depends on the underlying communication medium. The amount of overlap is determined by how much of $r_{1,k}$ may remain when $r_{2,k}$ is started, called the *remainder* of $r_{1,k}$ and how much of $r_{2,k}$ that may execute before $r_{1,k}$ is finished, called the *beginning* of $r_{2,k}$.

Note that in the special case when the composed collaborations are two-party (binary), with roles P_1 and P_2 and there is a {*weak*} responding flow from $r_{1,2}$ in aC_1 to $r_{2,2}$ in aC_2, the remainder of $r_{1,2}$ may only be message receptions, and the beginning of $r_{2,2}$ may only be message reception and so if no message reordering is possible on the link between P_1 and P_2 there will be no problem.

If no responding roles in aC_2 is played by a role that also participates in aC_1, there will be no responding flows local to a role that participates in both aC_1 and aC_2, and therefore no overlaps are possible between roles in aC_1 and aC_2. There may however be indirect {weak} flows as will be explained later in this section.

Send causality: This is a special case of {*weak*} causality where there is a total ordering of sending events. The initiating flow of aC_2 is local to a role that either terminates aC_1 (i.e., strong sequencing) or sends the last message in aC_1. If aC_1 is internally send causal, only message reception on a terminating role may remain in aC_1 when aC_2 is started. A possible role overlap will show up as a {*weak*} responding flow local to a role playing a terminating role in aC_1 and a non-initiating role in aC_2. If message reordering is possible between the last message this role receives in aC_1 and the first messages it receives in aC_2, unspecified reception orders may result. If these messages are exchanged between the same two components over a FIFO medium, there will be no problems.

In general, one must consider sequential flows consisting of several sequential composition steps aC_1; aC_2; ...; aC_n and consider the causality of each initiating and responding flow. It is now possible that a role in aC_i may overlap with a responding role in aC_j (i<j) if there is an indirect {*weak*} responding flow within a role k from a role $r_{i,k}$ in aC_i to a role $r_{j,k}$ in aC_j. Indirect responding flows occur when there may be one or more collaborations or local actions in which a component does not participate

between collaboration actions where it participates. When a component is not participating in a collaboration action aC_i, there will be a no-op action (an action where a role does not participate) in the responding flow. For example a component P_1 is not participating in aC_2 therefore aC_2 is a no-op action in a responding flow from aC_1 to aC_3 for component P_1 in Fig. 14.3(c).

We now generalize responding flows to include no-op actions: An *indirect responding flow* is a responding flow with one or more no-op actions, a *general responding flow* is a responding flow with zero or more no-op actions. The first step of a general responding flow determines whether the flow as a whole has property {*non-causal*}, {*weak*} or {*strong*}. A general responding flow that starts with a {*non-causal*} step remains {*non-causal*}. A general responding flow that starts with a {*strong*} step remains strong regardless how many {*weak*} or {*strong*} (no-op) steps that follows. A general responding flow that starts with a {*weak*} step remains {*weak*}, regardless how many {*strong*} or {*weak*} (no-op) steps that follows.

We conclude from the above that {*non-causal*} ordering can never be directly realized, and that {strong} ordering always can be directly realized. When using {*weak*} ordering there may be problems. The reasoning above justifies the following propositions:

Proposition 1 (realizability of general weak responding flow): A general {*weak*} responding flow between roles $r_{i,j}$ and $r_{k,j}$ is directly realizable only iff the remainder of $r_{i,j}$ and beginning of $r_{k,j}$ can be ordered by the underlying communication medium.

Proposition 2 (realizability of sequential composition): A sequential composition aC_1; aC_2; ...; aC_n is directly realizable iff no initiating flows are {*non-causal*} and all general {*weak*} responding flows resulting from a direct realization are directly realizable.

Proofs are omitted due to space limitation.

We note that it is possible to identify the presence of {*non-causal*} initiating flows and {*weak*} responding flows directly in the global choreography and thereby pinpoint where realizability problems may occur. One may then investigate each {*weak*} responding flow separately to see if any overlap (message interleaving) is possible.

In the following circumstances the ordering between events in the remainder $r_{i,j}$ and beginning $r_{k,j}$ can be ensured: When the remainder of $r_{i,j}$ and the beginning of $r_{k,j}$ are message receptions and local actions only, and the medium globally conserves the sending order, or when the remainder of $r_{i,j}$ and the beginning of $r_{k,j}$ are message receptions and local actions only and the remaining messages are sent by the same part, and the communication link between the two roles locally conserve the sending order (is FIFO) or by reordering messages for consumption such that the remainder of $r_{i,j}$ is always processed before the beginning of $r_{k,j}$. This may require that messages are tagged with the role invocations they belong to.

We note that the above propositions will apply also in the case where elementary collaborations are specified using sequence diagrams. All the realization problems and conditions associated with sequential composition identified in previous work [CBvB07] are covered and generalized by the propositions and conditions given above. The concept of responding flows needed in orchestrations therefore provides a new, simpler and more general indicator of these problems. In the following section, we consider the realizability analysis of control nodes on a flow-path.

14.4 Realizability of Control Nodes and Paths

When control nodes are present, two action nodes may be linked by a *flow-path* through a number of intermediate control nodes linked by direct *flow-edges*. Each intermediate flow-edge and control node in a choreography may be part of several flow-paths. For each flow-path linking a pair of collaboration actions aC_1 and aC_2 (or local actions) one can determine which causality property that holds, i.e. {*non-causal*}, {*weak*}, {*strong*}. The propositions and conditions given in Sect. 14.3 apply to each such flow-path. In order to be directly realizable, as explained in Sect. 14.3, it is necessary that every initiating flow-path linking two collaboration actions aC_1 and aC_2 (or local actions) is either {*strong*} or {*weak*} and local to the role initiating aC_2. (Enforcing strong sequencing by adding interactions is not considered as direct realization due to the interactions that are added). Initiating paths that can not be localized this way are considered as {*non-causal*} and hence, as not directly realizable as explained in Sect. 14.3.

For each {*weak*} flow-path, one may derive responding paths and check for realizability in a similar way as explained in Sect. 14.3. Note that control nodes that are external to a role are represented as responding control nodes on the responding flows of that role.

Proposition 3 (flow-path localization): An initiating flow-path from collaboration action aC_1 to collaboration action aC_2 is directly realizable iff the entire path with all intermediate control nodes can be localized to the role initiating aC_2 and the path is either {*strong*} or is {*weak*} with all its responding flow-paths realizable according to *Proposition 1.*

Each control node can be part of several paths. In order to be directly realizable all initiating flow-paths through a node should be local to the

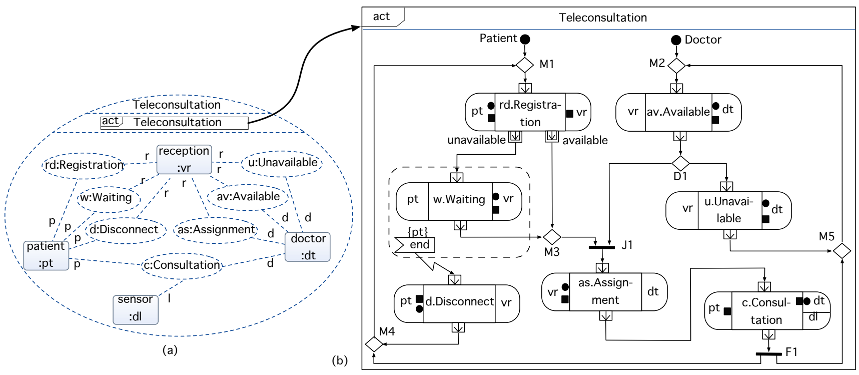


Figure 14.4: Teleconsultation Service Example

same role. If this is not possible, additional interaction flows will be needed, which means the flows through the nodes are not directly realizable.

Proposition 4 (control node localization): If all initiating flow-paths through a control node is directly realizable and the node is localized to one role then the node is directly realizable.

Whenever the paths through a choice node need to be local to different roles in order to satisfy causality, we have a case of *non-local* choice [Kra08, BF04]. Non-local choices have been extensively studied in the literature, and are normally not directly realizable. It is only when a non-local decision can be performed locally by each component based on local information, in a way that guarantee that the components choose corresponding branches that non-local choice is directly realizable. Some resolution techniques are presented in [KB10].

Let us now consider the *Teleconsultation* service, first described in [CBvB07], and its choreography specification shown in Fig. 14.4. In Fig. 14.4, *pt*, *dt*, *dl*, and *vr* represent patient, doctor, data logger, and virtual receptionist respectively. Will it be directly realizable or not? We assume that we are not applying enforced strong sequencing, but try to localize each path to the *role use* initiating the next *collaboration use*. In other words we explore the natural ordering properties of the choreography. We use the following notation for paths in the following: *path ::= roleId→causalityClass→roleId*, where
roleId ::= collaborationUseId.roleUseId. We first identify all flow paths and their associated causalities, and then consider all flow paths through each node. We note that a node may be part of several flow-paths, each having different properties, e.g:

- M1 is part of the following flow paths:

d.pt→*strong*→rd.pt; c.pt→ *strong* →rd.pt. In this case all initiating flow-paths through the merge M1 may be localized to *pt* and therefore M1 be directly realizable. Similarly M2, . . . M4 are directly realizable.

- J1 is part of the following flow paths:
  w.vr→*strong*→as.vr; rd.vr→*weak*→as.vr; and
  av.vr→*weak*→ as.vr. All initiating flows through J1 may be localized to *vr*. The paths w.vr→*strong*→as.vr; rd.vr→*weak*→as.vr are directly realizable. There will be no problem in av.vr→*weak*→as.vr if the communication between *vr* and *dt* conserves the sending order.

- F1 is part of the following flow paths:
  c.pt→*strong*→rd.pt; c.dt→*strong*→av.dt. These paths are local to two roles, *dt* and role *pt*, and therefore the fork F1 is not directly realizable. However, in this case, this flow can be replaced by two local flows, one within each terminating role, to achieve the same effect.

- D1 is a part of the following flow paths:
  av.vr→*weak*→as.vr; av.dt→*strong*→u.dt. These paths are local to two roles, *dt* and role *vr*, and therefore the decision D1 represents a non-local choice and is not directly realizable. If the decision node is localized to the {*strong*} path i.e. at *dt*, there will also be a {*weak*} responding flow through D1 at *dt* indicating a realizability problem.

Responding flows demand special consideration in the presence of choices. The decision making should be taken on the initiating flows, but the decisions must be reflected in responding flows allowing the responding role to choose among two or more collaborations based on the first messages received in them. This is called choice propagation in [CBvB07] and requires that the first messages are unique so that the choice is determined by the first message received. If this is not the case, the choice propagation is said to be ambiguous according to [CBvB07]. {*weak*} responding flows through responding decision nodes may cause choice propagation to become ambiguous because of message reordering, called race propagation in [CBvB07]. This may lead to the wrong choice being taken.

By introducing choices and merges, we have also introduced the possibility to make loops with repeated actions linked by {*strong*} or {*weak*} flows (as illustrated in Fig. 14.4). Since each iteration of a loop involves the same set of components, {*weak*} flows will cause problems unless the communication medium conserves the sending order. As a method guideline, {*strong*} sequencing should be preferred between loops iterations to be sure that one iteration is completely finished before the next one starts. Loops may also cause so-called process divergence, characterized by a role sending an unbounded number of messages ahead of the receiving role. This may happen where iterations of a loop is weakly sequenced and the loop body contains uni-directional interactions between a pair of roles. This may also give rise to so-called orphan messages, i.e. messages sent in one iteration and received in a later iteration.

Note that a tool may obtain all the diagnostic information discussed above directly from the choreography without needing to look into the behavior detail of any of the composed collaborations.

14.5 Related Work and Conclusion

There is a large body of work addressing realizability of choreography defined using MSC, UML sequence diagrams, interaction overview diagrams, use case maps, UML activity diagrams, BPEL, WS-CDL and related notations. It is interesting to observe that similar realization problems manifest themselves regardless of which particular notation is used to define choreography. This suggests that the core of the problem is a fundamental difference in the nature of global choreography v.s. distributed orchestration, rather than notation. To our best knowledge there is no other work that uses a flow analysis similar to the one presented here, to systematically deal with realization problems.

The realizability of specifications of reactive systems, in general terms, was studied in [ALW89]. In the context of interaction diagrams, the notion of realizability has been related to the notion of implied scenario, first in [AHP96], where the authors propose two notions of realizability, depending on whether the realization is required to be deadlock-free (safe realizability) or not (weak realizability). This work was later extended in [AEY01] to consider realizability of bounded HMSCs. In [MFEH07] the authors discuss the notion of safe realizability and causes of implied scenarios for MSC based Specifications. [BBJ$^+$05] discusses automatic detection and resolution of semantic errors such as blocking, non-local pathologies, non-local ordering, and false-underspecification associated with scenario based requirements captured in the form of UML/MSC sequence diagrams.

In [CBvB07] the authors provide a classification of realizability problems and give some criteria for detecting them at the level of choreographies. They use an earlier version of the activity diagrams (AD) presented in this paper to define and analyze global choreography. However in [CBvB07], sequence diagrams (SD) are used to specify elementary collaborations. Compared to their work, our work, *1.* use AD throughout, *2.* allow streaming and interrupting flows (not discussed in this paper), *3.* introduce the concept of responding flows and role integrity that simplifies and generalizes realizability analysis. *4.* defines orchestrations as modules for subsequent composition into systems. Compared to our previous work in [KB10], we have here elaborated on the responding flows for realizability analysis.

The SPACE method supported by the Arctis tool enables specification and model checking of collaborative behavior by composing collaborative building blocks together using a *swim-lane* like notation for UML activity diagrams [Kra08]. In the work presented here we use a more abstract choreography model [KB10] to define the complete behavior. Such models can be translated into Arctis equivalent (or similar) models for further design synthesis and code generation.

Most of the work in the business and web-services domain describe choreography in terms of interactions [ZBDtH06, MH05, Dec09]. Some use UML communication or interaction overview diagrams [SB09]. Some use the swim-lane notations of UML activity diagrams to define choreographies. Interaction diagrams require that interactions are named, while activity diagrams represent the purpose of interactions as flows, and therefore are more high level and abstract. Moreover the concept of collaboration enable complex interaction behavior to be separately modeled, encapsulated and reused as building blocks in their own right. Thus, using our approach the resulting choreography tends to be on a higher level, closer to the problem domain, than choreography expressed in terms of detailed interactions.

The semantics of choreography is represented as labelled transition system (LTS) in [KP06], set of conversations in [BGG$^+$06] or activity traces in [QZCY07]. We use UML activity diagrams and their semantics [KH10]. This enable us to define complete behavior that can be directly realized, unlike the partial or fragment scenarios normally provided by sequence diagrams. There are different techniques to check the realizability between choreography and orchestration. In [SB09] behavioral equivalence between the LTS of a choreography and a the LTS of a parallel composition of orchestrations is used. [KP06, BGG$^+$06] uses the bi-simulation for this. Trace equivalence is used by the authors in [QZCY07]. In our work we do not explore global state spaces, but provide rules for analyzing the choreography directly.

In conclusion, we have here presented a constructive approach to derive orchestrations from choreography that also provides means to analyze realizability using the new concepts of initiating and responding flows. The approach provides the following benefits:

- **Automation and cost-reduction**: initiating and responding flows are derived as part of normal component realization and does not require additional effort.
- **Efficient**: checking realizability of choreographies by analyzing flows avoids the exponential growth of state-space involved in traditional state-space analysis approach.
- **Simplification**: realizability problems can be spotted simply by checking initiating and responding flow-paths and their causality properties. There is no requirements that choreography graphs be well-structured.
- **Correctness by construction**: correctness of the choreographies can be checked early during requirement specificatoin, in many cases without going into the detailed behavior of collaborations.
- **Modularity**: ensuring integrity of the role orchestrations provides modules suitable for later reuse and composition.

Acknowledgement

The authors would like to thank Humberto Castejón and Gregor von Bochmann for many interesting and fruitful discussion leading up to the ideas presented here.

References

[AEY01] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Realizability and verification of msc graphs. In *Proc. 28th Int. Colloq. on Automata, Languages, and Programming*, pages 797–808. Springer, 2001.

[AHP96] Rajeev Alur, Gerard Holzmann, and Doron Peled. An analyzer for message sequence charts. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 35–48. Springer Berlin / Heidelberg, 1996.

[ALW89] Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, pages 1–17, London, UK, 1989. Springer-Verlag.

[BBJ$^+$05] Paul Baker, Paul Bristow, Clive Jervis, David King, Robert Thomson, Bill Mitchell, and Simon Burton. Detecting and resolving semantic pathologies in uml sequence diagrams. In *Proceedings of the 10th European software engineering conference*, ESEC/FSE-13, pages 50–59, New York, NY, USA, 2005. ACM.

[BF04] R. Bræk and J. Floch. Ict convergence: Modeling issues. In *In System Analysis and Modeling (SAM), 4th International SDL and MSC Workshop, pages 237–256, Ottawa, Canada.*, 2004.

[BGG$^+$06] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration conformance for system design. In *COORDINATION, volume 4038 of LNCS*, pages 63–81. Springer, 2006.

[CBvB07] Humberto Nicolás Castejón, Rolv Bræk, and Gregor von Bochmann. Realizability of collaboration-based service specifications. In *Proceedings of the 14th Asia-Pacific Software Engineering Conference*, pages 73–80, Washington, DC, USA, 2007. IEEE Computer Society.

[Dec09] Gero Decker. Realizability of interaction models. In *1st Central-European Workshop on Services and their Composition*, 2009.

[HKLB11] Fenglin Han, Surya Bahadur Kathayat, Hien Nam Le, and Rolv Bræk. Towards choreography model transformation by graph transformation. In *Proc. of 2nd IEEE International Conference on Software Engineering and Service Sciences*, Beijing, China, 2011.

[KB10] Surya Bahadur Kathayat and Rolv Bræk. From flow-global choreography to component types. In *System Analysis and Modeling (SAM)*, volume 6598 of *Lecture Notes in Computer Science*. Springer, 2010.

[KH10] Frank Kraemer and Peter Herrmann. Reactive semantics for distributed uml activities. In John Hatcliff and Elena Zucca, editors, *Formal Techniques for Distributed Systems*, volume 6117 of *Lecture Notes in Computer Science*, pages 17–31. Springer Berlin / Heidelberg, 2010.

[KHB96] Christian Kant, Teruo Higashino, and Gregor V. Bochmann. Deriving protocol specifications from service specifications written in lotos. *Distributed Computing*, 10:29–47, July 1996.

[KP06] Raman Kazhamiakin and Marco Pistore. Choreography conformance analysis: Asynchronous communications and information alignment. In *Proceedings of the 3rd International Workshop on Web Services and Formal Methods.*, volume 4184 of *LNCS*, pages 227–241. Springer, 2006.

[Kra08] Frank Alexander Kraemer. *Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks.* PhD thesis, Norwegian University of Science and Technology, 2008.

[MFEH07] Abdolmajid Mousavi, Behrouz Far, Armin Eberlein, and Behrouz Heidari. Strong safe realizability of message sequence chart specifications. In Farhad Arbab and Marjan Sirjani, editors, *International Symposium on Fundamentals of Software Engineering*, volume 4767 of *Lecture Notes in Computer Science*, pages 334–349. Springer Berlin / Heidelberg, 2007.

[MH05] Jan Mendling and Michael Hafner. From inter-organizational workflows to process execution: Generating bpel from ws-cdl. In *OTM 2005, LNCS 3762*, pages 506–515. Springer, 2005.

[QZCY07] Zongyan Qiu, Xiangpeng Zhao, Chao Cai, and Hongli Yang. Towards the theoretical foundation of choreography. In *Proceedings of the 16th international conference on World Wide Web*, pages 973–982. ACM, 2007.

[SB09] Gwen Salaün and Tevfik Bultan. Realizability of choreographies using process algebra encodings. In *Proceedings of the 7th International Conference on Integrated Formal Methods*, IFM '09, pages 167–182, Berlin, Heidelberg, 2009. Springer-Verlag.

[ZBDtH06] Johannes Zaha, Alistair Barros, Marlon Dumas, and Arthur ter Hofstede. Lets dance: A language for service behavior modeling. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4275 of *Lecture Notes in Computer Science*, pages 145–162. Springer Berlin / Heidelberg, 2006.