
Service Diagrams: Queries and Constraints

Ingolf H. Krueger

Joint work with Fred Doucet and Massimiliano Menarini

Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0404, USA

California Institute for Telecommunications
and Information Technologies
La Jolla, CA 92093-0405, USA



Background and Motivation

- Dramatic increase in distribution and complexity of software systems
 - Business/Enterprise Systems
 - Technical/Embedded Systems
- Shift from stand-alone to networked systems
- Internet/Wireless Networks have become key enabling technologies for advanced services
- Convergence between business and technical systems:
 - Telecommunication/Networking
 - Web Services
 - Embedded Systems

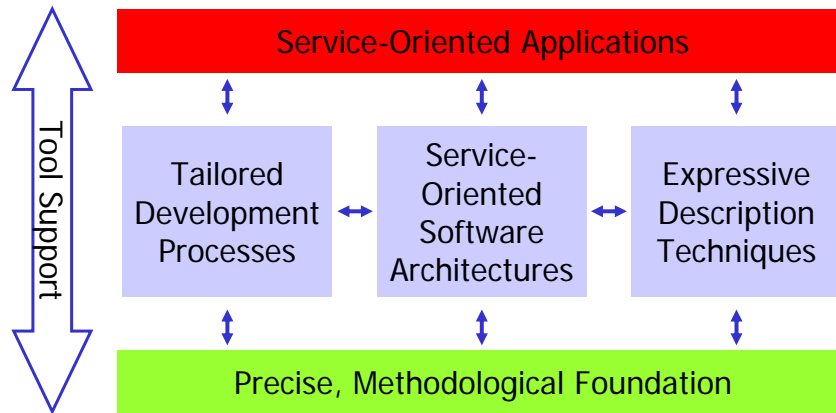
Application Domains

- Enterprise Service Bus (ESB)
- **Automotive Systems Engineering**
- Sensor Networks
- Command and Control
- Grid/Next Generation Internet

Example: Automotive Software

- Highly **complex, networked, and distributed**
- **Separate, integrated networks** for power train, chassis, security, MMI, multimedia, body/comfort functions
- In some vehicles: 70-80 electronic control units (ECUs), supporting **several hundred to more than 1,000 features**
- ECUs delivered by **multiple suppliers, differ** from vehicle line to vehicle line
- Need for **hardware independence**
- **Increasing interaction** between modules of different sub-systems (e.g. navigation system needs information from speed sensor)
- **Increasing interaction** also **beyond** the car's boundaries (with external devices, like PDAs, mobile phones and backend servers,...)

Service Engineering for Distributed, Reactive Systems

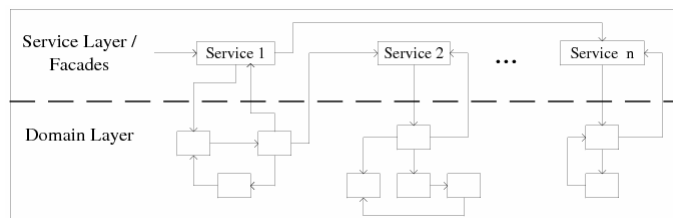


Two Complementary Approaches

- "Traditional" Development
 - Focus on individual components/complete specifications
 - Integration often an after-thought
 - Scattering of functionality
- Service-Oriented Development
 - Complements component-/object-oriented development
 - Focus on the interplay among components
 - Addresses cross-cutting concerns
 - "Services" decouple abstract behavior from implementation architectures
 - "Roles" decouple logical entities from physical components
 - Benefits:
 - Applicable across domains
 - Promotes reusability
 - Inherent loose coupling

Scattering of Functionality

- Logical and physical distribution of functionality
- Problem:
 - Operations cannot be located within specific domain objects
 - Operations cross-cut responsibility of multiple objects
 - Scattering of operations during deployment
- Service-oriented solution:
 - **Services orchestrate the interplay of multiple objects/components**

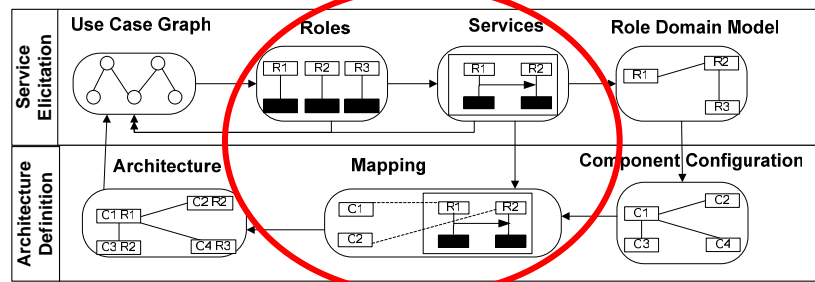


Overview

- Background and Motivation
- **Diagrams for Services**
- What are Roles, anyway?
- Queries and Constraints
- Summary and Outlook

Service-Oriented Development Overview

- Interaction patterns are cross-cutting concerns
 - shift focus from components to their interactions
- Services
 - projection of overall system behavior
 - defined by interactions between participating entities
 - first class elements of modelling and implementation
 - described using extended MSCs



© Ingolf Krueger

UCSD/Calit2 | 9

Example: CTAS System

- Center TRACON Automation System
- Key element: weather information → clients

Communications Manager (CM)

- Keeps track of initialized clients
- Receives weather updates
- Controls update cycle

Weather Clients

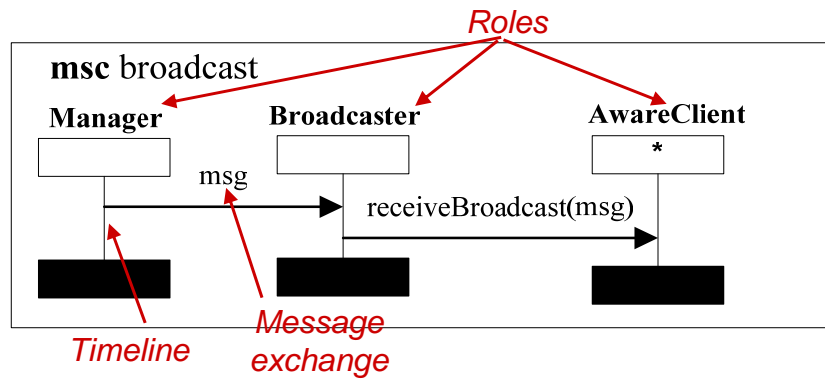
- Aware and Unaware
- Initialize with CM
- Receive update notifications from CM

- Subset of CTAS System (Update Phase)
- If all clients can update successfully then new weather information is used by all clients
- If at least one client cannot update successfully then old weather information is used by all clients

© Ingolf Krueger

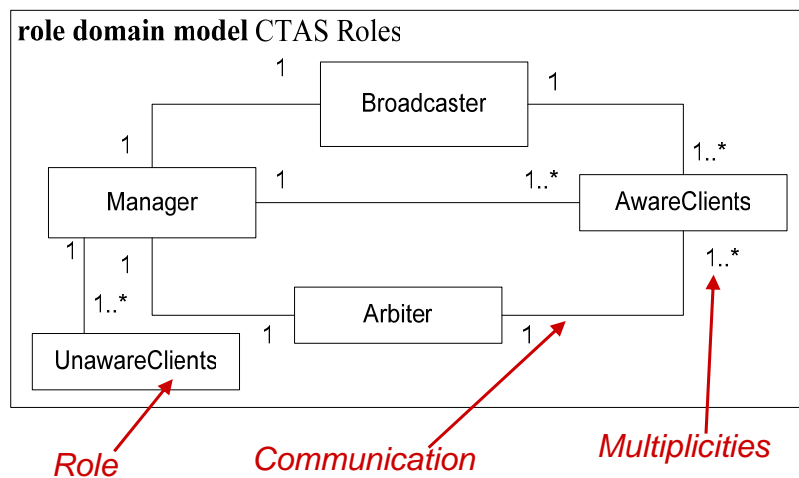
UCSD/Calit2 | 10

Roles in Services

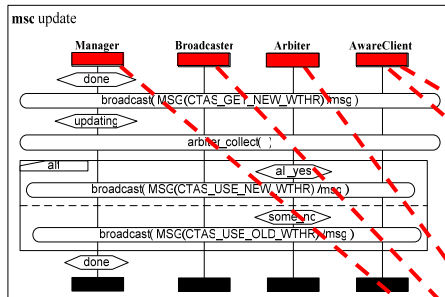


- A role is a *logical entity* :
 - describes contribution of participant in the service
- Roles have states

Example: CTAS Role Domain Model

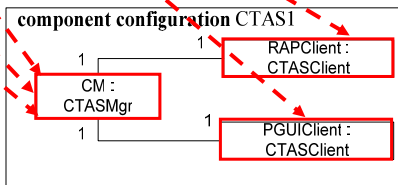


Example: Service Mapping – Deployment 1

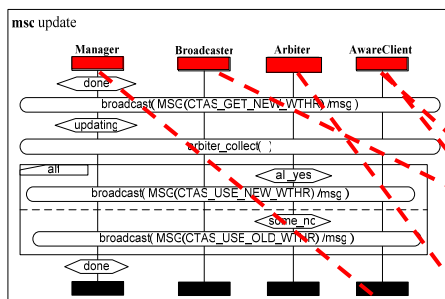


CTASClient plays the role of AwareClient

CTASMgr plays roles of Broadcaster, Manager and Arbiter



Example: Service Mapping – Deployment 2

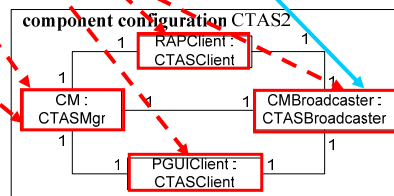


CTASClient plays the role of AwareClient

Deployment With New Component

CTASBroadcaster plays the role of Broadcaster

CTASMgr plays roles of Manager and Arbiter



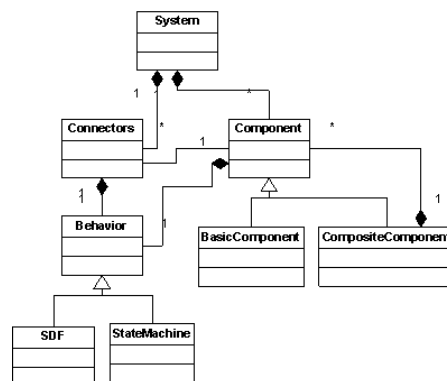
Mapping also defines the local automata

Overview

- Background and Motivation
- Diagrams for Services
- **What are Roles, anyway?**
- Queries and Constraints
- Summary and Outlook

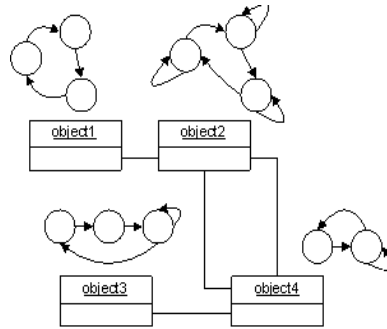
Dimensions of System Modeling: Static Structure

- Static view of the system
 - How to classify deployment
 - Behavior encapsulated in components and connectors
- Builds a graph of the system
 - Behavior is distributed over the system



System Modeling: Deployed Behavior

- Captures the running system
 - Instantiation of the model
- Behavior of the system emerges from integration of components
 - Does not explicitly describe services



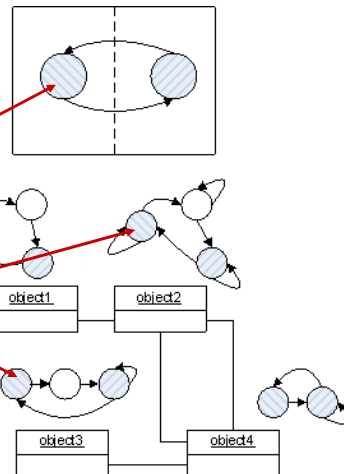
System Modeling: Deployment Structure with Modes

- System “mode” is one way to capture cross-cutting behavior

Global state one wants to observe

How it is “reached” in all the components

Cross-cutting behaviors: difficult to specify and reason about



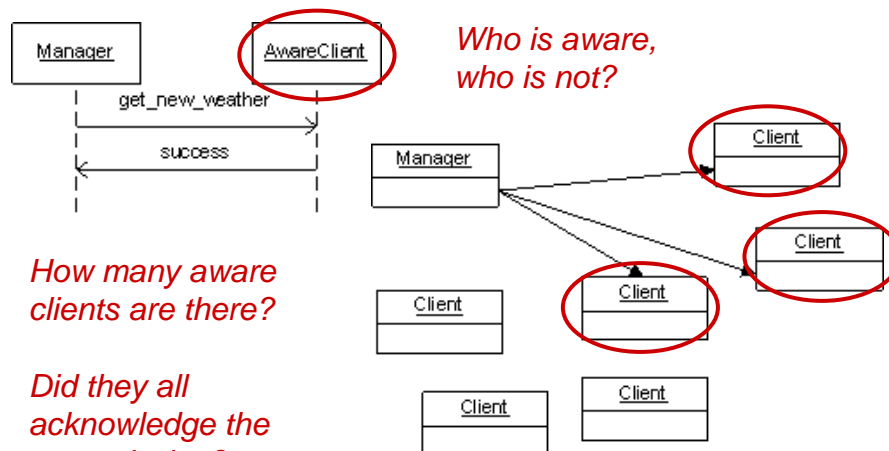
Additional Capabilities

Need for clarification on these features

1. Roles and multiplicity
 - how many components can one role cover?
2. Dynamic role change
 - Can a component change its role dynamically/switch modes?
3. Role refinements
 - Can a role refine, or imply another role?

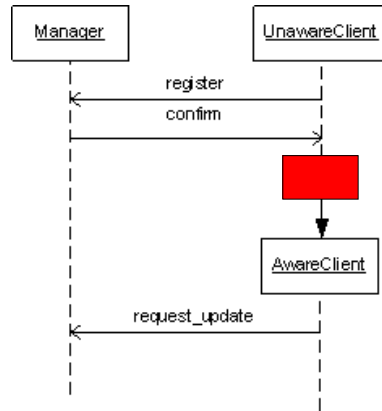
Example: Roles and Multiplicity

In the CTAS, the manager needs to identify the clients by their roles



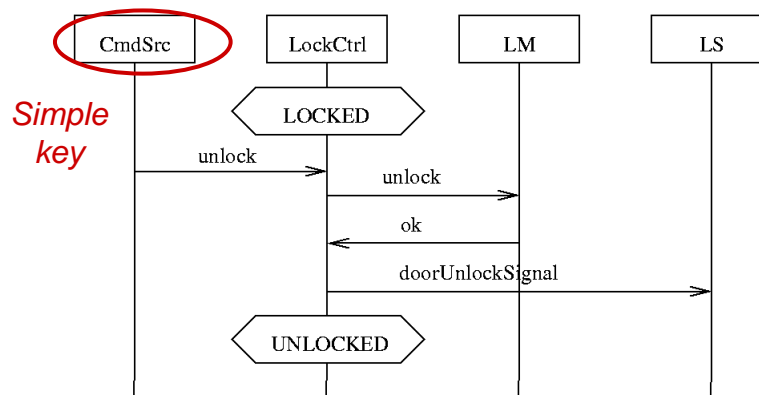
Example: Role Transitions

- Role can change in the execution
 - Implicitly specifies a state change in a component
 - Beginning/termination of roles
- Are role changes permitted during multiple concurrent services?



Example: Role Refinements

*Central locking system
basic service*

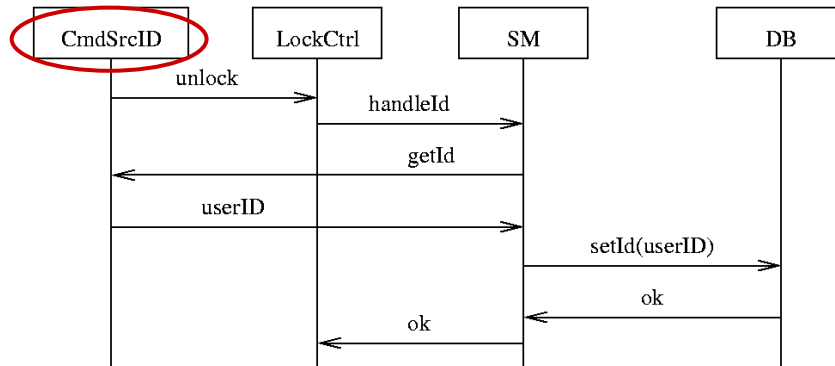


Specification for unlocking service

Example: Role Refinements (2)

One wants to add more optional features to the CLS

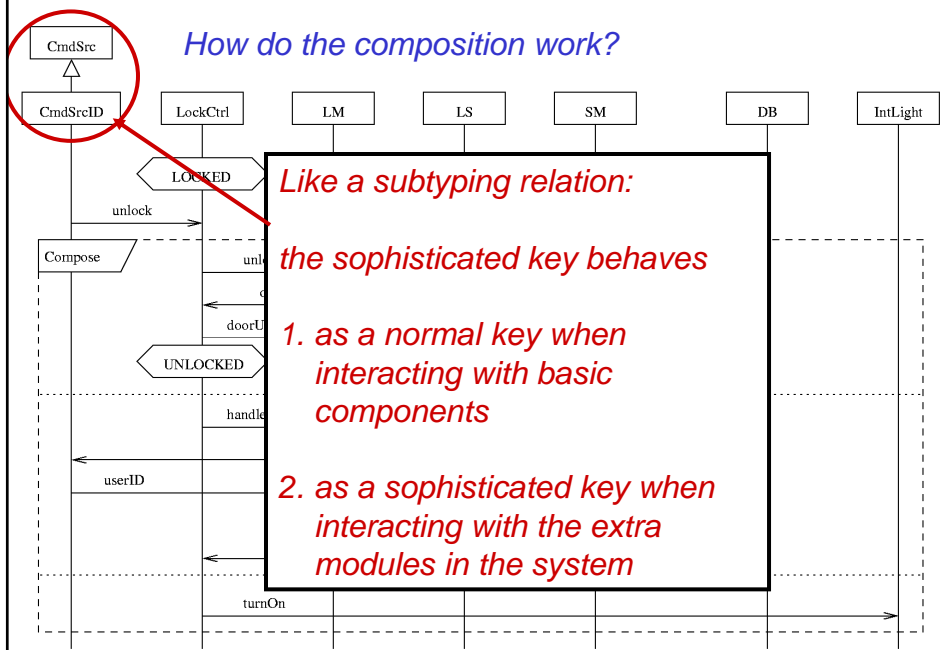
Sophisticated key



Specification adjust presets service

Example: Role refinements (3)

How do the composition work?



*Like a subtyping relation:
the sophisticated key behaves*

- 1. as a normal key when interacting with basic components*
- 2. as a sophisticated key when interacting with the extra modules in the system*

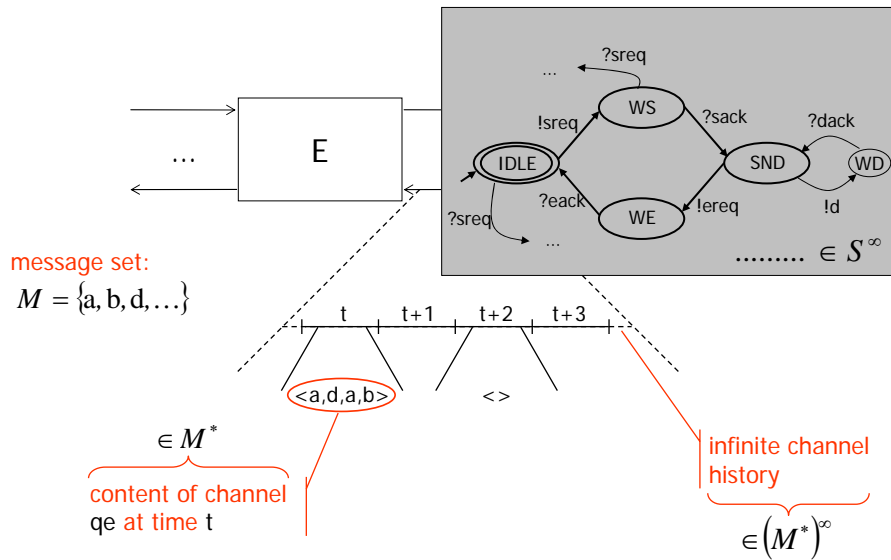
Additional Features need Extra Capabilities

- Roles and multiplicity:
 - quantification over some space
 - single instance
 - multiple instance
- Dynamic role mapping
 - encodes “super-state” transitions between roles
- Role refinements
 - classical trace refinement and subtyping theory
- How to capture these in a specification and programming framework?

Overview

- Background and Motivation
- Diagrams for Services
- What are Roles, anyway?
- Queries and Constraints
- Summary and Outlook

Precise, Methodological Foundation: Timed Streams



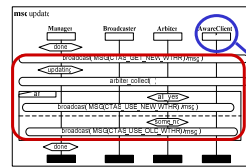
Consider Roles as Queries

- Diagram consist of two parts
 - Queries:
 - quantify over a universe
 - Constraints:
 - what the visual elements say about the selection
- “I want the entities that I queried to have the following interaction pattern”
 - only scenario
 - possibility scenario
 - negative scenario
- Query what you want to make a statement about
 - all the components that implement role
- Make that statement, by “constraining” their behavior

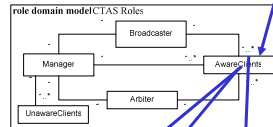


Queries and Constraints: A Sketch

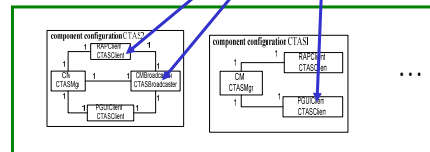
- need to specify the **universe**
 - set of all relevant properties (architecture, components, channels, ...)
- **select** from the universe:
 - select the components, for instance, based on the relevant properties for this service
- **constrain** the selected subset:
 - components have to comply to the interaction diagram



Interaction diagram



Role domain model



Universe of all implementations

Overview

- Background and Motivation
- Diagrams for Services
- What are Roles, anyway?
- Queries and Constraints
- **Summary and Outlook**

Summary and Outlook

- Services as a way to capture cross-cutting behavior
- Roles are logical entities participating in services
- Key questions:
 - Relationship between roles, services and components
 - Role transitions
 - Roles as behavioral types for interfaces
 - Expressive, comprehensive notation addressing this relationship
 - Semantic foundation
- Approach
 - Use role as queries over the deployment structure
 - Central elements of diagrams:
 - Pair (queries, constraints)
 - Queries select elements from the “target system”
 - Constraints determine behavior specification for selected elements
- Logical entities vs deployment entities
 - Example: design patterns
 - Example: framework specifications
 - roles “hybrids” between type, behavior, and instance
- Extend our semantic foundation and verification tools for these new concepts.
- Experiments/Case Studies