Run Time and Design Time Issues on Implementing Software Adaptors

Report on the Second International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'05)

Steffen Becker[1], Carlos Canal[2], Juan Manuel Murillo[3]*, Pascal Poizat[4], and Massimo Tivoli[5]

[1] Universität Oldenburg, Software Engineering Group
`steffen.becker@informatik.uni-oldenburg.de`
[2] Universidad de Málaga, GISUM Software Engineering Group
`canal@lcc.uma.es`
[3] Universidad de Extremadura, Quercus Software Engineering Group
`juanmamu@unex.es`
[4] Université d'Évry Val d'Essonne, LaMI
`poizat@lami.univ-evry.fr`
[5] Università degli Studi dell'Aquila, Software Engineering and Architecture Group
`tivoli@di.univaq.it`

Abstract. Coordination and Adaptation are two key issues when developing complex distributed systems. Coordination focuses on the interaction among computational entities. Adaptation focuses on the problems raised when the interacting entities do not match properly. This is the report of the second edition of the WCAT workshop, that took place in Glasgow jointly with ECOOP 2005. In this second edition, the topics of interest of the participants covered a large number of fields where coordination and adaptation have an impact: models, requirements identification, interface specification, extra-functional properties, automatic generation, frameworks, middleware, and tools.

1 Introduction

The new challenges raised by complex distributed systems have promoted the development of specific fields of Software Engineering such as Coordination. This discipline addresses the interaction issues among software entities (either considered as subsystems, modules, objects, components, or more recently web-services) that collaborate to provide some functionality.

A serious limitation of currently available interface descriptions is that they do not provide suitable means to specify and reason on the interacting behaviour of complex systems. Indeed, while the notations used provide convenient ways

* This work has been partially supported by CICYT under contract TIC 2002-04309-C02-01.

to describe the typed signatures of software entities, they offer a quite limited support to describe their protocol or concurrent behaviour. As a consequence, when an entity or component is going to be reused, one can only be sure that it provides the required signature based 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 [1]. Software Adaptation promotes the use of adaptors - specific computational entities whose main goal is to guarantee that software components will interact in the right way not only at the signature level, but also at the protocol, Quality of Service, and semantic levels. In particular, Adaptation focuses on the dynamic/automatic generation of adaptors. In this sense, models for software adaptation can be considered as a new generation of coordination models.

This report summarizes the second edition of the WCAT workshop, that took place in Glasgow jointly with ECOOP 2005. WCAT'05 tried to provide a venue where researchers and practitioners on these topics could meet, exchange ideas and problems, identify some of the key issues related to coordination and adaptation, and explore together and disseminate possible solutions. The topics of interest of the workshop were:

- Coordination Models separating the interaction concern.
- Identification and specification of interaction requirements and problems.
- Automatic generation of adaptors.
- Dynamic versus static adaptation.
- Documenting components to enable software composition and adaptation.
- Behavioural interfaces, types, and contracts for components, coordinators and adaptors.
- Formal and rigorous approaches to software adaptation.
- The role of adaptation in the software life-cycle.
- Patterns and frameworks for component look-up and adaptation.
- Metrics and prediction models for software adaptation.
- Prediction of the impact of software adaptation on Quality of Service.
- Extra-functional properties and their relation to coordination and adaptation.
- Aspect-oriented approaches to software adaptation and coordination.
- Coordination and adaptation middleware.
- Tools and environments.
- Coordination and adaptation in concurrent and distributed object-oriented systems.
- Interface and choreography description of Web-Services.
- Using adaptors for legacy system integration.
- Industrial and experience reports.
- Surveys and case studies.

The rest of this report 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

To enable lively and productive discussions, prospective participants were required to submit previously a short position paper, describing their work in the field, open issues, and their expectations on the workshop. We received twelve submissions, covering a wide range of aspects of coordination and adaptation. Both the Call for Papers, and the full position papers finally invited can be found at the Website of the workshop:

`http://wcat05.unex.es`

Nineteen participants coming from eight different countries attended the workshop. The list of participants, together with their affiliation and the title of their position papers is as follows:

– Steffen Becker, Workshop Organizer, becker@informatik.uni-oldenburg.de
  Universität Oldenburg (Germany)
  Using Generated Design Patterns to Support QoS Prediction of Software Component Adaptation [2]
– Javier Cámara, jcamara@citic.es
  Javier Cubo, jcubo@citic.es
  Carlos Canal, Workshop Organizer, canal@lcc.uma.es
  Universidad de Málaga (Spain)
  Issues in the Formalization of Web Service Orchestrations [3]
– Denis Conan, Denis.Conan@int-evry.fr
  CNRS UMR SAMOVAR (France)
  An Experience in Adaptation in the Context of Mobile Computing [4]
– Carlos E. Cuesta, cecuesta@infor.uva.es
  Universidad de Valladolid (Spain)
  M. Pilar Romay, pilar.romay@uem.es
  Universidad Europea de Madrid (Spain)
  Meta-Level Architectural Connectors for Coordination [5]
– Alan Davy, adavy@tssg.org
  Waterford Institute of Technology (Ireland)
  Coordinating Adaptation of Composite Services [6]
– Nikolay Diakov, nikolay.diakov@cwi.nl
  CWI - Centrum voor Wiskunde en Informatica (The Netherlands)
  Adaptation of Software Entities for Synchronous Exogenous Coordination [7]
– Ulf Hansson, uhansson@sti-hawaii.com
  STI Medical Systems (United States)

- Miguel A. Pérez Toledano, toledano@unex.es
  Juan Manuel Murillo, Workshop Organizer, juanmamu@unex.es
  Universidad de Extremadura (Spain)
  Using Interaction Patterns for Making Adaptors among Software Components [8]
- Nicolas Pessemier, pessemie@lifl.fr
  INRIA Futurs, USTL-LIFL (France)
  A Three Level Framework for Adapting Component-Based Systems [9]
- Pascal Poizat, Workshop Organizer, poizat@lami.univ-evry.fr
  Université d'Évry Val d'Essonne, LaMI (France)
- José Raúl Romero, jrromero@lcc.uma.es
  Universidad de Málaga (Spain)
  Software Adaptation in the Context of MDA [10]
- Pablo Sánchez, pablo@lcc.uma.es
  Universidad de Málaga (Spain)
  Aspect-Oriented Approaches for Component Adaptation [11]
- Massimo Tivoli, Workshop Organizer, tivoli@di.univaq.it
  Università degli Studi dell'Aquila (Italy)
- George Wells, G.Wells@ru.ac.za
  Rhodes University (South Africa)
  Coordination Languages: Back to the Future with Linda [12]
- Nesrine Yahiaoui, nesrine.yahiaoui@prism.uvsq.fr
  EDF R&D and UVSQ PriSM (France)
  Adaptation management in Multi-View Systems [13]

Invited participants made a five-minutes presentation of their positions, followed by a discussion that served to identify a list of open issues in the field. We divided these issues into three categories or groups, Adaptation at Design Time, Adaptation at Run Time and Implementation Techniques. Participants were then assigned to one of them, attending to their interests. The task of each group was be to discuss about a subset of the previously identified issues.

The group dealing with design time adaptation focused on detecting and resolving mismatches in component interactions which can be identified at design time, i.e., by statically analysing the available specification of the cooperating entities. Additionally, the members of the group discussed scenarios how to include adaptation methods into the software development process at system design. The group on runtime adaptation addressed issues related to the detection and correction of adaptation problems when the system is run, which yields additional constraints and difficulties. The group on implementation techniques studied how Coordination Languages and Aspect Oriented Techniques could support the adaptation process.

Finally, a plenary session was held, in which each group presented their conclusions to the rest of the participants, followed by a general discussion.

3 Summary of the Discussions

3.1 Classification of Adaptation Issues

Prior to the workshop, the organizers developed a scheme for classifying the more relevant issues raised in the position papers submitted. We came up with a two-dimensional table, classifying these issues with respect to the following criteria:

– The phase in the development process in which adaptation would take place was classified into *Design Time* and *Run Time*. The former addresses adaptation during system development, while the latter performs adaptation when the program is being executed in its target environment.
– The tasks to be carried for achieving adaptation were classified into the *Detection* of an adaptation problem; the *Models* used to measure the mismatch; the *Methods* or techniques to overcome this mismatch, and the actual *Implementation* of adaptors.

After each presentation we asked the speakers to classify their proposals, and the open issues they were interested in discussing according to this classification. With the results, we decided to build three groups. The first group discussed design time adaptation from the detection, modelling and method-development point of view. The second group did the same for runtime adaptation issues. Finally, the third group looked at implementation techniques suitable for deploying adaptors or coordinators.

In the following sections, we summarize the details of the discussions in the respective groups.

3.2 Adaptation at Design Time

The discussion group was constituted by Steffen Becker, Carlos Canal, Nikolay Diakov, Nesrine Yahiaoui, and José Raúl Romero. The aim of the group was to discuss open issues in the area of adaptation which rise during the design of a software system. The main focus of the group was on the detection of design time adaptation problems, and on processes guiding the software developer when discovering an adaptation problem. Another topic was the design of a common benchmark for approaches dealing with design time adaptation but it could not be discussed in depth due to a lack of time. This benchmark was supposed to compare different approaches to adaptation with respect to how good methods and respective tools can perform when dealing with certain problems. In the following we give an outline of the results on the discussion on the remaining issues.

Detecting an adaptation problem at design time can be seen as a special operation performed on at least two interfaces of two software entities. On the one side there is an entity requesting certain features. On the other side there is an entity offering certain other features. During design time of a system there is the challenge to decide whether the provided features are a superset of the features required. If they are not a superset we call this an interoperability problem. The detection of such problems is based on the available specification data of the interfaces. In the group we assumed that the entities itself are not existing in the early phases of the system development. Based on that assumption, the group identified three main classes of detectable interoperability problems:

1. *Signature mismatches*: Signature mismatches happen if protocols collide, that is, if the required service call has a different signature than the services available, e.g. a different service name or parameter count or incompatible parameter, return, or exception types.
2. *Behavioural mismatches*: Behavioural mismatches were classified into two subtypes, protocol and constraint mismatches, during the discussion:
   – Protocol mismatches happen if the required service call sequences are not part of the allowed service call sequences on the offering software entity. For example, an initialisation call, e.g. `open`, is necessary to access a file object, but the requiring entity is not issuing that call but calls `read` immediately.
   – Constraint mismatches happen if the pre-condition can not be established by the caller or if the post-condition is insufficient for the caller to proceed.

   Note, as you can express protocols by using constraints, the classification is ambiguous. Nevertheless, it is useful when a trade-off has to be made between the expressiveness and analysability of the different classes.
3. *QoS mismatches*: QoS mismatches occur if an entity requesting a service expects a certain quality of the delivered service and the expectation is not fulfilled. This is often important as the calling services' own QoS depends on the correct fulfilment of the QoS guaranteed by the called entity.

The discussion on the signature mismatches focused mainly on the detection of mismatches and their reasons. The group came to the result that many problems of this kind originate in semantic differences between the provided and the required signatures. For example, take the name of the signature: It might be given by simply using different languages or by using synonym terms. When dealing with the parameter types, different types of the parameters often come from a (slightly) different understanding of the underlying concept. Taking a `Price` object, one does not know if the stored price is with or without VAT. Hence, a comparison with an object of type `PriceWithVAT` can not be performed without further information.

Accepting the fact that problems belonging to the class of semantic mismatches are hard to detect, the group nevertheless discussed existing approaches to deal with them. Three approaches were highlighted:

– Ontology-based approaches are used, which relate concepts and hence enable the detection of mismatches. Nevertheless, they only work if the regarded software entities are specified using the same ontology. Using enhanced ontologies, it may be possible to detect synonyms and thus adapt the software entities accordingly.

– Test based approaches disregard the possible mismatches and perform tests on the components and check whether post-conditions are fulfilled or not. An encountered exception is also counted as violation of some conditions. As long as the tests are successful, we assume that there is no mismatch. Whenever a test fails, we assume a mismatch, try to resolve it, and perform the test again. This is repeated as long as tests fail.
– The use of domain standards is a feasible approach to prevent mismatches from the beginning. First, a domain standard containing the domain concepts and tasks is transformed into a set of interfaces. As long as all entities are designed according to this standard they are expected to work together.

Coming to the protocol related mismatches, the group realised that there is a trade-off between expressiveness and analysability. The term analysability characterizes the efficiency of the detection of mismatches. The detection of a protocol mismatch can be reduced to computing a subset relationship: Every sequence of calls a component may call on its required interfaces has to be part of the set of offered call sequences. In any case where this relationship is not computable in general or its computation is complex with respect to time or memory consumption it gets hard to detect interoperability problems in arbitrary cases. For example, the subset relationship for two given Finite State Machines (FSMs) is always decidable with quite efficient algorithms. However, for specifications based on some kind of logic this is not true in general. Nevertheless, it is possible to specify more cases when utilizing the logic instead of the FSM. Thus, there is a trade-off between the amount of information you can specify and the analysability of the specification with respect to interoperability problems.

Finally, the group discussed whether and how the software development process can be improved by methods dealing with the detection and the resolution of interoperability problems. One paradigm which has been considered by some group members as interesting is model driven architecture (MDA). MDA based methods utilize models to generate code from specifications which fits to these specifications. In the context of adaptation MDA can be used to generate adaptors from specifications of interoperability problems. The group agreed on the thesis that MDA based adaptation methods can help in this process as it has been applied a few times already and thus eases the introduction in industrial training courses.

Hence, MDA seems to be a method which can improve the development process, but the group agreed on certain prerequisites. First, there is a need of knowledge on the (domain) concepts which are used as modelling constructs. This knowledge can only be transformed into modelling constructs for well understood domains, since only in these domains concepts do not change frequently. Thus, they can be used to build fitting abstractions which also do not change often. In domains which are not understood deeply, we encounter the problem that we have to put a lot of information into the model without gaining advantages from pre-established terms. If the information required for making the automatic translation between models is too much, it will be probably better to use a different adaptation approach, instead of trying to do model transformations.

Due to a lack of time, the group only discussed briefly about other approaches. Generative and pattern based proposals to adaptation were mentioned by Becker [2]. Methods based on formal definitions of interaction and coordination have been presented by Diakov [7]. The use of explicit coordination of Web Services using process algebra based specifications was investigated by Canal [3]. Nevertheless, less is known on the impact of these methods on real-world software development scenarios. This has to be considered as future work.

Thus, a final topic we discussed was how to establish methods for adaptation in software development processes. One thing is to prove that the application of such methods can save resources during the development process. Thus, there is a need for experimental research in software engineering which examines the methods in case studies using realistic examples. Another one is to educate the developers on how to detect mismatches and to train them in the application of existing methods. Finally, it is of significant importance to automate the application of any method as much as possible, to ease the initial introduction of a method into a development process. Highly automated methods can be learned easier. Hence, they are available to a larger group of developers in shorter time spans.

3.3 Adaptation at Run Time

The aim of the group was to discuss adaptation issues in the context of runtime adaptation. As usual with adaptation, these issues are twofold: first being able to perform problem detection, and then (trying to) resolve it. The discussion group was integrated by Javier Cámara, Javier Cubo, Carlos Cuesta, Alan Davy, Nicolas Pessemier, Pascal Poizat, Pilar Romay and Massimo Tivoli.

Runtime and static adaptation. Opposite to static (*i.e.*, at compile time) adaptation, dynamic adaptation is performed at runtime. In static adaptation, detection and correction can be performed before the system is run, hence yielding systems correct by construction. This can be achieved because systems are closed. Dynamic adaptation has to cope with open systems in which the components and their protocols may change over time. Moreover, dynamic adaptation has to deal with efficiency issues: speed of the process, impossibility to stop the system while performing adaptation, etc. Hence a dynamic adaptation process has to be incremental. A first adaptor, A_0, is built (if needed) at compile time, and then adaptation has to evolve when things change in the system. This raises the question of detection of changes (and problems).

Detection phase. The first question the group addressed was *when does the detection process have to take place?* It has to be performed at deployment time and then also each time a modification takes place:

– adding or removal of elements of the system architecture used in the detection process: components, interfaces or bindings;

– modification in the nature of one of these elements. This mainly concerns the modification of a component interface (ports, port properties, protocol) or the refinement of components which cause an interface modification, but a change in a component implementation may also be a cause to run the detection process.

Detection can be performed in relation with (formal) models of the entities of concern (interfaces, components), see [1]. Whenever there do not exist suitable models or when there are models for the components but when no global model of the whole system can be computed (for efficiency matters), detection should rely on a *monitoring process*. It could be also interesting to be able to perform several detections at the same time, *e.g.*, for different aspects to be adapted, yielding aspect specific monitors.

Then the group addressed the issue of *what can be monitored or detected?* Mismatches are the first elements one wants to avoid using adaptation. However, different instantiations of the *mismatching* concept or of its consequences are possible:

– deadlocks which can be either local ones (at the level of interactions between two components interfaces *i.e.*, a binding), or global ones (at the whole system level). Deadlocked components cannot progress, maybe yielding a service denial, but this does not always mean that the global system is deadlocked. A (globally) deadlocked system cannot progress at all. Monitoring deadlocks requires timers and timeout detection to make the difference between deadlocked and slow systems;
– in order to avoid checking for deadlock at the system level, detection mechanisms should be *compositional*, enabling only local checking, *e.g.*, checking only bindings associated to the only one component which has been added, removed or replaced;
– in this context, the detection process should also check for incorrect replacements, *i.e.*, when a component is replaced by one or several components which does not yield a compatible interface (the notion of interface being taken in the large, that is taking into account either static, behavioural or non functional properties [1]).

There are other bad situations one wants to avoid using adaptation. These are related with (local or global) properties of the system:

– safety properties. Here one can use invariants/assertions or temporal logics to express these properties. The detection process can rely on monitors, exception treatment and runtime model checking. In the case of temporal logics, only past operators should be used as they can be checked against histories stored in the components;
– liveness and fairness properties, for which, like with deadlock detection, monitoring needs timer/timeout techniques;

– at the component level, a component can also see as a bad situation a change in its environment (no response from a server, QoS decreasing, ...). A certain level of context-awareness in components could be interesting.

Correction phase. Whenever a problem has been detected before it happens, two solutions are possible. The first one, from static adaptation, is to use a user-defined mapping to build an adaptor which concretely may be a new component, a new connector or even a change in an adaptive middleware behaviour. However this is not possible in dynamic adaptation, but for adaptor A_0. In replacement, heuristics or ontology based techniques should be used to obtain the mapping. The second solution consists in inhibiting specific behaviours which may lead to the detected problem, *e.g.*, sequences of communications leading to detected deadlocks or states in which properties do not yield. This solution also comes from static adaptation and can be applied in dynamic adaptation synthesizing a controller which makes components behave in a way that prevents them to reach problem states.

If a problem is only detected when it happens, then rescue processes may be possible. For local deadlocks, it could be possible to partially duplicate a deadlocked component and then perform a replacement. Other problems, such as global deadlock, have no solutions but to restart the whole system.

Conclusions. Not every problem, even if detected, can be solved using adaptation. This is even more difficult in a dynamic/runtime context. The group concluded that techniques developed in the context of distributed operating systems (monitoring, transaction mechanisms, ...) could help lifting static adaptation solutions up to the context of runtime adaptation.

3.4 Implementation Techniques

The discussion group was integrated by Denis Conan, Miguel Pérez Toledano, Juan Manuel Murillo, Pablo Sánchez, and George Wells. The discussion was focussed on how software adaptation can be managed through different existent implementation techniques. In particular, Coordination Languages and Models [14], and Aspect-Oriented Languages were considered as implementation techniques suitable for adaptation. The former because coordination languages are mainly concerned with the specification of the synchronized processes interaction and software adaptation usually requires adapting the way in which software entities interact. The latter because they provide a means to separately understand, focus on, reason about, specify, maintain, and adapt crosscutting concerns and software properties in general. Next, the conclusions reached through the discussions are summarized.

1. *Coordination Models*: Coordination Models and Languages provide mechanisms to specify the coordination constraints of a software system. Such constraints involve all the dependencies between the computational entities of the system, that is, their interactions. So, Coordination Languages could

be good tools to support adaptation when the subject of the adaptation is the way in which software entities interact.

Coordination Models can be classified in two different categories [14] named *Endogenous* and *Exogenous*. Such categories are motivated by the different mechanisms used by coordination models to deal with the coordination constraints of the system. So, during the discussions both categories were treated separately.

Endogenous coordination languages provide primitives that must be incorporated within the computational entities for its coordination. The most representative language in this category is Linda [15]. Linda supports coordination/communication through a shared *tuple space* in which processes put and get tuples. The semantics of the primitives to put and get tuples and the pattern matching mechanism to find tuples in the tuple space define how processes coordinate. Thus, the way in which processes interact is located in such primitives. If the way in which processes interact (interaction protocol) is wanted to be adapted, it could be easily done adapting the behaviour of the primitives to access the tuple space and the pattern matching mechanism. This is not a new idea. As indicated by Wells [12] a lot of works can be found in the literature proposing new primitives to access the tuple space and new behaviours for the pattern matching mechanism. Of course, since the behaviour of the mentioned primitives cannot be modified at runtime this is a design time adaptation. Anyway, the group also concluded that it is not easy to know in advance the kind of adaptation that could be managed modifying the behaviour of the mentioned components. It must be taken into account that all the entities communicating in a Linda application are sharing the same primitives to interact and modifying the behaviour of the primitives to adapt a particular pair of computational entities means that the interaction protocols for the rest of entities will be also modified.

In contrast, exogenous coordination languages place the coordination primitives in entities different from those to be coordinated. These entities can be seen as coordinators. Manifold [16] is the most known exponent in this category. In Manifold there are two kinds of processes: *atomic* processes and *manifolds*. Atomic processes are the software entities to be coordinated. Manifolds react to the event raised by atomic processes enforcing coordination policies over them. Thus, the coordination/interaction protocols are encapsulated in manifolds. Such clear separation between processes functionality and interaction protocols[6] provides a better support for adaptation. Now, adapting the interaction protocols can be managed changing the coordinators processes. Design time adaptation can be easily managed recoding the interaction protocols implemented by coordination processes. Moreover, runtime adaptation is also supported. In software applications supported

[6] Note that the interaction protocols are here considered as a non functional property of processes. In fact we considered that the functionality of the processes is those for which the process was conceived (abstracting the way in which the process has to interact to do it).

by exogenous coordination models coordinators are regular processes in an open system. So, they are able to appear and disappear during the system operation. In this way, an interaction protocol can be adapted eliminating the coordination process enforcing the protocol and introducing a new one enforcing the adapted protocol.

   The better support for adaptation provided by exogenous coordination models is mainly due to the fact that they provide a clearer separation between functionality and interaction. However, as separation is mainly concerned with coordination it is not clear how other properties could be adapted different from interaction.

2. *Aspect Orientation*: Aspect Oriented Software Development (AOSD) provides techniques to deal the separation of crosscutting concerns along the software life cycle. Aspect Oriented Programming (AOP) is focused on the implementation step proposing programming languages supporting the separated codification of system features that crosscut several software entities (classes, subprograms, processes,..). The code specifying the separated properties is located in a new kind of module commonly called *aspect*. In addition, it must be also specified how aspects affect software entities. In order to obtain the original system behaviour, that information is used to weave the code inside aspects with the software entities. Such weaving can be done both statically, before the application starts or dynamically when the application is already running.

   As introduced by Murillo [17] and Sánchez [11] AOP provides a good support for adaptation. In this case adaptation is managed both adapting system properties separated in different aspects and modifying the specification determining how aspects affects the software entities of the system. The adaptation can be done not only at design time but also at runtime. Since there exist aspect oriented platforms that support the dynamic (runtime) load of aspects, this feature can be used to manage the system adaptation either adding new properties or changing the existing ones.

   With regard to exogenous coordination models, the advantage added by aspect oriented systems is that they provided the possibility of adapting properties different from interaction protocols. So, the main conclusion of the group discussion was that using implementation techniques supporting separation of different concerns facilitates the system adaptation.

The group had initially the aim of discussing other topics but the lack of time prevented it. Some of them were the following:

– As pointed out by D. Conan [4] how reflection techniques can help on adaptation.
– Also, as pointed out by M. Pérez [8] how documenting components can help to adaptation during system implementation.
– Are there other implementation techniques such as pattern or frameworks that could be suitable to deal with adaptation?

– Regarding to aspect orientation, how to deal with the problem of aspect coordination? That is, what happens when new properties added to a system affect other existent properties?

4 Conclusion of the Workshop

After presenting the conclusions of the three discussion groups, the closing session was held. During this session attendants were asked for their general impression about the second edition of WCAT. While the first edition of WCAT tried to define adaptation and its limits with reference to other domains such as evolution or maintenance [1], during this second edition, people agreed that they had been able to discuss on more precise and technical issues. These issues were namely the differences between techniques for adaptation at runtime and design time, and mechanisms to implement or generate adaptors. Implementation, the third part of the detect-correct-implement adaptation triplet, is often eluded in adaptation proposals.

Some future work for next editions of WCAT can be seen in the following issues for which several complementary domains should be investigated:

– the relation with ontologies for the automatic specification of adaptation mappings;
– the relation with distributed and real-time domains for the detection of adaptation needs, and the online application of adaptation (either using design time or runtime techniques);
– the relation with AOSD and MDA techniques, languages and component models for the implementation of adaptors: should we use weaving (AOSD) or refinement (MDA)?

Finally, the community interested in adaptation and coordination identified the task of developing a common suite of example problems found in adaptation and coordination. It is envisioned to use this suite in the future to estimate strengths and weaknesses of proposed adaptation methods, to compare methods with each other, and finally, to validate the usefulness of the proposed approaches.

References

1. Canal, C., Murillo, J.M., Poizat, P.: Report on the First International Workshop on Coordination and Adaptation Techniques for Software Entities. In: Object-Oriented Technology. ECOOP 2004 Workshop Reader. Volume 3344 of Lecture Notes in Computer Science., Springer (2004) 133–147 Workshop proceedings available at http://wcat04.unex.es/.
2. Becker, S.: Using Generated Design Patterns to Support QoS Prediction of Software Component Adaptation. [19] 9–15
3. Cámara, J., Canal, C., Cubo, J.: Issues in the formalization of Web Services Orchestrations. [19] 17–24

4. Kouici, N., Conan, D., Bernard, G.: An experience in adaptation in the context of mobile computing. [19] 47–54
5. Cuesta, C.E., Romay, M.P., Fuente, P., Barrio-Solórzano, M.: Meta-Level Architectural Connectors for Coordination. [19] 25–30
6. Davy, A., Jennings, B.: Coordinating Adaptation of Composite Services. [19] 31–37
7. Diakov, N., Arbab, F.: Adaptation of Software Entities for Synchronous Exogeneous Coordination. An Initial Approach. [19] 39–46
8. Pérez Toledano, M.A., Navasa Martínez, A., Murillo, J.M.: Using Interaction Patterns for Making Adaptors among Software Components. [19] 63–70
9. Pessemier, N., Barais, O., Seinturier, L., Coupaye, T., Duchien, L.: A Three Level Framework for Adapting Component-Based Application. [19] 71–77
10. Moreno, N., Romero, J.R., Vallecillo, A.: Software Adaptation in the Context of MDA. [19] 55–61
11. Fuentes, L., Sánchez, P.: AO approaches for Component Adaptation. [19] 79–86
12. Wells, G.: Coordination Languages: Back to the Future with Linda. [19] 87–98
13. Yahiaoui, N., Traverson, B., Levy, N.: Adaptation management in multi-view systems. [19] 99–105
14. Arbab, F.: What Do You Mean Coordination? Bulletin of the Dutch Association for Theoretical Computer Science (NVTI) (1998)
15. N. Carreiro, D.G.: Linda in Context. Communications of the ACM **32** (1989) 133–147
16. Arbab, F.: The IWIM model for coordination of concurrent activities. [20]
17. Eterovic, Y., Murillo, J.M., Palma, K.: Managing component adaptation using aspect oriented techniques. [18] 101–108
18. Canal, C., Murillo, J.M., Poizat, P., eds.: First International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'04). (2004)
19. Becker, S., Canal, C., Murillo, J.M., Poizat, P., Tivoli, M., eds.: Second International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'05). (2005)
20. Ciancarini, P., Hankin, C., eds.: First International Conference on Coordination Languages and Models, (COORDINATION '96). Volume 1061 of Lecture Notes in Computer Science., Springer (1996)