Adaptable Multimodal Interfaces in Pervasive Environments

Pierre-Alain Avouac, Philippe Lalanda and Laurence Nigay
Université Joseph Fourier Grenoble 1
Laboratoire d'Informatique de Grenoble, LIG
Grenoble, France
{pierre-alain.avouac, philippe.lalanda, laurence.nigay}@imag.fr

***Abstract*—In the context of pervasive environments, multimodal interaction plays a pivotal role because multimodality provides flexibility and naturalness for interaction. The challenge of multimodal interfaces in pervasive environments is then to build reliable and autonomic processing systems able to analyze and understand multiple interaction modalities and reconfigure itself in real-time. Addressing this issue, we have developed an autonomic framework called DynaMo (Dynamic multiMOdality) for the development and runtime management of multimodal interaction in pervasive environments. DynaMo is composed by a specification language dedicated to the multimodality domain and a runtime machine that instantiates these specifications. In this paper, we present the overall architecture of our solution DynaMo that is based on partial interaction models, and how these models are completed at runtime to build multimodal interfaces adapted to the local execution environment.**

Keywords-pervasive computing; multimodal interaction; autonomic computing

I. INTRODUCTION

Pervasive computing is slowly changing the way we interact with computers [12, 15] and is gaining more and more attention from industrial and academic sectors. This computing domain relies on the use of smart communication-enabled devices integrated in our environment in order to provide humans with added-value services. Research is particularly active in domains such as smart homes or intelligent buildings where societal needs must be addressed. The purpose here is to assist us in our daily activities in a natural and non-intrusive fashion. For instance, monitoring devices can be used to allow disabled or elder people to stay safely in their home longer. Similarly, intelligent devices can be used to make our working environment more dedicated and efficient. For instance, rendering devices can help visitors to follow the right directions in an unknown building.

Pervasive devices are today becoming smaller and smarter. They fade away in the environment and appear as potential services rather than concrete hardware devices. They have the ability to communicate with each other, perform context-based cognitive and physical actions, and manage themselves in order to stay operational. Weiser's exciting vision [15] where myriads of devices team up transparently to provide human beings with services of all sorts seem now very reasonable! In this context of pervasive environments, input multimodal interaction plays a pivotal role because it provides flexibility and naturalness for interaction [10]:

- Multimodality allows the users to use a variety of devices to interact with an application, depending on the context (e.g. devices availability, reliability, user's mood, etc.).
- Multimodality provides a natural way to interact with device-stuffed pervasive environments by means of various interaction modalities including gestures or direct manipulation.

In [9], we define an input interaction modality as the coupling of a device d with an interaction language l: (d, l). A physical device is an artifact manipulated by the user that acquires (input device) information. An interaction language defines a set of well-formed expressions that convey meaning. The interaction language corresponds to the abstraction function that starts from raw data acquired from a device from which a meaningful task is defined that will be executed by the application. This abstraction function implies a sequence of operations, including multimodal fusion, syntactic and semantic alignments.

A single interaction device can be involved in several modalities. For instance a wiimote device can be used to define a gesture modality m1=(wiimote, gesture recognition) or a direct manipulation way of interacting m2=(wiimote, direct manipulation). Multimodality therefore does not imply multiple devices but multiple modalities, a modality being defined as a couple (d, l). On the one hand, several modalities can be defined for performing a single task. In this case, multimodality offers the flexibility required in pervasive environments: different modalities can be used by the user for performing a given task according to the context. On the other hand, several modalities can be used in a combined way more naturally and robustly: a seminal example of combined usage of modalities is the "put that there" paradigm combining speech and gesture [3]. In [5], we defined the CARE properties as a simple way of characterizing and assessing these aspects of multimodal interaction: the Complementarity, Assignment, Redundancy, and Equivalence that may occur between the interaction modalities available in a multimodal user interface.

Based on these definitions of an interaction modality and of multimodal interaction, we present in this paper the overall architecture of an autonomic framework, namely DynaMo, for

the development and runtime management of multimodal interaction in pervasive environments. We then illustrate the autonomic management of multimodal interaction by considering a simple scenario. The DynaMo framework leverages recent advances in model-based engineering and in service-oriented components [6].

II. MULTIMODALITY IN SERVICE-BASED ENVIRONMENTS

A major problem is that developing multimodal added-value services through the opportunistic and correct integration of volatile, heterogeneous elements is still a major challenge in software engineering. Current techniques can hardly face open environments where devices and applications appear or disappear at anytime. For that reason, the emergence of Service-Oriented Computing (SOC) has brought considerable expectation in the pervasive field [11]. The very purpose of this reuse-based approach is to build applications or interactions through the late composition of independent software elements, called services. Their capabilities are published at runtime and are subsequently discovered, chosen and called when needed. This is achieved within Service-Oriented Architectures (SOA) providing the supporting mechanisms for services description, publication, discovery, and invocation. Service orientation has distinct benefits for pervasive computing. It promotes weak coupling between consumers and providers, reducing dependencies among composition units. Late binding improves adaptability and substitutability favors runtime optimization. Several SOA implementations exist. For instance, Web Services (www.w3c.org) represent a solution of choice to expose software applications, UPnP (www.upnp.org) and DPWS (docs.oasis-open.org) are commonly used to implement volatile devices and even 6LoWPAN (www.6lowpan.org) present some service-like capabilities. Integrating different SOAs is admittedly complex since they rely on different description languages, notification mechanisms, invocation styles, etc. Dynamicity is obviously another challenging point. Since devices and applications join and leave the network at unpredictable times, compositions must be contextual and cannot rely on static orchestrations determined at design time. A major difficulty is that context-aware compositions must be resolved by the composite application itself since, in pervasive environments, it is not conceivable to rely on advanced user intervention. Finally, as of today's state-of-the-art, service composition cannot be based only upon service specifications. Syntactic compatibility does not ensure semantic compatibility. In practice, service composition is based on unexpressed assumptions/rules allowing us to attain the expected results.

As illustrated by Fig. 1, in this context of service-oriented environments, multimodal interaction is a truly illuminating case. Indeed multimodal interaction requires us to dynamically bind service-based interaction devices such as a mobile phone or a remote controller and service-based applications such as a game or a media player. Composition is context aware in the sense that it relies on the available interaction devices and on the currently running applications. The situation can change anytime. It is not possible to anticipate all the eventualities at design time. Multimodal interaction should be designed to dynamically adapt easily to different computing and interaction contexts, user profiles and application needs.

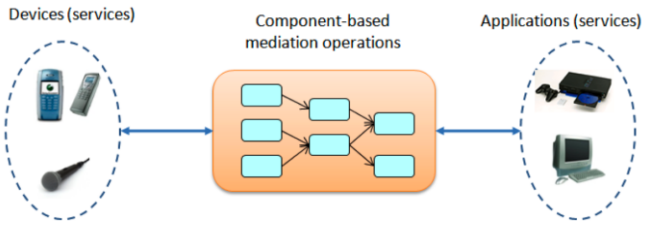

Figure 1. Multimodality in service-based environments: linking service-based devices with service-based applications.

Runtime adaptability has always been a daunting challenge. It requires us to prepare adaptation points in the code, to define a language to specify desired adaptations and to dispose of an adaptable runtime support. Since users are not supposed to play any kind of administrative role, some level of intelligence is clearly needed in the code to decide when, how and where to adapt multimodal interfaces. In addition, interaction adaptations should remain largely transparent to users. As such, necessary management operations should require minimum human intervention, while meeting specific performance and dependability constraints. However, some level of feedback is still necessary to be sure of users' acceptance and understanding.

Modern approaches tend to encapsulate mediation in a dedicated software layer. This is good engineering practice since it provides an isolation layer with a single point of access. It also reduces the number of connections needed and facilitates change management. Specific component-based frameworks have been recently proposed to support the development of mediation operations for multimodal interfaces [4, 14]. These approaches bring appropriate separation of concerns, clearly distinguishing functional aspects like data fusion and non-functional aspects like communication or synchronization. The existing frameworks provide a graphical editor in order to define the sequence of transformations/operations from data acquired from interaction devices to tasks supported by the applications. The developer defines tailored or generic components that are managed by the framework (i.e. the mediation operations of Fig. 1) while the designer graphically assembles the components to define the multimodal interaction. Fig. 2 presents an example of a graphically specified multimodal interaction using OIDE [13]: the example involves the combined usage of speech and gesture for performing a zoom task on a map displayed on an augmented table.

The existing multimodal frameworks, however, are made for well-delimited environments where applications to be controlled and interaction devices to be used are known in advance. They cannot handle highly dynamic environments where devices, applications, and the way multimodal interactions unfold, are rapidly evolving. More dynamic features are needed both at the design language level and the runtime execution framework level. We address these issues in our DynaMo (Dynamic multiModality) framework by adopting an autonomic approach for managing the multimodal processing system. Based on partial interaction models, the DynaMo autonomic manager builds complete multimodal interaction based on runtime conditions.

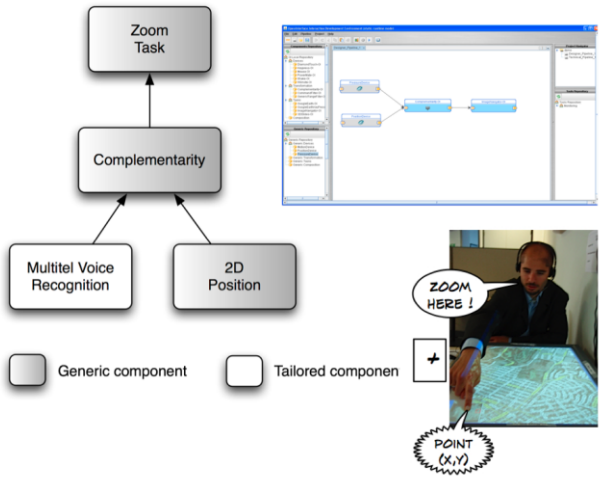

Figure 2. Combined usage of speech and gesture for performing a zoom task on a map displayed on an augmented table: designer assembly for specifying a zoom task and screenshot of the same assembly in the OIDE (adapted from [13]).

III. DynaMo overall architecture

DynaMo is a framework for the development and autonomic management of multimodal interfaces in service-based pervasive settings. Autonomic, here, means that management decisions are taken and realized by the framework itself. DynaMo relies on three main constituents that are presented in Fig. 3 and described in detail in [1]:

- A service integration platform: The purpose of this platform is to provide a flexible, context-aware execution machine. It is based on iPOJO, a dynamic service-oriented component framework built on top of OSGi. The goal of this platform is also to monitor the environment in order to trace any computing evolution.
- A lightweight component-based mediation framework allowing the execution of multimodal processing: This component framework, specific to mediation, provides abstract lifecycle management facilities and is fully dynamic (i.e., adaptable at runtime). Low-level technical aspects like synchronization are hidden away by the model.
- A model-based autonomic manager whose purpose is to build and maintain multimodal interaction at runtime: To make its decisions, the manager uses partial interaction models defined by interaction experts and contextual information provided by the execution machine. It builds multimodal interfaces through the composition of pre-defined components conforming to the component model mentioned above.

This architecture clearly separates the management of the dynamic computing infrastructure and the management of the multimodal processes. Of course, the mediation-specific component model plays a central role. It bridges the gap between high level models specifying interaction possibilities in abstract terms and low level considerations related to the execution platform. It puts in place a generative approach where abstract directives, expressed by the autonomic manager, are transformed into dynamic OSGi Java code.

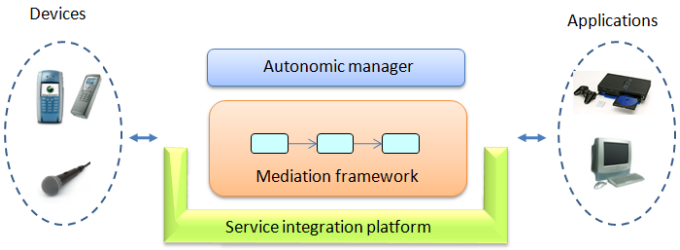

Figure 3. DynaMo overall architecture.

The execution machine is a generic platform providing the necessary runtime support for the execution of dynamic, context-aware applications. The platform is built on top of OSGi (www.osgi.org) and iPOJO [6]. OSGi provides the base mechanisms for modularity and dynamicity. iPOJO is the Apache service-oriented component model. It facilitates the development of dynamic component-based applications on top of OSGI through, in particular, the management of service dependencies and component lifecycle. This platform also integrates a specific module, called ROSE, whose purpose is to constantly reflect the state of the computing environment in the execution machine. ROSE [2] captures services in the computing environment and reifies them as iPOJO components (proxies) in a local registry. ROSE currently handles a number of protocols, including Web Service, DPWS, UPnP, Zigbee and Bluetooth. It is available on ObjectWeb (wiki.chameleon.ow2.org/xwiki/bin/view/Main/Rose). OSGI, iPOJO, and ROSE are largely used and validated in industrial applications (Schneider Electric and France Telecom in particular).

The mediation framework of DynaMo is called Cilia [7]. It is built on top of iPOJO and takes the form of a domain-specific component model. Such a model defines a language to specify components, a language to assemble these components and an execution framework. In Cilia, components are called mediators whereas components assemblies are called mediation chains. They can be both defined in a specification file with an XML syntax.

The DynaMo autonomic manager creates and adapts the multimodal interaction, using the dynamic capabilities of the underlying component model (Cilia). It is driven in its decisions by high level goals set by the users (or by an initial administrator). The manager contains the domain-specific knowledge needed to create and update multimodal processing chains. To make knowledge explicit, architectural models have been recently used [8] as a basis for system construction and update. This approach is limited to domains where reference architectures can be defined. This is not the case that we address because of the high dynamism of pervasive systems. Indeed mediation chains cannot just be instanciated, in conformance with a reference architecture, because all possible situations cannot be anticipated at design time (even if using variability mechanisms). We have defined an alternative approach where the autonomic manager manipulates a number

of models that have to be composed in order to make up a complete, accepted interaction means. Two kinds of models have actually been defined: proxy models and interaction models. The autonomic manager therefore relies on a general meta-model that integrates the proxy and interaction meta-models. Fig. 4 illustrates our model-based manager.

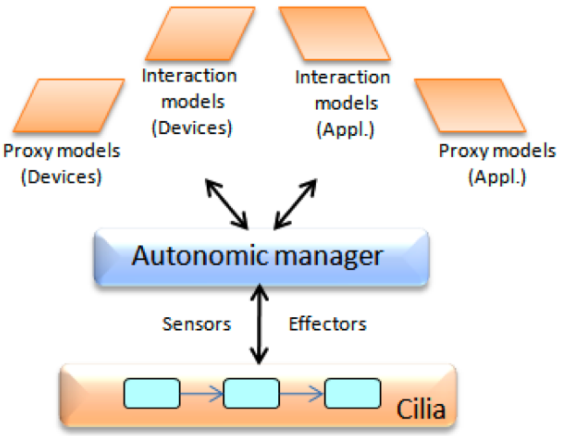

Figure 4. DynaMo model-based autonomic manager.

Proxy models are defined by developers while interaction models are designed by interaction experts.

Proxy models designed are attached to devices or applications. They express how « remote » services can be reified as « internal » services (proxies) in the execution machine. They contain information used to track the interesting remote services and to create the corresponding proxies. Proxy models also specify the discovery protocols to be used, the available ports and their type. Based on this information, the autonomic manager can create proxies and bind them to endpoints of mediation chains. It can also handle simple syntactic alignments. For instance, when dealing with interaction devices providing numbers, adaptors are often necessary to align the provided values and the ones expected by the applications. If a device provides values in [-180, +180] and an application needs values in [0, +100], then an adaptor is automatically added to perform linear transformation.

Interaction models are defined by interaction experts. They specify an interaction possibility for a proxy model, that is to say a way for a device or an application to be used in an interaction. An interaction model only describes a partial interaction that has to be completed by the autonomic manager. Interaction models take the form of mediation chains abstractly expressed. This means that the precise mediation chain to be used do not have to be specified. It is the purpose of the autonomic manager to find out the best suited assemblage of mediators at runtime. Interaction models contain information about data semantics, data processing (mediator class) and data path (bindings). Generic mediators specific to multimodal processing can be directly inserted in the interaction models, with a given configuration. For instance we defined a generic mediator for performing fusion that corresponds to the Complementarity component of Fig. 2.

Semantics-related knowledge is important for the autonomic manager in order to go beyond type alignments. In order to allow minimal semantics matching, we have defined a small ontology shared by all 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 [1], we provide an example that illustrates the interaction class named MediaPlayer. In the following example we use the GamePad interaction class.

IV. EXAMPLE

We consider a simple scenario example that illustrates the appearance of an interaction device and therefore a new modality. In terms of multimodal interaction, the example illustrates a case of equivalence of modalities as defined by the CARE properties for multimodal interaction [5].

The described example corresponds to the following usage scenario: “Alice is going to play a sudoku game (KSudoku, http://games.kde.org/game.php?game=ksudoku). A TV remote controller (BD Remote Controller, or BDRC) and a video game console controller (Wii Remote, or Wiimote) are present in her environment. Sitting in the sofa, Alice starts KSudoku in order to play. She controls the game using the BDRC that is activated. After a while, she notices the Wiimote near her. She grabs it, activates it, and plays with one device in each hand.” In this scenario, Alice does not have to explicitly inform the system that the application has been started. A mediation chain is generated as soon as DynaMo is informed that an application or a device is present.

We now explain what needs to be done for this scenario running using DynaMo. Three proxy models are defined by developers, respectively for the KSudoku application and for the two devices, BDRC and Wiimote. KSudoku is an existing application that we reused. It is not an ad-hoc application that we developed for making the scenario run. The application can be accessed through an inter-process communication, the D-Bus protocol (http://dbus.freedesktop.org/). A simple proxy and its model are created by a developer for the application, without adding any code to the existing application. Indeed DynaMo already manages the D-Bus discovery. A proxy requires only a very small amount of code. Each task of the application has a method, and two lifecycle-related methods (start and stop) have to be implemented. For simplicity, we consider here only a subset of the tasks supported by KSudoku. Four tasks are considered: *selectValue* that takes an integer (the number [1, 9] to be added in a cell of the Sudoku), *enterValue* that takes an event (a confirmation to enter the specified number in the current selected cell of the Sudoku), and the two tasks *moveUp* and *moveDown* that take an event (for changing the selected cell of the Sudoku). We illustrate the proxy model of KSudoku in Fig 5. The same approach as for the application is used for the two devices. We defined a simple proxy (access to the device via the Bluetooth protocol) and a model for each of the two devices. For simplicity, we only consider some of the sensors (buttons, accelerometers) of the TV remote controller (BDRC) and of the Wiimote.

At this stage, without further information, the autonomic manager will define random bindings between the KSudoku proxy ports and the ones of the two device proxies. The autonomic manager will nevertheless verify the data type compatibility.

Instead of a blind generated mapping between the KSudoku application and the two devices, interaction models can be defined to guide the autonomic manager in order to generate adequate multimodal interaction. An interaction model defines interaction possibilities for a proxy model. This scenario illustrates the GamePad interaction class. We defined three interaction models of Ksudoku, Wiimote and BDRC related to this interaction class. Fig. 6 shows an excerpt of the corresponding KSudoku interaction model.

Driven by these interaction models, Fig. 7 shows an excerpt of the mediation chain generated by the autonomic manager for the two tasks *enterValue* and *selectValue*. Based on the same mechanism, for the two other tasks *moveUp* and *moveDown*, the mediation chain will specify that:

- The two buttons padDown and padUp of BDRC are linked to the two tasks *moveUp* and *moveDown,* because they correspond respectively to the meanings *up* and *down* of the GamePad class.
- For the Wiimote, vertical movements with the Wiimote change the selected cell in the Sudoku because the interaction model of the Wiimote declares that the accelerometer coupled with gesture recognition has the two meanings *up* and *down* of the GamePad class.

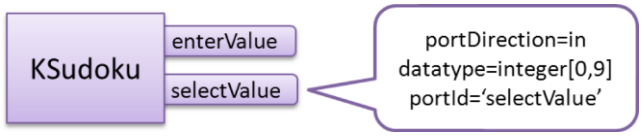

Figure 5. Excerpt of KSudoku proxy model.

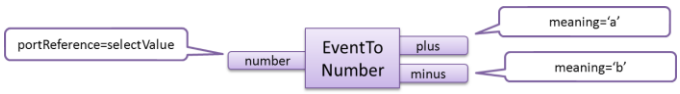

Figure 6. Excerpt of the KSudoku interaction model.

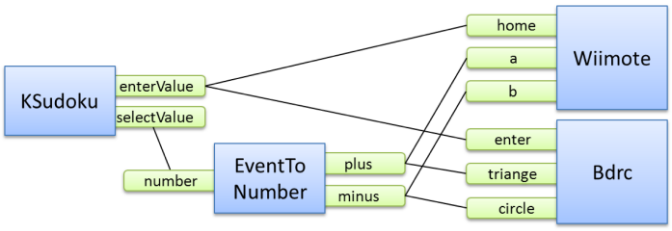

Figure 7. Generated mediation chain for the KSudoku, Wiimote and BDRC interaction.

V. CONCLUSION

In this paper we presented the overall architecture and an illustrative example of the autonomic DynaMo framework for the development and runtime management of multimodal interfaces in pervasive environments. The DynaMo architecture makes a clear distinction between the management of the dynamic computing infrastructure and the one of the multimodal interaction. This distinction allows us to identify two distinct roles while using our framework: the developers that define the proxies and the interaction designer that design partial interaction models at a high level of abstraction without considering implementation details. Moreover the underlying execution machine is robust and validated in industrial applications. As further work, we plan to perform experimental evaluation with users.

REFERENCES

[1] P-A. Avouac, P. Lalanda, and L. Nigay, "Service-Oriented Autonomic Multimodal Interaction in a Pervasive Environment", Proc. of ICMI'11, ACM Press, 2011, in press.

[2] J. Bardin, P. Lalanda, and C. Escoffier, "Towards an Automatic Integration of Heterogeneous Services and Devices", Proc. of APSCC'10, IEEE Computer Society, 2010, pp. 171-178

[3] R. A. Bolt, "Put-that-there: voice and gesture at the graphics interface", Computer Graphics, vol. 14, issue 3, ACM Press, 1980, pp. 262-270.

[4] J. Bouchet, L. Nigay, and T. Ganille, "ICARE software components for rapidly developing multimodal interfaces", Proc. of ICMI '04. ACM Press, 2004, pp. 251-258.

[5] J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, and R.M Young, "Four easy pieces for assessing the usability of multimodal interaction: the CARE properties", Proc. of INTERACT'95, Chapman and Hall, 1995, pp. 115-120.

[6] C. Escoffier, R. S. Hall, and P. Lalanda, "iPOJO: an Extensible Service-Oriented Component Framework", Proc. of SCC'07, IEEE Computer Society, 2007, pp. 474-481.

[7] I. Garcia, I., G. Pedraza, B. Debbabi, P. Lalanda, P., and C. Hamon, "Towards a service mediation framework for dynamic applications", Proc. of APSCC'10, IEEE Computer Society, 2010, pp. 3-10.

[8] D. Garlan, S. Cheng, A. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure", Computer, vol. 37, issue 10, IEEE Computer Society, 2004, pp. 46–54.

[9] L. Nigay and J. Coutaz, "A Generic Platform for Addressing the Multimodal Challenge", Proc. of CHI'95, ACM Press, 1995, pp. 98-105.

[10] S. Oviatt, "Multimodal interfaces", Chap. 14 in Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications, L. Erlbaum Assoc. Inc., 2007, pp. 286-304.

[11] M. P. Papazoglou and D. Georgakopoulos, "Service-Oriented Computing: Introduction", Com. of the ACM, vol. 46, issue 10, ACM Press, 2003, pp. 24-28.

[12] M. Satyanarayanan, "Pervasive computing: vision and challenges", IEEE Personal Communications, vol. 8, IEEE Computer Society, August 2001, pp. 10-17.

[13] M. Serrano, D. Juras, and L. Nigay, "A Three-dimensional Characterization Space of Software Components for Rapidly Developing Multimodal Interfaces", Proc. of ICMI'08, ACM Press, 2008, pp. 149-156.

[14] M. Serrano, L. Nigay, J-Y. Lawson, A. Ramsay, R. Murray-Smith, and S. Denef, "The openinterface framework: a tool for multimodal interaction", Proc. of CHI EA '08, pp. 3501-3506.www.oi-project.org.

[15] M. Weiser, "The computer for the 21st century", Scientific American, vol. 265, issue 3, NPG, 1991, pp. 66-75.