Interaction Pattern Gathering in Service-oriented Applications

A. M. Silva Filho  
Innovation and R&D  
CESAR Research Institute  
amsf@cesar.org.br

H. K. E. Liesenberg  
Institute of Computing  
University of Campinas  
hans@unicamp.br

R. S. M. Barros  
Center of Informatics  
University of Pernambuco  
roberto@cin.ufpe.br

Abstract

Application integration is a challenge and it will continue for the next few years. The industry is still figuring out how to best address the challenges of application heterogeneity through modern integration tools and service-oriented and event-driven architectures. We are experiencing a new paradigm shift. However, the impact of service-oriented technology will be evolutionary rather than revolutionary. Companies need to support enterprise service buses and selectively apply business component design practices and Web Services. This paper addresses how interaction patterns in service-oriented applications can be gathered and discusses related collaboration patterns.

1. Introduction

Today, we are living the information era. In this era, people can perform several activities without regarding to geographical location. In that sense, it is possible to interact with colleagues, share data and computational resources, access instrumentation, and access information in digital libraries [Watters 1998]. This has become possible because of the support the Web provides to human interactions with textual data and graphics.

People use the Internet daily to look up stock quotes, buy consumer goods, and read the latest news. Nevertheless, to transfer large amounts of data, applications need to interact directly with one another, automatically executing instructions that would otherwise have to be entered manually through a browser. Note that Web applications need to be able to find, access, and automatically interact with other ones. Web services improve Internet use by enabling program-to-program communication. Through the widespread adoption of Web services, applications at various Internet locations can be directly integrated and interconnected as if they were part of a single, large system [Guest 2003].

The aforementioned scenarios can be envisaged as a group of services and a set of applications they are working on. Herein, services are used in general sense. Within this context, we want a service-oriented application in which all services being shared on it can be accessed indifferently from the geographical location and the objects can interact with each other in real-time [Silva Filho 2000].

Note we have a collaborative software that demands quick access to the services for the purpose of rendering within a single view. Getting into this point involves the provision of support to collaborative systems design. Two important features being addressed in this paper are the interaction mode and geographical distribution.

Next section provides background issues on dependability requirements in social computing. Then, our protagonist-oriented approach for gathering interaction pattern in service-oriented applications is presented. Collaboration patterns issues are also discussed. Concluding remarks are given at the end.

2. Dependability in Social Computing

Today the traditional computing is giving way to the social computing which allows an ever increasing interaction of research fields, involving people of various backgrounds and abilities as well as of different cultures. Social computing is one of the examples of the Internet era where a very large
number of Web sites can be visited, queried and played with. Schuler says that

“social computing describes any type of computing application in which software serves as an intermediary or a focus for a social relation”. [Schuler 1994].

Within this context, designing service-oriented applications involves a great deal of exploration of alternative solutions. Each solution has to be analyzed in order to evaluate its suitability. In that sense, dependability is the most system property in complex socio-technical systems [Silva Filho 1999].

The dependability of a system is a judgement about the user’s trust in that system. It reflects the extent of the user’s confidence that it will operate as expected and that it will not ‘fail’ in normal use. Usefulness and trustworthiness are not the same thing. A system does not have to be trusted to be useful, so long as the user is aware of the risks.

Dependability is an emergent system property that cannot be predicted until the system has been integrated [Littlewood 2000]. Emergent properties are unpredictable because of the complexity of the relationships between components and the environment. Environmental factors are particularly important for the perception of system dependability. The main dependability attributes are:

- **Reliability** – The ability of a system to deliver the services as specified.
- **Safety** – The ability of a system to operate without catastrophic failure.
- **Availability** – The ability of a system to deliver the services when requested.
- **Security** – The ability of a system to protect itself against accidental or deliberate intrusion.

Other important factors for dependability are maintainability, recoverability and timeliness. Note that dependability is part of quality of services (QoS) in Web services. By QoS, we refer to non-functional properties of Web services such as performance, integrity, reliability, accessibility availability, and security. QoS covers a whole range of techniques that match the needs of service requesters with those of the service provider's based on the network resources available.

It is worth observing that service providers can provide high QoS to the service requesters, by using approaches like caching and load balancing of service requests. Caching and load balancing can be done at both Web server level and at Web application server level. Load balancing prioritize various types of traffic and ensure that each request is handled suitably. To do capacity modeling, a Web service provider can create a top-down model of request-traffic, current capacity utilization, and the resulting QoS. Note that a service provider can also categorize Web service traffic by the volume of traffic, traffic for different application service categories, and traffic from different sources. This will help in understanding the capacity that will be required to provide good QoS for a volume of service demand and for future planning like capacity and type of load balancing Web application servers [Landry 2002].

In addition, design activities usually lead to a constant need for communication between all the people within the development. In that sense, design issues depend on appropriate methods for describing the solutions at hand.

The development of large Internet-based systems is complex and it becomes even more difficult because of the need of involving the user as an active participant during the entire process. Henceforth, we need a continuous evaluation during the whole development process. This requirement justifies the need of incorporating human factors into the system development aiming to achieve quality applications.

Interaction design involves user actions, interface feedback, screen appearance, and user tasks. On the other hand, application software design involves algorithms, data structures, widgets, and calling structure of modules.

Within this context, we have been investigating how interaction design models can contribute to the design and development of Internet-based applications. This is discussed in Section 4. Next we discuss design issues in Web applications.

### 3. Interaction Pattern Gathering

In this section an approach to gather interaction patterns is presented. Herein, our interest lies on the way applications interact with each other, i.e. their control component design. In that sense, in the
In introductory section, we point out that social computing describes any type of computing application in which software serves as an intermediary or a focus for a social relation. A set of design difficulties arises in Internet-based applications as highlighted below.

- the inherent complexity of tasks and applications;
- the variety of different aspects and requirements;
- theories and guidelines are not sufficient;
- difficulty of doing iterative design.

In order to tackle these afore-said difficulties and to better support the process of developing Internet-based applications we consider the tasks carried out by protagonists. Protagonists are all the components that play roles in an interaction scenario. Such protagonists are both user(s) and system components. These components can as well be viewed as software components with major functionality roles. Note services are composable and loosely coupled, as well as the reusable logic can be divided into services.

This approach for gathering interaction pattern considers the diversity of users with different knowledge levels as well as a variety of both interaction styles and implementation technology. This is important in collaborative scenarios, as in Internet-based applications, due to the inherent complexity.

We also consider the need to start interaction patterns gathering at a high abstraction level where only user intentions are captured. This makes this step easier for handling and upgrading as the refinement process proceeds until detailed requirements are obtained and implementation decisions are made.

Besides the use of the protagonist-based approach, we can get users involved early enough in the design process provided that appropriate metaphors related to their daily tasks are used. Using metaphors in such a process augments the integration between users and designers because they can communicate effectively between each other using a common language derived from such metaphors. This approach based on metaphors of the domain aims at facilitating the user involvement in the design process.

In addition, user intentions at the task abstraction level are captured as high-level goals which exist in the user's conceptual model about the system. User intentions to reach a goal are described at a high abstraction level without reference to any system presentation feature. Figure 1 illustrates our approach.

![Diagram](image)

**Figure 1 – Service Interaction Patterns Gathering.**

**Requirements Gathering** - It is concerned with understanding needs. That is, this phase aims at understanding the nature of the system to be developed and the required functionality. Within this context, the main objective is to capture the system's organization, i.e., viewing such an organization as a collection of protagonists. In addition, this process aims at identifying the role that each protagonist will play. At the end, the designer is expected to obtain a system description of the system under development (SUD) [Silva Filho 2004, Liesenberg 2004].
**System Protagonists Identification** - These protagonists can be both user and system components depending on decisions of what parts of a working process should or should not be carried out automatically. Note a service is considered a loosely coupled component. The designer makes use of the general service model (GSM) to capture the system protagonists playing roles in interaction scenarios. A way of doing that is developing interaction scenarios involving protagonists.

**Interaction Scenarios Development** - The designer must describe interaction scenarios involving the system protagonists in order to identify their goals and tasks. It is worth observing that all interaction details such as the kind of information exchanged between protagonists are abstracted out. The focus is on the interactions between protagonists.

**System Architectural Model Identification** - A system is described in terms of its architectural model, i.e. the system is in terms of its components or protagonists. It is worth pointing out that both users and software components can play the role of protagonists.

**System Protagonist Services Representation** - From the identification of the protagonists and their interactions at a high abstraction level an architectural model is identified. Such identification is derived based on the interaction scenarios worked out where each system protagonist is capable of performing some particular tasks. In that case, the task set for each protagonist is described by using Service Interaction Models (SIMs), Service Hierarchical Models (SHMs), and Service Component Model (SCM). These models comprise the specification of a service-oriented application.

**Service analysis and design enhancement** – The three models (SHM, SIM and SCM) are used to obtain the specification. Note that all the models are given at a high abstraction level and so a further refinement is needed before finishing the design specification. It is worth observing that TSM, SIM and SCM are simply distinct viewpoints of the same entity, that is the service set of the system protagonists.

---

### 4. Collaboration Patterns

Previous section discussed a way of gathering interaction patterns for Internet-based applications. This section discusses four possible collaborative relationships we may face with in such applications.

A **collaborative relationship** describes how the partners or components of a system interact with each other. A set of partners can collaborate in such a way that they can observe changes of states and/or data values from other partners. Table 1 shows a set of four possible different collaborative relationships [Silva 2004].

<table>
<thead>
<tr>
<th>Collaborative relationships</th>
<th>Collaboration level</th>
<th>Knowledge about other partners</th>
<th>System complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stand-alone Mediator</td>
<td>Low</td>
<td>No</td>
<td>Low</td>
</tr>
<tr>
<td>Everyone knows each other (EKEO) Mediator</td>
<td>Medium High</td>
<td>No</td>
<td>Medium High</td>
</tr>
<tr>
<td>Mediator and EKEO (MEKEO)</td>
<td>Medium</td>
<td>Yes/No</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 1: Collaborative relationships.

Consider a **stand-alone component**. It does not interact with another partner in the system. In general, its state and operations do not affect the overall timing constraints. For instance, a power display component is only receiving the power reading and showing the power value without interacting with any other partners.

Alternatively, interactions among different partners can be coordinated through a **mediator**. In this case, every partner registers the services it can provide and the services needed from other partners while the mediator keeps records of provider and requester profiles.

The collaboration pattern called **everyone knows each other** is that which allows partners to register and request directly one from another. In that case, each partner needs to know the interfaces of the other partners and to request the services through these interfaces. Herein, a **peer-to-peer architecture** can be used where all the partners involved in a collaboration relationship can be either serving or clienting at any time. This collaboration pattern may
also have a broadcaster (working in background) and, therefore, not interfering with the communication process among partners. An Internet-based application called DirectionLeader, using this collaboration pattern, is presented by Silva Filho and Liesenberg [Silva Filho 00].

The other collaboration pattern is that where there exists a mediator and everyone knows each other. In this case, partners can request services either from the mediator or directly from providers through their interfaces. An example of this collaboration pattern is the application presented in [Silva Filho 2003], an Internet-based system for multiple users, called NetConnect4.

Table 1 shows that the EKEO (everyone knows each other) collaboration pattern has both high collaboration level and high system complexity. As for the collaboration pattern called mediator and everyone knows each other the system complexity is also high, but it has a slightly lower collaboration level compared to the everyone knows each other.

The high system complexity takes place because an application where all the partners involved in a collaborative relationship know each other is intrinsically more complicated than single-user systems. Issues like handling updates from several applications are difficult, specifically in decentralized systems, due to the need to support uniformity. Note this is an important issue when dealing with limited bandwidth and network delays. Furthermore, the Internet-based applications are to consider the following requirements:

- **Functionality supported** - Applications involving collaboration can support three kinds of functions:
  - Computer-mediated communication, that supports direct communication between protagonists. An example is the email system.
  - Decision support system, that aims at capturing a common understanding about the tasks the protagonists are to perform. An example may be a shared drawing surface which can be used for synchronous remote design meetings.
  - Shared applications, which support the protagonists' interactions with shared work objects, i.e., the artefacts of work. Henceforth, users can act upon the shared workspace at any time.

- **Sort of information being shared** - Internet-based applications can share different sort of information, such as text, image, report, and audio. In addition, collaborative systems may differ as to the granularity of sharing they allow in terms of both object chunk size and frequency of update.

- **Network delays** - In Internet-based application, the feedback loop includes transmission over the network, then network delays are to be considered. For example, the granularity of messages is an important design choice because it will imply in a low or heavy network traffic. Other issue is the system architectural model being used.

- **Types of activities supported** - The activities supported in Internet-based systems, are determined by the goals and tasks of the users. This includes e-commerce where you may have short-term collaborations, to inquire about and then order a product, as well as long-term negotiations, to craft a major business deal. Another example is the structured work processes that let people with different roles collaborate on a task such as the edition of a scientific journal involving online submission, reviewing, and publication.

5. Concluding Remarks

This paper has presented a way of how interaction patterns in service-oriented applications can be gathered and dealt with. From our experience based on case studies worked out involving Internet-based applications, one of the major difficulties in designing such applications comes from the fact that we need support integration in heterogeneous applications. Within this context, Web services are quickly becoming significant technology in the evolution of the Web and distributed computing. Web services leverage the data independence of XML to solve enterprise integration problems. Sometimes, Web services are portrayed as the most suitable solution to contemporary computing problems, filling the role previously played by the original Web, relational databases, fourth-generation languages, and artificial intelligence. Nevertheless, Web services are indeed a new layer and are not a fundamental change that replaces the need for existing computing infrastructure. This new layer of technology supports an integration mechanism defined at a higher level of abstraction.
References


