The Flow-Service-Quality Framework: Unified Engineering for Large-Scale, Adaptive Systems

Alan Hevner
Information Systems & Decision Sciences Dept
University of South Florida, Tampa, FL
ahevner@coba.usf.edu

Ann Sobel
Computer Science & Systems Analysis Dept
Miami University, Oxford, OH
sobelae@muohio.edu

Richard Linger
CERT Coordination Center
Software Engineering Institute
Carnegie Mellon University, Pittsburgh, PA
rlinger@sei.cmu.edu

Gwendolyn Walton
School of Electrical Engr & Computer Science
University of Central Florida, Orlando, FL
gwalton@mail.ucf.edu

Abstract

Modern enterprises are irreversibly dependent on large-scale, adaptive, component-based information systems whose complexity frequently exceeds current engineering capabilities for intellectual control, resulting in persistent difficulties in system development, management, and evolution. We propose an innovative framework of engineering representation and reasoning methods for developing these complex systems: the Flow-Service-Quality (FSQ) Framework. In dynamic network information systems with constantly varying function and usage, work flows and their corresponding traces of system services act as stable foundations for functional and non-functional (quality attribute) specification, design, and operational control. Our objective is to provide theoretical foundations, language representations, and rigorous yet practical unified engineering methods to represent and reason about system flows as essential artifacts of system specification, design, and operation.

1. Intellectual Control in Large-Scale System Development

Modern enterprises are irreversibly dependent on large-scale information systems whose complexity frequently exceeds current engineering capabilities for intellectual control, resulting in persistent difficulties in system development, management, and evolution. Critical enterprise missions depend on system services composed of complex combinations of distributed computation, communication, and human components whose interactions are often not fully understood. These
distributed systems and motivates why the FSQ Framework maintains intellectual control of large-scale, highly adaptive distributed systems. Next, the FSQ engineering process is outlined with particular emphasis placed on the refinement of system requirements into FSQ specifications. The use of the FSQ ideas on a small telemedicine example is presented in section five. In conclusion, we focus on the innovative features of the FSQ Framework and provide a brief comparison to related work.

2. The FSQ Framework

Information systems are usefully viewed as networks of asynchronously communicating components, where the components provide system services whose functions can be combined in various patterns to satisfy business requirements. The sequencing and alternation of system services in user workflows can be mapped into compositions of network component functions. These compositions are end-to-end traces that define slices of network architectures whose net effect is to carry out operations that satisfy user requirements. Large-scale systems support many users in many roles with many possible workflows, and particular system services may appear over and over in their definitions. In fact, a principal design objective in large-scale systems is coordination and synchronization of multiple uses of particular services specified by control flows. In dynamic networks with constantly varying function and usage, flows and their corresponding traces of system services act as stable foundations for functional and non-functional (property) specification and intellectual control.

The objective of our FSQ Framework research and development is to provide theoretical foundations, language representations, and rigorous yet practical engineering methods to represent and reason about system flows as essential artifacts of system development. System flows are composed of system services and are evaluated against a set of quality attributes. Therefore, it is these three first-class concepts, namely, flow, service, and quality that form the basis of this framework for engineering of large-scale, adaptive systems.

Flow:

A flow is expressed using a flow structure language that specifies user requirements in precise terms. The concept of flow embodies many of the same ideas found in current workflow research [8]. Flows can be represented as procedural structures composed of nested and sequenced service invocations that are expressed in terms of sequence, alternation, iteration, and concurrent structures. For example, a sequence structure \( \text{do } g; h \) end composes services \( g \) and \( h \), and an alternation structure \( \text{if } p \text{ then } g \text{ else } h \) end references services \( g \) and \( h \) conditionally. It is easy to see that such structures define an algebra of composition with desirable properties. Internal non-persistent state may be maintained during a single execution of a flow. Flows are graphically represented as follows:

```
[ ]
```

Flows contain graphical representations of their internal service execution orderings. Flows also define required levels of quality attributes for themselves, as well as for execution of the individual services they reference. The semantics of the flow structure language preserve powerful reasoning methods of composition and referentially transparent refinement, abstraction, and verification.

Service:

A service is a function provided by a system component or set of components. Services are specified as state machine transition functions, e.g. black boxes [11], where an input stimulus and the given state of the service results in an output result and a new service state. Services can be of one of two categories: component or basic. Component services are refined into a set of flows, whereas basic services perform a primary function and are not considered for refinement. The two categories of services, component and basic, are graphically represented as follows:

```
[ ]
```

Component Service

```
[ ]
```

Basic Service

It is inevitable that particular system components, for example, database or business rule components, will experience many uses in many flows. In operational use of a system, flows can be sequenced and interleaved by users in unpredictable ways. In fact, a principal design objective in large-scale systems is coordination and synchronization of multiple uses of service components required by user flows. Furthermore, the components of large-scale distributed systems respond asynchronously to a blizzard of inputs whose innumerable interleavings are essentially unknowable. Yet every asynchronous use of a service may change its current state and thereby its responses to future uses. Thus, any operational use of a service can encounter any possible state in execution and receive a response determined by that state. Because of this, it is necessary
in a flow specification to define the behavior of every possible responses resulting from all possible service states that may be encountered. *Relational specifications* over equivalence classes are suitable for this purpose. A relational specification can define the set of all potential responses accumulated from all the uses of a service in all the flows within which it appears. In this way, completeness and consistency can be achieved in flow definitions. Compact specifications are possible through use of equivalence classes that reduce the number of potential states to be considered.

**Quality Attributes:**
Attributes of quality, such as reliability, availability, security, and survivability are defined as computational functions and are associated with both flows and services. Substantial effort has been devoted in the past to the development of descriptive and often subjective characterizations of non-functional system properties, or quality attributes, for example, survivability attributes [2, 15]. Rather than focusing on descriptive methods, we adopt an alternate approach and ask how such properties can be defined, computed, and acted upon as dynamic characteristics of system operation. That is, we wish to define properties as functions to be computed, rather than as descriptions of capabilities to be achieved. While such functions rely on what can be computed and may differ thereby from traditional views of non-functional properties, they may permit new approaches to property analysis, design, and operational evaluation. In illustration, a function implementing computational survivability, centralized or constituent service in terms of all decentralized within the control structure of a system, could accept as stimuli the status of system services and intrusion activities, and produce as responses modified flows that maintain the survivability properties of essential services where possible [10]. While this paper focuses on the interaction of flows and services, our future research will detail the semantics and syntax of quality attribute specification.

3. **The FSQ Engineering Process**

The FSQ Framework is composed of a flow structure language and an engineering process for 1) specification of user flow types in terms of required service types and quality attributes, 2) design and verification of architecture trace refinements in terms of system services and quality attributes that satisfy specific instantiations of user flows, and 3) implementation and verification of network refinements in terms of system hardware, software, and communications that satisfy architecture traces of user flows.

Figure 1 presents the Engineering View of the FSQ Framework. Four essential development stages are represented:
- **FSQ Requirements:** System requirements consist of all required services, plus all user flows that define both service orderings dictated by the organization’s business rules and necessary quality attributes on services and flows.

![Figure 1: FSQ Framework, Engineering View](image-url)
• FSQ Specification: A thorough system requirements analysis will allow formal specification of flows, services, and quality attributes. The close interactions among the three types will result in rigorous closure verifications. For example: Are the set of services necessary and sufficient for all user flows? Do the flows and services define their quality attributes in ways supportive of clear selection of service implementations for execution?

• FSQ Design: The design stage will generate flow, service, and quality attribute instances. Each instance is a fully realized design of the corresponding FSQ specification. Quality criteria (e.g., performance, security, availability) may dictate the design of redundant service and flow instances. The design of the FSQ instances can be verified against the FSQ specifications.

• FSQ Implementation: The network architecture provides the foundation for the allocation of service instances onto the distributed system. The network architecture consists of system nodes, transport connections between nodes, and the distribution of system control and communication among the nodes. The implementation of the distributed system can be verified against the FSQ designs.

A primary control task in large-scale systems is managing the sequencing of system services in real time to satisfy flow specifications. System control functions reconcile flow requirements with service availabilities, and are a natural vehicle for implementing operational management strategies based on dynamic network and service capabilities and workloads. The concept of an FSQ manager, either centralized or decentralized within the architecture of a system, can embody flow instantiation and management functions that include quality attribute management, for example, dynamic survivability management through a variety of strategies such as rerouted communication paths, resource substitutions, state purging, alternate provisioning, and system re-initialization and reconfiguration. The FSQ Manager is a key component of the Operations View of the FSQ Framework as shown in Figure 2. The FSQ Manager can be designed and instantiated in a variety of forms and technologies, depending on user requirements, network configuration, and the operational environment. The implementation details of the FSQ Operations View will be presented in our future research.

4. Developing FSQ Specifications

We focus now on the process of refining FSQ requirements into specifications. Generally, the process can be described as the decomposition of a user’s view of an abstract service into a set of flows where each flow is represented as an ordering of execution on a set of new, refined services. This decomposition process proceeds in a recursive manner in order to maintain referential transparency. The process begins by eliciting the user’s service requirements and each service’s quality constraints.

Figure 2: FSQ Framework, Operations View
These services may also be constrained by business rules. The recursive component of the process continually adds implementation details to each service of the FSQ requirements until a fully realized FSQ specification is reached. Iterating over the main body of this process permits and easily integrates user modifications to the user’s original set of FSQ requirements. Incremental service development is supported by viewing any basic service of the FSQ specification as a service of a FSQ requirement with its own quality and business rule constraints.

The FSQ set of requirements includes the necessary services, an ordering of these services, and their quality attributes. The services are described at their highest level of abstraction and represented using black boxes. Services may also have flow requirements that are dictated by the user’s set of business rules. Each service may and likely will require a set of quality attributes for itself and for any system implementation that provides the service.

Once an initial requirements statement is completed, the FSQ specification process begins. The algorithm in Figure 3 outlines the human process of refining FSQ requirements into an FSQ specification.

```plaintext
For each requirement Service, s
    Do
        RefineService(s)
            For each q - Quality on s
                VerifyQualityAttribute(s, q)
            Until all q - Quality on s are satisfied

    RefineService(S)
        If S is basic
            RETURN
        Otherwise
            Do
                Identify Flows of S
                For each Flow, f
                    SpecifyFlow(f)
                        Until specification of S is satisfied by all f of Flow
                For each q - Quality on S
                    VerifyQualityAttribute(S, q)
                Until all q - Quality on S are satisfied

    SpecifyFlow(F)
        Do
            Do
                Identify and sequence Services of F
                For each Service, s
                    RefineService(s)
                Until specification of F is satisfied by control sequence and all s of Service
                For each q - Quality on F
                    VerifyQualityAttribute(F, q)
                Until all q - Quality on F are satisfied
```

Figure 3: FSQ Specification Algorithm
The set of services listed in the FSQ requirements are refined into either basic services or expanded into a set of flows of new services and a predicate that is used to select one of these flows. If the latter refinement is chosen, then the set of flows must also satisfy the specification of any quality attributes associated with the refined service. Also, any new services represented in these flows must satisfy these quality attributes as well. If the set of flows does not satisfy the specification of the service or the service’s quality attributes, a new set of flows must be considered. This process is recursive in that these newly generated services must also be refined. The FSQ specification process ends when all services are refined into basic services. It should be noted that basic services are not required to maintain this distinction throughout the lifetime of the FSQ system; they may, at some later date, be subject to further refinement.

5. Example: A Telemedicine System

To illustrate the application of the FSQ Specification Process, we present a brief example inspired by a recent workshop case study on remote medical care systems [12].

Telemedicine is becoming an increasingly important delivery mechanism in effective health care systems. Distributed telecommunications capabilities and supporting information systems can enable the delivery of medical services to remote locations or to the patient home. Telemedicine examples include:

- A patient has a medical monitor (e.g., a pacemaker) that can be read and updated remotely via the telephone or the Internet. The monitoring software can raise an alert and transfer control to a human coordinator if dangerous values are detected.
- A patient located in a remote location can be seen by a physician via video conferencing. Medical evaluation of conditions requiring visual and audio perceptions can be performed.
- Routine and, if necessary, emergency medical requests can be sent to a designated help center. A trained medical coordinator makes a judgment on criticality and determines an effective treatment plan that may include dispatching an ambulance or establishing an online medical consultation and diagnosis.

To illustrate the application of the FSQ framework, we develop a high level specification of a telemedicine system. A complete set of requirements for a telemedicine system is beyond the scope of this paper. We focus on user flows and system services in the example. Quality attribute specification will be detailed in a future paper. We assume a context in which a health care organization provides a defined group of telemedicine offerings for its registered clients. Clients, as patients, can schedule telemedicine appointments or contact the system for ad-hoc encounters.

5.1 Telemedicine System Services

Modeling the system in the FSQ framework begins by eliciting essential system services from the primary stakeholders. Stakeholders of the telemedicine system include patients (i.e., paying clients), health care professionals (e.g., physicians, nurses, care coordinators), health care organization officers (e.g., CIO, accounts receivable), and various regulators (e.g., quality of service auditors). A partial list of services for our example is:

- Patient Encounter – A client enters the system, transmits requested data (e.g., voice, video, monitored data), and receives a diagnosis and recommended course of action.
- Physician Interaction – A physician enters the system and interacts with one or more patients in encounters.
- Patient Repository – Operations on a patient repository are performed to add, delete, and update information on electronic patient records.
- Policy Repository – Operations on a telemedicine policy repository are performed to add, delete, and update health care policies. For example, patient eligibility for telemedicine procedures would be defined as rules in the repository.
- Patient Schedule – A patient schedule is maintained with operations for adding, deleting, updating, and querying the schedule.
- Patient-Procedure Logs – Logs of all activities in the telemedicine system are maintained for use in control and accounting procedures.

Quality attributes are identified and defined as requirements on each of the essential services. Telemedicine systems have critical quality attribute demands [7]. For example, system availability must be 24x7 for ad-hoc emergency Patient Encounters. In addition, performance and reliability are important for services with on-line data transmission requirements. The formal specification of quality attributes will not be detailed further in this paper.

5.2 The Patient Encounter FSQ Specification

The FSQ Specification Process, as detailed in Section 4, is applied to each of the essential system services. We select the services of Patient Encounter
and Physician Interaction to illustrate this process. Due to space limitations, only one level of service-flow recursion is demonstrated in each example.

For the service Patient Encounter, we identify a set of possible flows based upon the identity of the client, the telemedicine procedure requested, and other encounter parameters. Let us consider a normal patient flow called Normal Patient. The process calls SpecifyFlow(Normal Patient). The component services and their sequencing are specified as shown in Figure 4.

We identify six composite services in a defined control structure. Each of the services is now refined by the RefineService process as referentially transparent components. The recursive decomposition process continues until a service is considered basic, or not subject to further decomposition. To illustrate, consider one of the possible flows for the Verify Schedule service as shown in Figure 5. The flow begins with a query on the Patient Schedule Service, which is considered a basic service. Patient Schedule Service encapsulates the Schedule data structure and provides an interface with basic operations on the Schedule. If the patient is not currently scheduled, then the flow moves to a Verify Eligibility composite service, which would use the Patient Repository Service and the Policy Repository Service to determine patient eligibility for the requested procedure. Finally, if the telemedicine procedure is to be performed the patient/procedure request is entered into the Active Queue Service which is a basic service encapsulating the active queue data structure.

The specification of the Patient Encounter service continues recursively until all services are satisfied by the specification of all its internal flows and all flows are satisfied by the specification of all its internal services.

**Figure 4: Patient Encounter Flow**

**Figure 5**
5.3 The Physician Interaction FSQ Specification

The FSQ specification process for the Physician Interaction Service would proceed identically to the process described above. The composite service would be refined into a set of possible flows, each of which would be sent to the SpecifyFlow procedure. Again, considering a Normal Physician Flow, as shown in Figure 6, a set of six internal services are identified within a defined flow structure.

While we will not continue the recursive specification further for this example, it is important to note the synchronization points between the Normal Patient Flow and the Normal Physician Flow:

- Transmit Data in the patient flow must precede the Receive Data in the physician flow.
- Send Diagnosis in the physician flow must precede Receive Diagnosis in the patient flow.
- Send Patient Actions in the physician flow must precede Receive Patient Actions in the patient flow.

Such points of synchronization between interacting system flows must be explicitly specified in the internal services where required. Thus, in this example, the Receive Data, Receive Diagnosis, and Receive Patient Actions services will include a synchronization operation (e.g., a wait state) that holds until the expected input is received.

6. Contributions of the FSQ Framework

Our goal in this paper is to briefly describe the Flow-Service-Quality (FSQ) Framework. We believe the identification of Flow, Service, and Quality as first-class concepts in the development of large-scale, adaptive systems is central to achieving unification of this complex engineering activity. While the content of this paper barely scratches the surface of the comprehensive FSQ Framework, it is the first ‘stake in the ground’ in laying out a full-scale FSQ research project. The following observations on the FSQ Framework - its major advances, strengths, and unique features - summarize our research vision.

The FSQ Framework provides intellectual control over both the engineering and operation of large-scale, adaptive systems. The separation of concerns between the engineering view and the operations view supports appropriate control and focus. Types and their refinements are the only concern in specification and traces are the only concern in operation, thereby avoiding undue development complexity.

- The FSQ Framework permits unification of network system engineering through uniform, scale-free semantic structures for requirements, specification, design, and implementation; and generic control architectures for managing system operation, service quality, adaptation, and evolution.
- The FSQ engineering view of system development is based on a seamless hierarchical decomposition beginning with user flow and service requirements and ending with flow and service implementations. FSQ system development provides intrinsic traceability from requirements through implementations.
- User flows of service uses drive FSQ system engineering. Systems are decomposed in terms of a user’s view of services, rather than applying a strictly functional or object-based composition.
- The FSQ operational view of system control is achieved through a systematic process of flow
management of system services and quality attributes.

- FSQ adaptive behavior is achieved through reconciliation of flow processing requirements with system capabilities computed dynamically in system operation.
  
  The FSQ Framework supports rigorous decomposition, composition, and referential transparency in systems development. Central to the FSQ Framework is the definition of scale-free flow structure primitives and algebraic rules for their combination that preserve compositional reasoning and referential transparency for precise abstraction, refinement, and verification of specifications, designs, and implementations of high-confidence network systems.

- The FSQ Framework provides a recursive development process wherein flows define sets of services and services define sets of flows.

- Asynchronous behavior is effectively dealt with by relational specifications of system services. Every service use in a flow must account for all possible responses resulting from asynchronous uses of that service in all the flows in which it appears, with the specific response obtained determined by a postifix predicate on every use. This use-predicate structure deals with the inherent non-determinism of asynchronous execution, and enables powerful reasoning methods of referentially transparent composition, abstraction, and refinement to be applied.

- The effort required for FSQ system scale-up through addition of user flows and system services is no more than linear due to the inherent localization and absence of transitive propagation of effects of such additions.

- Any flow can be constructed from a basis set of single-entry/single-exit primitive structures that provide service uses in sequence, alternation, iteration, and concurrent forms. Flows expressed in this flow structure form can be refined, abstracted, and verified with mathematical precision [9].

  Functions are used to describe quality attributes. The ability to model quality attributes as functional entities is an innovative feature of the FSQ framework. The notions of system quality attributes are open to improved understanding and optimization in this representation. Numerous quality attributes are candidates for modeling in a distributed software system, including security, survivability, availability, reliability, performance (time and space), usability, safety, fault tolerance, interoperability, and accuracy.

- FSQ quality attribute management is achieved through reconciliation of quality requirements specified for every flow and its services, with quality capabilities computed dynamically in system operation.

7. Relationship to Previous Research

We recognize the existing research streams that contribute to the FSQ Framework. In the following we briefly describe these research directions and their relationship to FSQ.

Workflows:

In modern system development, workflows are a principal method for specifying business requirements [8, 13]. A workflow can be modeled graphically as a trace diagram describing the order in which services are to be executed. An FSQ flow is analogous to a workflow, yet, specified in connection with embedded services and qualities, can provide additional engineering capability.

Use Cases:

Use Cases are a graphical notation for capturing user-system interactions [5]. They have become popular as a part of the Unified Modeling Language (UML). While valuable for brainstorming with users, Use Cases have well known deficiencies that preclude their use as a formal requirements specification language [3].

Component-based Systems:

An important direction in systems development is component-based software development [1]. The effective use of software architecture concepts provides a blueprint for the construction of large systems made up of smaller, pre-developed system components. The components are selected and connected to provide required business system functionality. A goal of component-based software development is to have an open marketplace for vendors to develop state-of-the-art components that will plug-and-play in open system architectures.

Services in the FSQ Framework are analogous to the concept of components. While a few research studies have addressed the combination of services and quality attributes (e.g., [16]), a systematic framework to do so does not exist. Methods that attempt to rigorously describe system-level behavior and quality properties typically do not scale well to systems the size and complexity of critical infrastructure systems [15]. System descriptions are often given at a high level of abstraction and provide insufficient rigor for quantitative analysis at the level of detail necessary to understand whether a system can satisfy qualities of low-level essential services.
Open Distributed Processing:
The term Open Distributed Processing (ODP) refers to an ISO standard framework for the development and operation of distributed systems [6]. The ODP Reference Model consists of a set of viewpoints that define functions that implement transparencies. ODP is an on-going effort to provide a unifying foundation for the many distributed system standards in existence. The goal of FSQ is to provide a cleaner, more parsimonious model of large-scale system development that includes quality as a first-class concept.

Agent-Based System Coordination:
Research on agent-based systems addresses the problem of coordination and integration in large-scale distributed systems using a multi-agent framework [14]. This representational formalism is used as the basis for integrating a collection of autonomous information systems, each of which is regarded as an agent. Different types of coordination mechanisms and control schemes can be used for integrating the different information system units. System levels of quality attributes can be evaluated based on the interaction of the agents.

8. Conclusion

The FSQ Framework is fundamentally different from the approach of others in that the focus is on the ability of the system to satisfy individual requests for services at specified quality levels. Systems are hierarchically decomposed in terms of user requests for services, rather than in terms of functions or objects. Large-scale distributed system control is reduced to a systematic process of flow management of service uses and quality assessments, centralized or decentralized within the structure of the system. Quality attribute analysis is reduced to systematic analysis of flows of service uses, and computation and composition of survivability properties with respect to dynamic models of flows in operation.

9. References


