Towards Autonomic Multimodal Interaction

Pierre-Alain Avouac, Laurence Nigay, Philippe Lalanda

Towards Autonomic Multimodal Interaction

Pierre-Alain Avouac
Grenoble University
220 rue de la chimie
38 000 Grenoble
France
Pierre-Alain.Avouac@imag.fr

Laurence Nigay
Grenoble University
220 rue de la chimie
38 000 Grenoble
France
Laurence.Nigay@imag.fr

Philippe Lalanda
Grenoble University
220 rue de la chimie
38 000 Grenoble
France
Philippe.Lalanda@imag.fr

ABSTRACT
Heterogeneity and dynamism of pervasive environment prevent to build static multimodal interaction. In this paper, we present how we use the autonomic approach to build and maintain adaptable multi-modal interaction. We describes characteristic of adaptation, realized by an autonomic manager that relies on models specified by interaction designers and developers. Finally, an example with a real application and existing devices is explained.

Categories and Subject Descriptors
H.5.2 **[Information Interfaces and Presentation]**: User Interfaces – *input devices and strategies, interaction styles, user interface management systems (UIMS)*. D.2.2 **[Software Engineering]**: Design Tools and Techniques – *User interfaces*.

General Terms
Algorithms, Design.

Keywords
Multimodal interaction, autonomic computing.

1. INTRODUCTION
As envisioned by Mark Weiser [6], computers are getting more and more numerous and interconnected. Somehow, they even disappear from the user's awareness that now tends to reason in terms of services and not in terms of computing elements. One purpose of pervasive computing indeed is to realize this vision of increasingly ubiquitous network-enabled devices. Specifically, it aims at filling our environment with communication-enabled devices in order to assist us in our daily activities.

Service-oriented Computing (SOC) has recently emerged and has clearly fostered the domain of pervasive communication-enabled devices [4],[1]. The very purpose of this reuse-based approach is to build applications through the late composition of independent software elements, called services. Services can be implemented within smart devices. They are described and dynamically published by service providers; at runtime they are chosen and invoked by service consumers. This is achieved within a service-oriented architecture (SOA), providing the supporting mechanisms. Service orientation brings in major software qualities. It promotes weak coupling between consumers and providers, reducing dependencies among composition units. Late binding and substitutability improve adaptability. Since a service can be chosen or replaced at runtime, it is easier to improve the way requirements are met. A number of implementations have been proposed in the last years. Web Services (www.w3c.org), for instance, represent a solution of choice for software integration. UPnP (www.upnp.org) and DPWS (Devices Profile for Web Services) are heavily used in pervasive applications in order to implement volatile devices. OSGI (www.osgi.org) and iPOJO (www.ipojo.org) provide advanced dynamic features to many software systems.

It is expected that, in the long term, devices will fade into the environment and users will not be aware of their location or their precise nature. There will be no need for complex, specific interfaces. Users will simply express their needs or desires and the environment and the objects in it will configure themselves autonomously. In the midterm, the situation is a bit different. Interactions with smart devices are generally explicit. Users have to engage explicit interaction with devices in order to obtain a service.

Effectively interacting with pervasive devices is complicated by several factors. First, an interaction may need several devices. It is then necessary to deal with several sources of heterogeneous data in order to trigger an action. The activity of integrating disparate information sources in a timely fashion is known under the name of mediation. Mediation has been historically used to integrate data stored in IT resources like databases, knowledge bases, file systems, digital libraries or electronic mail systems [7],[8],[3]. It is now also used to allow interoperation between heterogeneous software applications and devices [2].

Another problem is related to the number and variety of smart communication devices that is just exploding! PDAs, smartphones, set-top-boxes, cameras, and electronic appliances can be found in many houses today. As envisioned by Moore's law, these devices are getting cheaper, smaller and are pervading every aspect of our life. The problem is that this invasion is chaotic: devices use a number of communication protocols and are rarely interoperable. For instance, there are today more than 50 candidate protocols, working groups and standard specifications for home networking already exist (see www.caba.org for an updated list). As a consequence, building consistent interactions based on network enabled devices that spontaneously enter and leave the network turns to be a real challenge. We believe that significant progress is needed along several dimensions. New architectures and techniques are actually

MAASC'11, May 12, 2011, Paris, France.

needed in order to seamlessly integrate heterogeneous and changing devices and networks.

In this paper, we present an autonomic solution to this problem of multimodal interaction in heterogeneous, dynamic domains. The paper is organized as it follows. Firstly, our autonomic approach is motivated, and adaptation characteristics of our work are described. Secondly, models used to guide adaptation are explains. Finally, a concrete example is provided with emphasis on the autonomic aspect and its coupling with models.

2. AUTONOMIC INTERACTION

A multimodal interface enables a user to use more than one device to interact with a system. In the pervasive vision, a user is surrounded by devices that can be seen as resources or as means to trigger an action on the environment. Input devices, like remote controllers or webcams coupled with movement recognition, allow users to express their needs and desires. Other devices, like heating systems or movie players, host application and are actually in charge of fulfilling the users' needs. Finally, output devices inform the user of the system response: loudspeaker, screen, etc. Our work focus on input multimodality, that is to say on unidirectional communication from input devices to services-based devices.

Due to the heterogeneity of environments, no assumption can be made about how a user will interact with an application. Thus, applications and devices are independent. However, interaction designers should be able to add some of their knowledge. Finally, a user should be able to configure some aspect of interaction along its preferences. Therefore, the interaction has to be adaptable. However, doing this adaptation requires deep technical knowledge that users and interaction designers do not have. The need of autonomic emerges from this set of facts. We develop DynaMo (standing for "dynamic modalities"), a software that generates and maintains context-adapted multimodal interactions. General assumptions behind DynaMo are:

- An application is a set of tasks that consume data;
- A device is a set of sensors that provide data;
- Applications and devices are exposed as services, as defined in service-oriented computing.

Providing users with multimodal interaction facilities is a hard task where a number of requirements have to be met. A user generally has preferences that have to be considered carefully. A user does not want to deal with low level details. For instance, he/she neither wants to notify the system that a device is no more usable nor creates a whole interaction. Also, a user may want to prevent a device to be used for some task, or give a policy in order to guide the interaction generation along his or her preferences.

Providing multimodal interaction facilities is even made harder in pervasive environments, characterized by their heterogeneity and dynamicity. More precisely, the following aspects have to be taken into account:

- The environment (which devices and services are accessible at a given point in time?);
- Information about devices and services (how do services and devices interact?);
- User preferences (which policy should follow an interaction?).

In our work, we propose to use an autonomic manager in order to generate (and maintain) an interaction. Because carrying data from input devices to services can be seen as a mediation problem, we have decided to implement an interaction through a mediation chain. In this approach, the autonomic manager has thus three kinds of input, which will be detailed here after, and its output is a mediation chain. The autonomic manager is reactive: when a modification occurs in the input elements, it computes a new chain or adapts the current one.

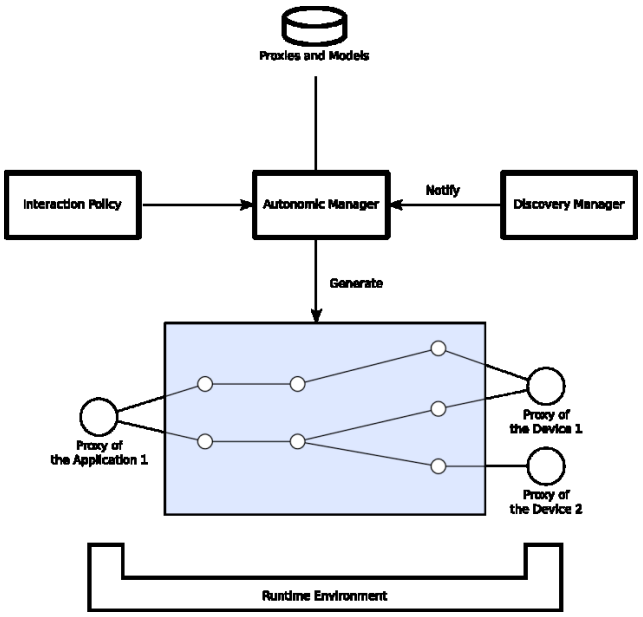

Figure 1. Global approach

In order to better understand the adaptations that have to be made by the autonomic manager, we present here the objects to adapt, the realization issues, the temporal characteristics and the interaction concerns as suggested in [5].

The **object to adapt** is a mediation chain that is made of specific elements called mediators. Adaptations are done globally, at the chain level. The impact is low from the system's point of view: because the mediation chain is in a well-defined place in the architecture, there is no side effect elsewhere. The mediation chain is decoupled from the autonomic manager. The device and application are not aware of the mediation chain, so they do no see any change. Currently, the time cost is not negligible from the user point of view (about 3 seconds). It will be really negligible when the low level anomaly is spotted: less that one second is aimed. The impact on the user can be huge: a new interaction takes place, the old interaction is no longer available. Of course, the adaptation is done in order to enhance the user communicative capabilities: a new interaction is built due to new communication possibilities. As a conclusion, the adaptation is strong because a part of architecture is modified.

The **realization issue** is split into approach and type. The decision-making is static: a user can choose a policy at runtime, but he/she cannot modify the effect of a rule without recompiling. Same thing for the models used to generate an interaction: models can be modified at runtime, but the decision-making that relies on these models cannot be changed at runtime. These statements have to be mitigated: DynaMo itself is built with the dynamic service approach. In theory, deciding processes can be changed at runtime but that has not been investigated. The adaptation is external: the adaptable mediation chain is decoupled from the

autonomic manager. This separation eases the evolution of the mediation chain in one hand, and the evolution of autonomic manager in the other hand. The adaptation is realized without learning from the user inputs. However, several cases would obviously leverage the learning. For example, if a button is never used by a user, DynaMo could propose to bind this button to another function. Usage of autonomic architecture will ease the machine learning process because sensing and effecting are already done. The adaptation is model-based: description of devices and applications resides in model. These models have to be conformed to a meta-model; that is specific to interaction domain.

The **temporal characteristics** are split into reactive/proactive adaptation and continuous/adaptive monitoring. The adaptation is reactive: a change in the environment potentially trigger the generation of the new interaction. Same thing occurs for rules: if the user chooses another rule, a new interaction is generated. No assumption can be done about the evolution of the environment. Hence, the monitoring is continuous.

Finally, **interaction concerns** are split into human involvement, trust and interoperability. The user is involved when he/she chooses a policy rule. Moreover, the user can choose which application she or he wants to control through the multimodal interaction. By contrast, changes in the accessible device set do not required user's involvement. The trust is obtained by different ways. First, used algorithms are deterministic: if autonomic manager inputs are the same, the provided mediation chain is the same. Then, appearance and disappearance of services and devices are notified to the user. Finally, a simple mechanism is used, allowing that if a button has a validation function, it will be used for validation, whatever the application deals with. The main lack concerns the observability by a user of an interaction. Currently, a user cannot easily know what task is bound to a sensor. Some works is being done to improve this.

Along this presentation, we have underscored some advantages of our modeling approach. The next section presents models that can be specified by stakeholders.

3. MODELS

The autonomic manager needs some information to generate a useful interaction. We have decided to store most of the necessary information in models. Two kinds of model are handled: proxy models and interaction models. This separation has been done in order to target the two different stakeholders: developers and interaction designers.

A developer is in charge of developing both device and application proxies. She/he has to provide one model for each proxy. A **proxy model** contains information about the discovering process that is used by the discovery manager. From this information, the discovery manager is able to track the concerned device or application and start its corresponding proxy. The model contains other information relative to how send to data to a proxy, or receive data form a proxy, especially the data type. From this information, the autonomic manager can connect the endpoints of a mediation chain to the proxies. If a data type is a number (float or integer), the interval has to be provided. An interval is composed of a lowest bound and an upper bound. This information enables the autonomic manager to adapt intervals at runtime by inserting an adaptor between incompatible intervals. For example, if a device provides numbers in the interval [-180, +180], and a connected application task handles numbers in the interval [0, +100], the inserted adaptor does a linear transformation for each value between these two intervals. Figure 2 shows the meta-model of expected proxy models.

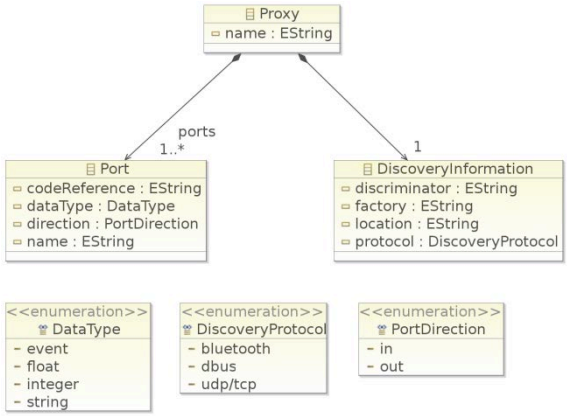

Figure 2. Detailed proxy meta-model

An interaction designer is an expert: she/he is able to detect what is the best way to interact with an application or what is the best way a device can be used. She/he may provide one or several **interaction models** for each proxy model. Because these models only depend on one proxy, they describe only a partial interaction that is completed by the autonomic manager that generates a full interaction at runtime. Interaction models contain information about data semantics, data processing and data path. Without knowing the semantics of a data, the autonomic manager is only able to reason about its type. That prevents it from generating a broken interaction caused by incompatible data types. Guided by semantics, the autonomic manager is able to generate a more useful interaction. Data processing is important: the interaction designer is able to enhance an interaction by adding tasks to an application, synchronizing data and so on. For example, if a media player application proposes a task to control the sound volume trough a number, then the interaction designer can add a task that mute the volume. A data path is the succeeding functions that a data will pass through. Figure 3 shows a detail of the general meta-model about partial interactions.

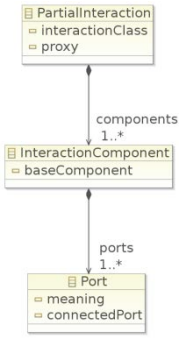

Figure 3. Detailed interaction meta-model

Freely defining semantics does not make sense because the autonomic manager needs to match defined meanings in the different interaction models. Several interaction classes have been predefined. An interaction class defines several meanings that make sense together. An interaction model references one interaction class, so only the meanings of this class can be attached to data of this model.

In order to ease the data processing definition, a predefined library of processing functions is provided. The interaction designer declares which function has to be used, and provides a

configuration for the function. For example, a triggering function sends an event as soon as it receives a value greater than a configured ceiling. These functions are specific to the interaction domain. They have been tailored with reuse concerns in mind. The functions are implemented by components. Thus, the interaction designer specifies a partial interaction by declaring which base component to use, configuring them and binding their ports together. At this stage of specification, data types can generally be ignored because the autonomic manager will be able at runtime to infer each port data type. This inference leads to complete the component configuration, and add data type converting component if necessary.

Specifying a partial interaction by assembling and configuring several domain specialized components eases the interaction designer work. This approach maximizes the reusability by providing generic components, lowers the difficulties by hiding implementations details, and facilitates the implementation of autonomic manager because the abstraction gap between components and mediators is narrow.

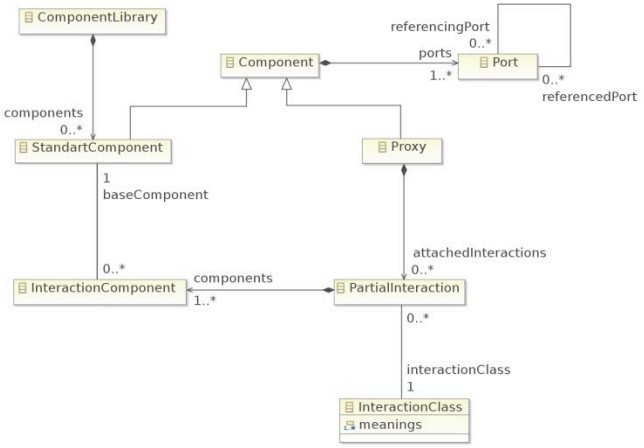

Figure 4. General meta-model used by the autonomic manager

These models are conformed through instantiation to their corresponding meta-model. The autonomic manager relies on a general meta-model that integrates these two meta-models. Figure 4 shows an excerpt of this general meta-model, notably relation between proxies, partial interactions and component library.

We have presented two main meta-models used by DynaMo. The next section provides example of some model specifications.

4. EXAMPLE

The following example shows how the autonomic manager deals with two devices and one application. The application is a media player software, namely *VLC*[1]. The first device is a Blu-ray remote control, namely *BD Remote Control* (or *BDRC*). The second device is a controller for a video game console, namely *Wii Remote*[2] (or *Wiimote*). *VLC* can receive commands through an inter-process communication system, namely *D-Bus*[3]. The two devices use the *Bluetooth* protocol[4] to send data. The Figure 5 shows the proxy models. Only an excerpt of actual proxy and interaction model are shown: *VLC* proxy model has *stop*, *next* and *previous* tasks, etc.

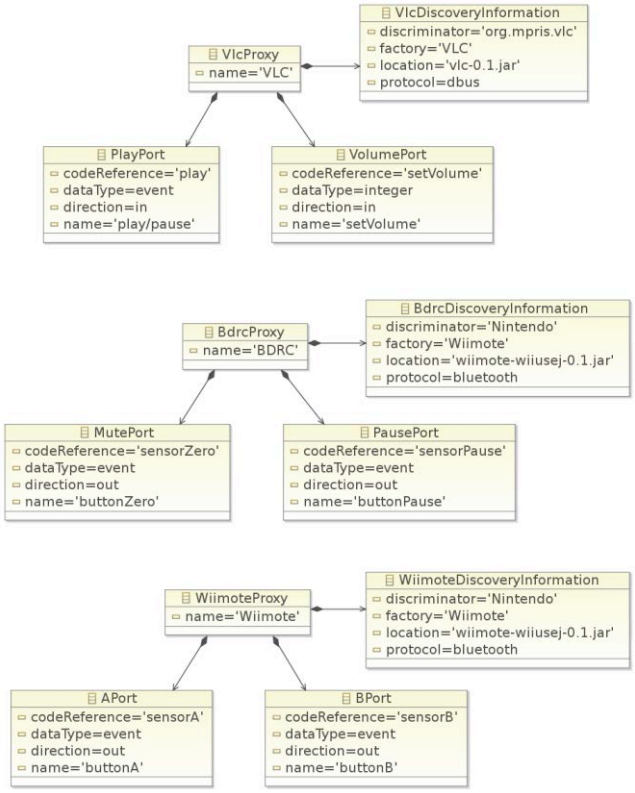

Figure 5. Excerpt of the proxy models

The example follows this scenario: "Alice has previously used DynaMo. She wants to watch a film. She starts *VLC* and activates her *BDRC*. Bob comes and talks to Alice, so she pauses the film. Later, when the *BDRC* is almost run out of energy, she activates her *Wiimote*. Finally, she found the film boring, so when Bob comes to ask something, she just mutes the sound volume to answer."

In the autonomic manager point of view, it receives a discovery notification about *VLC*, hence it download the *VLC* binary proxy from the repository and start it. Since no device is discovered, no mediation chain is generated. Then, it receives a discovery notification about the *BDRC*. It starts the *BDRC* proxy. Now, a mediation chain can be generated. Amongst the interaction models of *BDRC* and *VLC*, it selects the two that have the same interaction class. It instantiates the components declared in the interaction models, and binds mediator of each interaction if their meanings matches. When Alice pushes the pause button, a data is sent by the *BDRC*. The proxy gets the data and passes an event to a mediator. In this mediator, we could eventually notify the autonomic manager that the button has been used. The event follows a path through the mediation chain, and arrives in the pause port of the *VLC* proxy. The proxy calls the pause task on *VLC*. As soon as the autonomic manager is notified of the *Wiimote* discovery, it starts the proxy. Any interaction class of *VLC* and *Wiimote* matches each others. The autonomic manager generates the same mediation chain plus a part that connect the *Wimote* proxy to the *VLC* proxy. This new part is created only from information about data type. Finally, when the *BDRC* runs

[1] Official website: http://www.videolan.org/vlc/

[2] Wikipedia article: http://en.wikipedia.org/wiki/Wii_Remote

[3] Official website: http://www.freedesktop.org/wiki/Software/dbus

[4] Official website: http://www.bluetooth.com/

out of energy, the autonomic manager is notified, and generates the same mediation chain without the *BDRC* part.

Excerpts of partial interaction models are shown in the Figure 6. The matching interaction class is "MediaPlayer". Its meaning set contains "pause" and "mute". Since only components can be declared in the interaction models, attaching a meaning is done by declaring an "identity" component, and by attaching a meaning to a port of the component. Since same meanings are employed in each model, the autonomic manager is able to bind directly these ports.

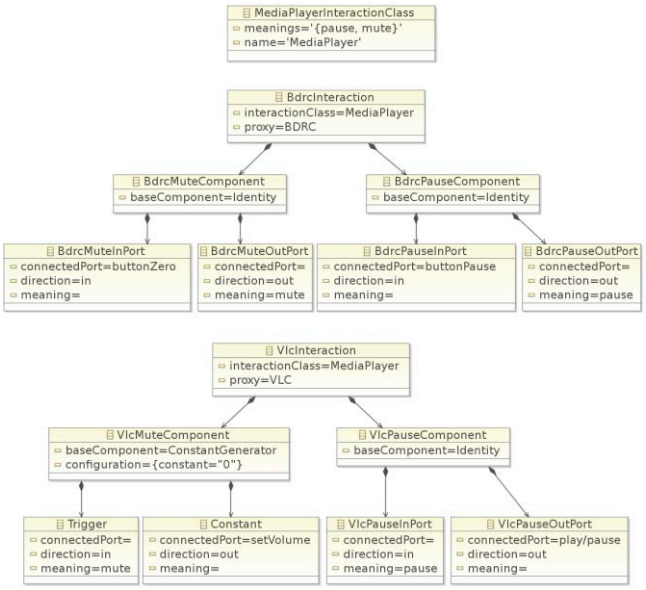

Figure 6. Excerpt of the partial interaction models

The generated mediation chain is shown in the Figure 7. The identity components declared in the interaction models are not apparent in the mediation chain. Since the "identity" component does not modify data that pass through it. They are removed at the end of the generation process. The *Wiimote* proxy model does not have an interaction model that use the "MediaPlayer" interaction class. This lack of information results in random bindings between *Wiimote* proxy ports and *VLC* ports. Of course, the autonomic manager verifies the data type compatibility. Moreover, it distributed the bindings between application tasks, to prevent that only one task is bound to all sensors.

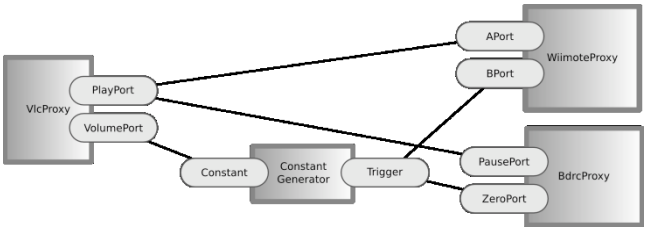

Figure 7. Excerpt of the generated mediation chain

For now, a user can choose between two predefined interaction policies: "simple" and "bind all". If the "bind all" policy is chosen, then the autonomic manager would try to connect all available sensors to tasks. In this example, the *Wiimote* has more sensors that *VLC* has tasks. With the "simple" policy, each task is commended by one *Wiimote* sensor; and some buttons have no effect. With the "bind all" policy, each task is commended by one or more *Wiimote* sensors; and every buttons are bound to a task, thus two buttons can commands the same task. Because only excerpt are shown, this is not viewable in the Figure 7.

5. Conclusion

In this paper, we have shown how we leverage autonomic computing principle to build context-adaptable multimodal interactions. Heterogeneity and dynamicity of pervasive environments are handled. We have created DynaMo, a software that generates and maintains interactions. Generation and maintenance are realized by an autonomic manager, which reasons with models. At the conception time, these models are specified by developers or interaction designers, according to their knowledge. At runtime, a user is able to choose a policy to guides interaction building towards its preference. An example with two existing devices and a real application has been explained.

Our overall architecture has already enables to adapt an interaction to the context. Some interesting works remain, that should easily take place in this architecture. For example, collecting data inside a mediation chain should enable DynaMo to analyze user's usages of interactions, in order to propose new adaptations to the user.

6. REFERENCES

[1] Escoffier, C., Hall, R. S. and Lalanda, P. *iPOJO: an Extensible Service-Oriented Component Framework*. IEEE International Conference on Services Computing (SCC), pages 474–481, 2007.

[2] Garcia, I., Pedraza, G., Debabbi, B., Lalanda, P.and Hamon, C. *Towards a service mediation framework for dynamic applications*, IEEE APSCC, 6-10 december, 2010, Hangzhou, China

[3] Lalanda, P., Bellissard, L. and Balter, R. *Asynchronous Mediation for Integrating Business and operational Processes*. IEEE Internet Computing, vol. 10, no. 1, 2006, pp. 56–64

[4] Papazoglou, M. P. and Georgakopoulos, D. *Service-Oriented computing: Introduction*. Commununications of the ACM, 46 (10):24–28, October 2003

[5] Salehie, M. and Tahvildari, L. *Self-adaptive software: Landscape and research challenges*. ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 4, no. 2, 2009, pp. 1–42

[6] Weiser, M. *The computer for the 21st century*. Scientific American, 265(3):66-75, September 1991.

[7] Wiederhold, G. *Mediators in the Architecture of Future Information Systems*. Computer, vol. 25, no. 3, 1992, pp. 38–49

[8] Wiederhold, G. and Genesereth, M. *The Conceptual Basis for Mediation Services*. IEEE Expert, vol. 12, no. 5, 1997, pp. 38–47