complex adapters for the needs of composition is not required. Several standards are associated with Web services like UDDI, WSDL, and SOAP. For composition requirements, a composite service is always associated with a specification that describes among others the list of component Web services that participate in the composite service, the execution order of these component Web services with respect to data and temporal dependencies, and the corrective strategies in case of exception handling.

The Web services community uses different languages for specifying Web services composition like BPEL and WSFL. The primary objective of these specification languages is to provide a high-level description of the composition process independent from any implementation details or concerns. The specification of composite services is also concerned with the semantics of information exchanged by the component Web services. The need for a common semantics is intensified when Web services, which originate from distinct providers, participate in the same composition.

Despite the widespread use of Web services, they still lack the capabilities that could propel them to the acceptance level of traditional integration middleware like Corba and DCOM. This lack is primarily due to the trigger-response pattern imposed on the interaction of Web services with the external environment of peers, users, and so on. The compliance with the trigger-response interaction pattern means that a Web service only processes the (say, SOAP over HTTP-based) requests it receives without considering its execution status before it commits to satisfying an additional request, or even questioning if this additional request would be, in the future, rewarding for its potential selection from a community of similar Web services. In addition to this interaction pattern, Web services are still considered by some as distributed objects that react upon request [2].

However, there are several situations that call for Web services self-management to satisfy the requirements of scalability, flexibility, and autonomy. By scalability requirement, we mean the capacity of a Web service to interact with a small or large community of Web services without having its expected performance either disrupted or reduced. By flexibility requirement, we mean the capacity of a Web service to adapt its behavior by selecting the appropriate operations that accommodate the situation (for example, negotiation, monitoring) in which it operates. Lastly, by autonomy requirement, we refer to the capacity of a Web service to accept demands of participation in composite services, reject such demands in case of unappealing rewards, or even propose other alternatives for its participation by recommending other peers. To reduce the current limitations of Web services, they must first assess their current capabilities and ongoing commitments, and second, their surrounding environment prior to binding to any composition. In fact, Web services must be context-aware. Context is the information that characterizes the interactions between humans, applications, and the environment [3].

By developing context-aware Web services it would be possible, for example, to consider the aspects of the environment in which the Web services are to be executed. These aspects are multiple and can be related to users (stationary, mobile), their level of expertise (expert or novice), computing resources (fixed device, handheld device), time of day (in the afternoon, in the morning), and physical locations (meeting room, cafeteria). Here, we discuss the suitability of context for Web services. This discussion revolves around three major elements: how to deploy context-aware Web services, how to contextualize Web services composition, and how to conciliate contexts of Web services.

On deploying context-aware Web services. Context-aware computing refers to the ability of a software application to detect and respond to changes in the environment. Making Web services context-aware is not straightforward; many issues must be addressed (adapted from [8]): How is context structured? How does a Web service bind to context? How are changes detected and assessed for context update purposes? What is the overload on a Web service of taking context into account?

Responses to some of these questions have led us to organize the context of a Web service along three interconnected perspectives (Figure 1). The participation perspective is about overseeing the multiple composition scenarios in which a Web service

By developing context-aware Web services it would be possible to consider the aspects of the environment in which the Web services are to be executed.