Coordination and Adaptation Techniques for Software Entities

Carlos Canal[1], Juan Manuel Murillo[2], and Pascal Poizat[3]

[1] University of Málaga, Spain
canal@lcc.uma.es
[2] University of Extremadura, Spain
juanmamu@unex.es
[3] Université d'Évry Val d'Essonne, France
poizat@lami.univ-evry.fr

Abstract. The ability of reusing existing software has always been a major concern of Software Engineering. The reuse and integration of heterogeneous software parts is an issue for current paradigms such as Component-Based Software Development, or Coordination Models and Languages. However, a serious limitation of current approaches is that while they provide convenient ways to describe the typed signatures of software entities, they offer a quite limited support to describe their concurrent behaviour. As a consequence, when a component is going to be reused, one can only be sure that it provides the required interface, but nothing else can be inferred about the behaviour of the component with regard to the interaction protocol required by its environment. To deal with this problem, a new discipline, Software Adaptation, is emerging. Software Adaptation promotes the use of adaptors-specific computational entities guaranteeing that software components will interact in the right way not only at the signature level, but also at the protocol and semantic levels. This paper summarizes the results and conclusions of the First Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'04).

1 Introduction

In the recent years, the need for more and more complex pieces of software, supporting new services, and for wider application domains, together with advances in the net technology, have promoted the development of distributed systems. These applications are made up of a collection of interacting entities (either considered as subsystems, objects, components, or —more recently— Web services) that collaborate to provide some functionality.

One of the most complex tasks when designing such applications is to specify the coordinated interaction that occurs among the computational entities. This fact has favored the development of a specific field in Software Engineering devoted to the Coordination of software. Some of the issues addressed by such discipline are:

1. To provide the highest expressive power to specify any coordination pattern. These patterns detail the order in which the tasks developed by each component of the distributed application have to be executed. The state of the global computation determines the set of tasks that can be performed at each instant.
2. To promote reusability, both of the coordinated entities and of the coordination patterns. The coordinated entities could be used in any other application in which their functionality is required, apart from the coordination pattern that directs them. The same holds for the coordination patterns; they could be used in a different application, managing a different collection of entities with different behaviour and different interfaces, but with the same coordination needs.

In fact, the ability of reusing existing pieces of software has always been a major concern of Software Engineering. In particular, the need for the reuse and integration of heterogeneous software parts is at the core of Component-Based Software Development. The paradigm "write once, run forever" is currently supported by several component platforms.

However, a serious limitation of available component platforms (with regard to reusability) is that they do not provide suitable means to describe and reason on the interacting behaviour of component-based systems. Indeed, while these platforms provide convenient ways to describe the typed signatures of software entities via Interface Description Languages (IDLs), they offer a quite limited and low-level support to describe their concurrent behaviour. As a consequence, when a component is going to be reused, one can only be sure that it provides the required interface, but nothing else can be inferred about the behaviour of the component with regard to the interaction protocol required by its environment.

To deal with this problem, a new discipline, *Software Adaptation*, is emerging. Software Adaptation focuses on the problems related to reusing existing software entities when constructing a new application, and promotes the use of adaptors —specific computational entities for solving these problems. The main goal of software adaptors is to guarantee that software components will interact in the right way not only at the signature level, but also at the protocol and semantic levels. In this sense, Software Adaptation can be considered as a new generation of Coordination Models.

However, Software Adaptation is not restricted to the adaptation of the interaction through coordination models. Other functional and non-functional properties of software systems can be adapted as well, while other topics such as automatic generation of adaptors are under study. With the aim of stating the boundaries and interests of Software Adaptation, the *First Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'04)* was held in conjunction with the 18th European Conference on Object-Oriented Programming (Oslo, Norway, 14-18 June, 2004). The topics of interest of the workshop were:

- New coordination models separating the interaction concern;
- Aspect-oriented approaches to software adaptation;

– Coordination and adaptation in concurrent and distributed object-oriented systems;
– Interface and choreography description of Web services;
– Coordination and adaptation middleware;
– Rigorous approaches to software adaptation;
– Identification and specification of interaction requirements;
– Patterns and frameworks for component look-up and adaptation;
– Automatic generation of adaptors;
– Documenting components to enable software composition and adaptation;
– Metrics and prediction models for software adaptation;
– Extra-functional properties in their relation to coordination and adaptation;
– Tools and environments;
– Industrial and experience reports.

The main conclusion of the workshop is that Software Adaptation is an emerging and well-differentiated discipline motivated by the new challenges of Software Engineering on building complex software systems in an efficient way. This paper summarizes the discussions and conclusions from the workshop.

The rest of this paper is organized as follows: Section 2 presents an outline of the contributions submitted, Section 3 summarizes the results of the three discussion groups that worked during the workshop, and Section 4 presents the conclusions of the workshop. Finally, we provide some references to the work on coordination and adaptation being developed by workshop attendants.

2 Summary of the Contributions

According to ECOOP'2004 Workshop Guidelines, we tried to put up a working meeting in which participants not merely presented the technical details of their work, but could instead discuss their points of view, trying to identify which are the main issues, challenges, and their possible solutions in the field of Software Coordination and Adaptation.

For these reasons, in the CfP we asked for short (about five pages long) position papers. We also recommended that papers should contain a section on Open Issues, in which authors indicated the relevant issues that they would like to address during the workshop.

We received twelve submissions, covering a wide range of aspects of coordination and adaptation. In particular, many of them dealt specifically with software adaptation, which was rather encouraging, taking into account that this is the first event specifically addressing this field. As workshop organizers, and acting also as program committee, we reviewed the position papers, and replied to the authors with our impressions, suggesting them to deepen in those aspects that we considered more interesting for the workshop. Due to their quality and interest, we decided to accept all the papers submitted, as a way of ensuring different points of view that would produce more lively discussions. Both the Call for Papers and the final versions of the position papers can be found at the Web

page of the workshop:

`http://wcat04.unex.es`

Fifteen participants, coming from seven different countries, attended the workshop, which started with a short presentation (a five-minutes talk followed by five minutes for questions and comments) of ten of the position papers submitted. Once again, the idea was not to give out academic speeches but to present succinctly the positions, saving time for discussions during the workshop.

The list of participants, together with their affiliation, and the title of the position papers they presented is as follows:

- Marco Autili, marco.autili@di.univaq.it
  University of L'Aquila (Italy)
  Automatic Adaptor Synthesis for Protocol Transformation
- Steffen Becker, becker@informatik.uni-oldenburg.de
  University of Oldenburg (Germany)
  The Impact of Software Component Adaptors on QoS Properties
- Carlos Canal, Workshop Organizer, canal@lcc.uma.es
  University of Málaga (Spain)
  On the Dynamic Adaptation of Component Behaviour
- Antinisca Di Marco, adimarco@di.univaq.it
  University of L'Aquila (Italy)
- Viktoria Firus, firus@informatik.uni-oldenburg.de
  University of Oldenburg (Germany)
- Thomas Heistracher, thomas.heistracher@fh-sbg.ac.at
  Salzburg University of Applied Sciences (Austria)
  Pervasive Service Architecture for a Digital Business Ecosystem
- Tobias Ludwig, toby@gmx.de
  ECOOP'2004 Student Volunteer (Germany)
- Claudius Masuch, claudius.masuch@fh-sbg.ac.at
  Salzburg University of Applied Sciences (Austria)
  Pervasive Service Architecture for a Digital Business Ecosystem
- Juan Manuel Murillo, Workshop Organizer, juanmamu@unex.es
  University of Extremadura (Spain)
  Managing Components Adaptation Using Aspect-Oriented Techniques
- José Luis Pastrana, pastrana@lcc.uma.es
  University of Málaga (Spain)
  Client Oriented Software Developing
- Pascal Poizat, Workshop Organizer, poizat@lami.univ-evry.fr
  Université d'Évry Val d'Essonne (France)
  Formal Methods for Component Description, Coordination and Adaptation

– Sibylle Schupp, schupp@cs.chalmers.se
Chalmers University of Technology (Sweden)
How to Use a Library

– Björn Törnqvist, bjorn.tornqvist@bth.se
Blekinge Institute of Technology (Sweden)
On Adaptative Aspect-Oriented Coordination for Critical Infrastructures

– Nesrine Yahiaoui, nesrine.yahiaoui@prism.uvsq.fr
UVSQ Prism (France)
Classification and Comparison of Dynamic Adaptable Software Platforms

– Jie Yang, jie@cs.uit.no
University of Tromsø (Norway)

The papers submitted to the workshop may be classified into three categories. The first one corresponds to papers related with Coordination and Adaptation Models. This category includes the works presented by Autili [1], Canal [3], and Heistracher and Masuch [6]. In [1], the authors propose the automatic generation of adaptors to manage the difficulties of building systems from components, in particular when some changes (improvements, evolution, etc.) must be introduced in the composite system. Their approach is based on formal methods. In [3], an approach to deal with component adaptation at runtime is presented. Once again, his proposal is supported by formal methods. Finally, [6] presents an architecture to support the expression of business models addressing the composition of service chains, and its optimization through automatic self-organizing and evolutionary (genetic) algorithms.

The second category is constituted by works focused on Adaptation and Aspect-Oriented Techniques, including the works presented by Murillo [5], Pastrana [7], and Törnqvist [10]. [5] is focused on how adaptation can be considered a crosscutting concern and, consequently, managed with aspect-oriented techniques. In [7], applying the contract metaphor the authors propose the specification of some non-functional requirements aimed over components by means of connectors (adaptors). The work described in [10] presents the future EU integrated power grid as a real case study of coordination and adaptation. The authors propose the use of coordination, aspect-oriented and adaptation techniques to deal with that system.

Finally, the third category corresponds to papers presenting general studies about Coordination and Adaptation, and includes the works presented by Becker [2], Poizat [8], Schupp [9], and Yahiaoui [11]. [2] presents an analysis on the impact that adaptors may have on the Quality of Service (QoS) of adapted components, showing the interdependencies that exist between QoS attributes, and how adaptors can affect such attributes. [8] contains an interesting study about how formal methods can help on designing and composing/adapting CBSE systems. The authors argue for the pragmatic use of formal methods. In [9] the problems of building and using software libraries are highlighted. These problems are similar to those of searching, retrieving, and adapting components from

repositories. Finally, [11] analyzes the features shown by platforms allowing dynamic adaptation, and classifies these features in several dimensions.

¿From the presentation session, a list of open issues that we would like to address was identified and grouped. This helped to make clear which were the participants' interests, and served also to establish some goals for the workshop. Then, participants were divided into three groups (about 4-6 persons each), attending to their interests, each one related to a topic on software coordination and adaptation. The task of each group was to discuss about the assigned topic, to detect problems and its real causes, and to point out solutions. Finally, a plenary session was held, in which each group presented their conclusions to the rest of the participants, followed by some discussion.

3 Summary of the Discussions

As indicated above, after the short presentations a small brainstorming was performed, trying to determine the most common remarks and questions. These issues were classified into three groups of topics:

Problems. This group dealt with which are the different problems that can be solved by adaptation, and how. The main issues were: what does adaptation mean? (i.e. what is to be adapted: functional, or non-functional properties?); static versus dynamic adaptation (i.e. when to adapt/not to adapt?); and what are the relations between adaptation and coordination, maintenance, and aspect-oriented techniques? (the latter related to the adaptation of non-functional properties).

Languages and processes. This group dealt with the languages and processes that can be used to support coordination and adaptation. The main issues addressed were: which aspects of components to take into account? (either functional, behavioural/protocol, or non-functional properties, such as performance or reliability); how to do it? (e.g. formal methods, contracts, biological metaphors...); what level of detail is needed for describing component interfaces? what are the requirements for connector/adaptor languages? what kind of formalism is required? (and its relation with implementation); is more than one language needed? (for example, using several coordination models, or protocols with value-passing).

Benefits. This group dealt with the benefits that can be expected from the use of coordination and adaptation techniques (apart from solving the adaptation problems). The main issues were: what is the relation between adaptation and quality of service? do adaptation increases the reliability or reusability of components? which measure/assessment mechanisms to use? what about scaling up? and which is the role of tools.

The classification above induced to divide the participants into three different discussion groups. The first group would start by finding out which are the

system properties that can be adapted (the what). Then, the means to achieve their adaptation (the how) must be identified, too. Both (what+how) would lead to a definition of adaptation. Finally, this group would compare the concerns of adaptation with those of other related fields, such as aspect-orientation and maintenance. The second group would find out requirements for coordination and adaptation languages in terms of description expressiveness, methods and processes. The third group would discuss on what could be the by-products of coordination and adaptation processes.

3.1 Interest of Adaptation and Coordination

The discussion group was integrated by Steffen Becker, Viktoria Firus, Juan M. Murillo, Nesrine Yahiaoui, and Jie Yang. The aim of the group was to discuss about what adaptation is, the different kinds of adaptation that can be demanded by a system, and the relationships, differences and similarities between adaptation and other fields such as coordination, maintenance and aspect-orientation. The next paragraphs depict the conclusions reached.

The group started discussing about what is software adaptation in a broad sense (and not restricted to the meaning of adaptation underlying in the works being developed by the people in the group). Thus, thinking about what adaptation is, the first emerging question was what kind of properties can be adapted in a software system. The immediate answer was that, of course, several non-functional properties of the system can be adapted. For instance, the interaction constraints of a system can be adapted using coordination models and languages. Indeed, when a coordination model is used for tuning the interaction protocol between two components, these components are being adapted to work altogether. However, adaptation is not restricted to non-functional properties. The functionality of the system can be adapted as well. A system can be adapted to provide new services, or the functionality provided by a piece of software can be adapted for using it in a different system. For example, a component providing the functionality of a list of elements could be adapted for reusing it in a particular system where the functionality of a queue is required. Thus, thinking about adaptation, both functional and non-functional adaptation must be taken into account.

After that, we tried to classify the different ways to proceed with software adaptation (extending the classification presented by Yahiaoui in [11]). The group agreed that a good criteria to classify adaptation is attending to the moment in which the need of adaptation is detected. Thus, three different moments requiring different procedures for adaptation were catalogued:

1. Requirements adaptation. This kind of adaptation is made when the requirements of a system are extended to meet new properties, or when we want to reuse a system specification, but it must be adapted to meet the requirements of the new system. Examples of this kind of adaptation are adding a security property to a bank account, or adding an attribute to a table in a database. It could require both functional and non-functional adaptation.

2. Static adaptation. This is the case in which we want to adapt pieces of software developed independently in order to make them work together. Suppose a Bank_account component providing a get_money service and a Client component requiring a decrement_balance service. As a general characteristic of this kind of adaptation, it can be said that the steps to proceed with the adaptation are known —and have been planned— in advance with respect to the moment in which the adaptation takes place. Although some kind of functional adaptation could be required, usually, static adaptation involves non-functional adaptation.
3. Dynamic adaptation. Running pieces of software need to be adapted in order to provide some particular service. For example, suppose a mobile phone adapting to the local phone provider of a different company. In comparison with static adaptation, now the components to be adapted are unknown until the moment of the adaptation. Hence, the steps to manage the adaptation are unknown as well. Dynamic adaptation should be limited to non-functional adaptation, if possible[1].

The group also discussed about the procedures to manage adaptation. We agreed that two different procedures can be distinguished:

1. Manual adaptation. The adaptation steps as well as the adaptors are specified and developed by the people intervening in the software process (designers, architects, programmers, etc). However, the adaptation task may be assisted by software tools. Nevertheless, adaptation must be non-intrusive, that is, it must not require a modification of the component adapted (modifying the adapted component would be more related to maintenance than to adaptation).
2. Automatic adaptation. All the adaptation steps and the adaptors themselves are automatically generated by a tool. This kind of tool must be able to detect the need for adaptation and whether the required adaptation is possible or not, and will have to determine the steps to manage adaptation, and finally, generate the adaptors.

Anyway, regardless of the adaptation procedure used, the work presented by Becker [2] —focused on how adaptation can affect the QoS attributes— led us to think that it would be good to have procedures to determine whether the adaptation of a component is convenient (instead of building a new component providing the desired functionality).

Having all the above in mind, it is easy to conclude that there exist some relationships between adaptation and other disciplines such us coordination,

[1] Here, it must be understood that the aim of this kind of dynamic adaptation is not to provide new unexpected functionality to a running component. Thus, in the mobile phone example it is considered that, although the operation of contacting a network provider is a functional property, the way in which the provider is contacted is not a part of the functionality (but the way in which the functionality is provided, instead).

maintenance and aspect-orientation. However, although such relationships do actually exist, adaptation have connotations that provide it with an own identity.

Coordination models and languages can be used to manage the adaptation of the interaction protocols among components. In particular, exogenous coordination models provide entities —commonly called coordinators— with the aim of forcing the interaction protocols among components. Such coordinators can be considered as adaptors. However, Coordination is not concerned with topics such us how the need of adaptation can be detected, how coordinators can be generated automatically and which is the information needed to achieve it, and, of course, with the adaptation of non-functional properties different from interaction.

Adaptation is close to maintenance as well. Maintenance is concerned with how to deal with system evolution in an easy and efficient way. In particular, adaptation can help maintenance by providing the technical means to deal with changes in the requirements. In some sense, the work by Becker [2] analyzing how adaptation can affect the QoS attributes can be related with maintenance. Nevertheless, maintenance is focused on how software methodologies can face system evolution, and on the methods needed to give support to evolution in the software process, more than on the technical means to adapt an existing system.

Finally, aspect-oriented techniques can be used to adapt non-functional properties of a system. As Murillo presented [5], adding new aspects to an existing software system can have the effect of adapting the system to satisfy new properties. However, apart of this co-lateral effect, Aspect-Oriented Software Development is concerned with how crosscutting concerns can be identified and separated at the different stages of the software life cycle getting out of its scope topics such as automatic adaptation or adaptation of functional properties.

The final conclusion of the group was that adaptation is concerned with how the functional and non-functional properties of an existing software entity (class, object, component, etc.) can be adapted to be used in a software system observing that:

- Adaptation must be non-intrusive.
- The need for adaptation can appear at any stage of the software life-cycle, from requirements specification to system operation stages.
- Software adaptation requires automatic or at least semi-automatic (computationally assisted) procedures.

In practice, some kind of adaptation is being done using tools such as exogenous coordination models or aspect-oriented techniques. However, the new challenges in Software Engineering for building complex software systems in an efficient way do require new techniques, dealing with the detection of adaptation needs, automatic adaptation, functional adaptation, formal methods allowing to reason about adaptation, dynamic adaptation, and automatic adaptation. All these topics are the interest of the emerging discipline of Software Adaptation.

3.2 Languages for Coordination and Adaptation

The discussion group was integrated by Marco Autili, Claudius Masuch, Pascal Poizat, and Björn Törnqvist. The goals of this group was to discuss on the requirements of languages and processes to support adaptation.

The first issue to be discussed was if formal languages were needed. The group recognized then that formal methods have great benefits in any component-based framework (tools for subtyping detection, protocol compliance, ...). The most interesting works on component coordination are now based on formal languages of some form (automata, transition systems, process algebras or MSC). Formal languages are emerging in the more challenging (up to now) challenging application domain for coordination and adaptation techniques, Web services. [8] gives some details on these two points. On adaptation, Autili [1] has very interesting results with MSC. Moreover, the seminal paper on adaptation [12] deals with transition systems. Hence the group accepted that the language should be formal. However, it was also said that the specifications should be executable (which is possible if the language has an operational semantics). This is a mandatory requirement for example to be able to relate component and adaptors specifications with their implementations. This is also needed to be able to have automatic or dynamic adaptation.

Then, the group interested himself on what kind of components are to be adapted. Usually components are black boxes with type-only require/provide interfaces given in an Interface Description Language. It has been recognized both in the ADL and Coordination communities that such interfaces are not sufficient: for example two components may have compatible provide/require interfaces but may fail to interact due to incompatible protocols. Therefore, the group advocated for a Behavioural IDL (BIDL) to describe the components, thus yielding a grey box description of them.

The group rejected the possibility to use white box descriptions of components (*i.e.* having the whole description of the component in some language, not only its abstract protocol). This would increase the complexity of the adaptation process, which is an important issue in case of dynamic adaptation. Moreover, such interfaces are to be exchanged between components in such adaptation (see for example the Pastrana proposal [7]). Hence they have to be quite small and abstract.

Adaptation has been recognized by the group as composing in some way (depending on the language structuring means) components and adaptors. But this should yield (semantically) a component in order for the adapted component to be reused in place of the first one. Hence adaptation should yield components. Then a question arose: is the adaptation language the same than the components protocol one? Here there are two possibilities.

The first one is to use the same one (a process algebra or any state transition language is the solution). Hence the basic structuring means of the language can be reused (for example parallel composition in process algebras). This is more simple, adaptors can be components too (to be used and reused as components are). This expresses the need for languages in which compositions are compo-

nents. Moreover, the language could benefit from being compositional. Hence any formal process (verification but also adaptation) could be done on a composition by doing it on its subcomponents (adapting a composed component could be defined as adapting each of its subcomponents). Note that once again abstraction was noted as an important issued here (compositional languages are often the ones with abstraction features).

A second solution is to use a specific language for adaptation (as it can be for coordination using a temporal logic [8] or on adaptation with MSC [1]). This is more expressive as the language has only to have specific hooks on the components protocols but can then extend this. This second solution is much more related to Aspect-Oriented weaving techniques (the adaptation language being used as a description of the joint point).

Then the discussion opened on extensions of the components and/or adaptation language.

A first aspect was some form of state description. The group rejected this for components, or only in a very abstract form (components should be protocol oriented). This is to keep a grey box description of components. However, the interest of states was recognized for adaptors (denoting states of the whole system). In effect, adaptation can depend on a state of the system (Törqvist expressed this on the EU power grid system requirements [10]: different adaptation schemes depending on the power level, etc).

A second aspect was on more expressive communication semantics, mainly queues.

The problem here is that more expressive communication schemes (asynchronous with queues) yield undecidability results. Different queue protocols can be used (for example FIFFO in place of FIFO). However, this question is not so important and adaptation can be more simple at an abstract level (basic synchronous communication) as then lots of tools (such as process algebraic ones) can be used.

Other aspects were then quickly reviewed: non-functional ones. It has been recognized that they are more difficult to be taken into account still keeping in the preceding requirements (formality, abstraction, decidability, tools,...). However, several are interesting such as real time to give boundaries for services or timeouts, probabilities as an information on components for adaptors (again, on the EU power grid case study, possibility for a subsystem of going down), localities. Here a single very expressive language can be used but its verification/adaptation should be difficult. Several languages can also be used (one for each aspect), but then some form of Aspect-Oriented weaving would be needed.

To end the group discussed the use of data. Components may exchange data and may have to be adapted on this. The group decided that data are useful but should be given in a very abstract form in order to make the adaptation possible.

As a conclusion, good candidate languages for adaptation seem to be process algebras (which can be extended for lots of aspects, adaptation in a single for-

malism) and MSC (should be extended if different aspects, adaptation in MSC and protocols in any state transition language).

3.3 Benefits of Coordination and Adaptation

The discussion group was integrated by Carlos Canal, Thomas Heistracher, José Luis Pastrana, and Sibylle Schupp. Its main goal was to find out which are the benefits that coordination and adaptation techniques may have in software development.

However, it is worth saying that the group also discussed more general questions (e.g. the meaning of coordination and adaptation, their boundaries, etc.) that partly overlapped the goals of the other two groups. Anyway, the group tried to give its particular point of view about these issues, which is with no doubt closely related to each participant's position and experience in the field.

The discussions started considering the value of software adaptation and the need of a discipline devoted to it. The participants agreed that since there are lots of examples of adaptation in our daily life, we have to put it in software, too. Where and when? The group discussed if adaptation could be proactive, or if one must wait till designers find a particular solution for adapting a given component to a given context. The conclusion -however, more a wish list than a state-of-the-art description-, was that it should be proactive, in the dynamic, automatic and self-initiated sense. The group also concluded that this may be better achieved by means of some kind of middleware infrastructure, containing built-in adaptation functions, which could decide when and how to adapt. More precisely, adaptation should be performed at binding time, when a component joins a context, rather than being static (i.e. performed at design time). It should be also as much automated as possible.

The work of Schupp [9] deals with the development of large libraries containing different versions of components performing a single well-defined task (e.g. the Fast Fourier Transform). These libraries are implemented in an object-oriented language, and can be instantiated by parameterization, not only of generic types, but also of higher-order functions. Hence, the technology involved is object-oriented and could be described as pre-component-based. The group took this as a sort of case study, a starting point for discussions, and tried to analyse how the techniques and methods related to the scope of the workshop could help.

First, the group considered if this kind of parameterization could be taken as a form of adaptation. The conclusion was it was not. The use of generics -and also polymorphism- in OO languages provide degrees of freedom to the components developed, which can be customized afterwards. However, these degrees of freedom must be foreseen and carefully defined during design, while adaptation should allow forms of reuse and changes in software characteristics that were not thought of by the original developers of a component. Thus, adaptation is more related to the black-box reuse style of CBSD, than to the white-box reuse typical of the OO paradigm.

One of the problems when using these libraries of components is how to choose the right version. Different users may have different criteria. Hence, the problem of how to describe components characteristics is crucial. The clear benefits are that the components would be more accessible, helping in the scalability of the libraries, and also in the predictability of their behaviour.

Consequently, extended interfaces (containing not only signature specifications, describing not only functional properties) are a strong need, as the only way of scientifically evaluate the software, and take decisions. Anyway, an open issue here is that once we inspect the interfaces and decide which component to use, how could we trust it? That is, how could we ensure it behaves as described? Once again, putting this functionality into the middleware could be a solution.

In particular, the work of Pastrana [7] addresses the specification of extended interfaces for components, following the design by contract metaphor. He made the group reflect on the fact that while the description of functionality has been extensively studied, and one has a sound mathematical background for it (for instance, the use of pre- and post-conditions), much less has been done with non-functional (no-fun!) properties; each of them must be defined individually, and described separately using a specific notation for which a similarly sound foundation should be established. If not, the possibilities for automatic dynamic adaptation, and its benefits will always be very limited.

The group discussed also whether one needs to find the best choice or solution, or just one "good enough". Here, the biological simile present in the work of Heistracher [6] helped the group in considering some kind of dynamic natural selection of components, where different solutions may compete for their existence, and even extinguish. This would help by reducing the number of choices in the future.

Finally, the formal background of the work of Canal [3], related to process algebras for behavioural descriptions, made the group also discuss the role and benefits of formal methods. One of the benefits is that abstraction, hiding the irrelevant details, would help in the selection process, and in component trading in general. However, one has to decide which details are important and which are not, and this decision determines the notation used and also the properties that can be analyzed. Once again, the group arrived to the importance of describing non-functional properties (which notations, and which "algebras" to use here?).

Anyway, we agreed that the benefits of formal methods should be really operational, not only descriptive, which is a lack of many proposals in the field. The importance of tools was also discussed: real solutions should provide real tools!

Other envisioned benefits were reduction of complexity: adaptation techniques try to cope with the complexity of giving a solution; increasing reuse: using more technologies with less time spent in integration; and last, but not at all less important, the value added by these techniques to the software process: allowing more technically advanced developers to compete with solutions based only in man-power.

3.4 Conclusions of the Discussions

After the group discussions, the report from each group was presented and the conclusions were discussed to reach agreement. Summarizing the conclusions of the three groups it can be said that Software Adaptation can be identified as a new discipline with its own topics of interest concerned with how the functional and non-functional properties of an existing software entity (class, object, component, etc.) can be adapted to be used in a software system. The need of adapting a software entity can appear at any stage of the software life-cycle and adaptation techniques for all the stages must be provided. Anyway such techniques must be non-intrusive and based on Behavioural IDLs and formal executable specification languages. Such languages and techniques should support automatic and dynamic adaptation, that is, the adaptation of a component just in the moment in which the component joins the context supported by automatic and transparent procedures.

4 Conclusion of the Workshop

After the presentation of the conclusions of the discussion groups the Closing Session was held. During this session attendants were asked for their general impression about WCAT. We all agreed that the workshop was very interesting and productive. We also discussed about the possibility of having a new edition of WCAT in 2005. Again, all the attendants agreed that it would be good to have a new meeting in one year to mature the discussions about Software Adaptation.

The high quality of the papers submitted to the workshop encouraged us as workshop organizers to put up a special issue where selected technical papers about the works of the groups intervening in the workshop could be presented at length. Currently we are in conversations with an international journal interested in the publication of the special issue.

References

1. Marco Autili, Paola Inverardi, and Massimo Tivoli. Automatic adaptor synthesis for protocol transformations. In Canal et al. [4], pages 39–46.
2. Steffen Becker and Ralf H. Reussner. The impact of software component adaptors on quality of service properties. In Canal et al. [4], pages 25–30.
3. Carlos Canal. On the dynamic adaptation of component behaviour. In Canal et al. [4], pages 81–88.
4. Carlos Canal, Juan Manuel Murillo, and Pascal Poizat, editors. *First Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'04). Held in conjunction with the 18th European Conference on Object-Oriented Programming (ECOOP). Published as a Technical Report of the Universities of Málaga (Spain), Extremadura (Spain) and Évry (France). ISBN 84-688-6782-9*, 2004.
5. Y. Eterovic, J. M. Murillo, and K. Palma. Managing component adaptation using aspect oriented techniques. In Canal et al. [4], pages 101–108.

6. Thomas Heistracher, Thomas Kurz, Claudius Masuch, Pierfranco Ferronato, Miguel Vidal, Angelo Corallo, Gerard Briscoe, and Paolo Dini. Pervasive service architecture for a digital business ecosystem. In Canal et al. [4], pages 71–80.
7. M. Katrib, J. L. Pastrana, and E. Pimentel. Client oriented software developing. In Canal et al. [4], pages 9–16.
8. Pascal Poizat, Jean-Claude Royer, and Gwen Salaün. Formal methods for component description, coordination and adaptation. In Canal et al. [4], pages 89–100.
9. Sibylle Schupp. How to use a library? In Canal et al. [4], pages 47–53.
10. Björn Törnqvist and Rune Gustavsson. On adaptative aspect-oriented coordination for critical infrastructures. In Canal et al. [4], pages 63–69.
11. Nesrine Yahiaoui, Bruno Traverson, and Nicole Levy. Classification and comparison of adaptable platforms. In Canal et al. [4], pages 55–61.
12. D. Yellin and R. Strom. Protocol specifications and component adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333, 1997.