Measuring component adaptation

Antonio Brogi[1], Carlos Canal[2], and Ernesto Pimentel[2]

[1] Department of Computer Science, University of Pisa, Italy
[2] Department of Computer Science, University of Málaga, Spain

Abstract. Adapting heterogeneous software components that present mismatching interaction behaviour is one of the crucial problems in Component-Based Software Engineering. The process of component adaptation can be synthesized by a transformation from an initial specification (of the requested adaptation) to a revised specification (expressing the actual adaptation that will be featured by the deployed adaptor component). An important capability in this context is hence to be able to evaluate to which extent an adaptor proposal satisfies the initially requested adaptation. The aim of this paper is precisely to develop different metrics that can be employed to this end. Such metrics can be fruitfully employed both to assess the acceptability of an adaptation proposal, and to compare different adaptation solutions proposed by different servers.

1 Introduction

One of the general objectives of coordination models is to support the successful interoperation of heterogeneous software elements that may present mismatching interaction behaviour [12]. In the area of Component-Based Software Engineering, the problem of component adaptation is widely recognised to be one of the crucial issues for the development of a true component marketplace and for component deployment in general [5, 6]. Available component-oriented platforms address software interoperability at the signature level by means of Interface Description Languages (IDLs), a sort of *lingua franca* for specifying the functionality offered by heterogeneous components possibly developed in different languages. While IDL interfaces allow to overcome signature mismatches, there is no guarantee that the components will suitably interoperate, as mismatches may also occur at the protocol level, due to differences in the interaction behaviour of the components involved [18].

The need for component adaptation is also motivated by the ever-increasing attention devoted to developing extensively interacting distributed systems, consisting of large numbers of heterogeneous components. Most importantly, the ability of dynamically constructing software adaptors will be a must for the next generation of nomadic applications consisting of wireless mobile computing devices that will need to require services from different hosts at different times.

In our previous work [2–4], we have developed a formal methodology for component adaptation that supports the successful interoperation of heterogeneous components presenting mismatching interaction behaviour. The main ingredients of the methodology can be summarised as follows:

- *Component interfaces.* IDL interfaces are extended with a formal description of the behaviour of components, which explicitly declares the interaction protocol followed by a component.
- *Adaptor specification.* Adaptor specifications are simply expressed by a set of correspondences between actions of two components. The distinguishing aspect of the notation used is that it yields a high-level, partial specification of the adaptor.
- *Adaptor derivation.* A concrete adaptor is fully automatically generated, given its partial specification and the interfaces of two components, by exhaustively trying to build a component which satisfies the given specification.

The methodology has proven to succeed in a number of diverse situations [2], where a suitable adaptor is generated to support the successful interoperation of heterogeneous components presenting mismatching interaction behaviours. The separation of adaptor specification and adaptor derivation permits the automation of the error-prone, time-consuming task of constructing a detailed implementation of a correct adaptor, thus notably simplifying the task of the (human) software developer. One of the distinguishing features of the methodology is the simplicity of the notation employed to express adaptor specifications. Indeed the desired adaptation is simply expressed by defining a set of (possibly non-deterministic) correspondences between the actions of the two components.

Consider for instance a typical scenario where a client component wishes to use some of the services offered by a server. (For instance, a client wishing to access a remote system via the network, or a mobile client getting into the vicinity of a stationary server.) The client will ask for the server interface, and then submit its service request in the form of an adaptor specification (together with its own interface). The server will run the adaptor derivation procedure to determine whether a suitable adaptor can be generated to satisfy the client request. If the client request can be satisfied, the server will notify the client by presenting a (possibly modified) adaptor specification which states the type of adaptation that will be effectively featured. The client will then decide whether to accept the adaptation proposal or not. (In the latter case the client may decide to continue the trading process by submitting a revised adaptor specification, or to address another server.)

Expressing adaptation trading by means of adaptor specification features two main advantages:

- *Efficiency.* Clients and servers exchange light-weighted adaptor specifications rather than component code. Besides contributing to the efficiency of communications, this notably simplifies the trading process, when the client has to analyse the adaptation proposed by the server.
- *Non-disclosure.* The server does not have to present the actual adaptor component in its full details, thus communicating only the "what" of the offered adaptation rather than the "how".

Summing up, the process of component adaptation can be synthesized by an adaptor specification S, representing the client request, and by a (possibly modified) adaptor specification C, representing the actual adaptation offered by the server. The specification C is then interpreted as a contract guaranteeing that:

1. The client will be able to interoperate successfully with the adaptor (viz., the client will not get stuck), and
2. all the client actions occurring in C will be effectively executable by the client.

An important issue in this context is hence how to evaluate to which extent a contract proposal satisfies the initial specification defining the requested adaptation. The aim of this paper is precisely to develop different metrics for component adaptation. Such metrics can be fruitfully employed both to assess the acceptability of an adaptation proposal and to compare different adaptations proposed by different servers.

The rest of the paper is organised as follows. Section 2 recalls the key aspects of the methodology for component adaptation developed in [2–4]. Section 3 is devoted to present several metrics to assess the acceptability of an adaptation proposal. Such metrics are then extended in Section 4 to deal with additional quantitative information. Finally, some concluding remarks are drawn in Section 5.

2 Component adaptation

Throughout this paper we will use an example concerning a Video-on-Demand system (VoD), simplified to some extent. The VoD is a Web service providing access to remote clients to a repository of movies in video format.

When interacting with a client, the VoD executes client's instructions, represented by the different input messages the VoD is able to react to (message names are in italics in the description that follows). For instance, clients may *search* the VoD catalog for a movie, or ask the service which is the *première* of the day. Once a certain movie is selected, the client may decide to *preview* it for a few minutes, or to *view* it all. In either case, the client must indicate whether it wants to *play* the movie (which will be shown by a viewer), or to *record* it on its disk. During video transmission, chunks of data are sent by the server by means of repeated *stream* output messages, while the end of the transmission is marked with an *eof* output message.

Following [2], component interfaces are extended with a description of the interactive behaviour of components using a process algebra (here, we use the π-calculus [16]). Suppose that the behaviour of the VoD server during the video data transmission phase is described by a term of the form:

$$Server \ = \ stream!(y).\, Server \ + \ \tau.\, eof!().\, 0$$

where τ is an internal silent action.

Consider now a simple *Client* that (repeatedly) receives data by performing an input action *data*, and which may decide autonomously to abort the data transmission by performing an output action *abort*:

$$Client \;=\; data?(x).\, Client \;+\; \tau.\, abort!().\, 0$$

It is worth observing that the mismatch between the two components above is not limited to signature differences (viz., the different names of actions employed, such as *stream* in the server and *data* in the client), but it also involves behavioral differences, in particular the way in which either component may close the communication unexpectedly.

2.1 Hard adaptation

The objective of software adaptation is to deploy a software component, called *adaptor*, capable of acting as a component-in-the-middle between two components and of supporting their successful interoperation. Following our approach, a concrete adaptor will be automatically generated starting from the interfaces of the components and from a specification of the adaptor itself. Such a specification simply consists of rules establishing correspondences between actions of the two components. More precisely, an adaptor specification is a set of rules of the form:

$$a_1, \ldots, a_m \lozenge b_1, \ldots, b_n \;;$$

where a_i and b_j are the input or output actions performed by the components to be adapted. By convention, given a specification S, actions on the left-hand side of rules refer to one of the components to be adapted (in the examples here, the client), and we will refer to them by $[S]_{cl}$, while actions on the right-hand side of rules refer to the other component (the server), and will be denoted by $[S]_{sr}$. To simplify notation, and without loss of generality, we assume that the sets of client and server actions are disjoint.

The formal semantics of adaptor specifications is defined in [4], here we shall recall this semantics informally by means of examples. For instance, the adaptation required for the video transmission between the server and the client can be naturally expressed by the specification:

$$S_1 = \left\{ \begin{array}{rcll} data?(u) & \lozenge & stream!(u) & ; \\ abort!() & \lozenge & stream!(u) & ; \\ abort!() & \lozenge & & ; \\ & \lozenge & eof!() & ; \end{array} \right\}$$

The first rule in S_1 states that the input action *data?* in the client matches or corresponds to the output action *stream!* in the server. This same *stream!* action is also matched to client's *abort!* in the second rule, stating that the client may abort the transmission even when the server is trying to send video data. Thus, combining both rules we have that the action *stream!* may correspond non-deterministically to either *data*? or *abort*!, depending on the evolution of the client. In the third and fourth rules, actions *abort!* and *eof!* are represented without a corresponding action in the other side. For instance, the fourth rule of S_1 indicates that the execution of an *eof!* server action does not necessarily call for a corresponding client action. Notice that rules in an adaptor specification may establish one-to-one, one-to-many, and many-to-many correspondences among actions of the two components. They can also be employed to express asymmetries naturally, typically when the execution of an action in one of the component may not require a corresponding action to be executed by the other component.

Given an adaptor specification and the interfaces of two components, a fully automated procedure [2] returns non-deterministically one of the possible adaptors components (if any) that satisfy the specification, and that let the two components interoperate successfully. For instance, for the specification S_1 above the generation procedure may return the adaptor:

$$\begin{array}{rl} A = & \mathit{stream}?(u).\ (\mathit{data}!(u).\ A + \mathit{abort}?().\ \mathit{eof}?().\ 0) \\ & + \\ & \mathit{eof}?().\ \mathit{abort}?().\ 0 \end{array}$$

In this example a full adaptation is achieved, and hence the server will return the submitted specification S_1 as the contract proposal. However, in general, the server may be able to offer the client only a partial adaptation, namely a contract C which is a proper subset of the submitted specification S.

To illustrate the idea, let us develop further our example of the VoD system. Suppose now that the client uses different action names for accessing the service than those considered by the server (again action names are in italics in the description that follows). For instance, the client wishes to perform its *info* action either for requesting information on a particular movie (by indicating its title), or for asking the service to suggest a movie (by using in this case the empty string as parameter), and its *watch* and *store* actions to play and record a movie, respectively. Hence the client submits the adaptor specification S_2 below, establishing correspondences with the previously described server actions[3]:

$$S_2 = \left\{ \begin{array}{c} \mathit{info}!(t) \Diamond \mathit{search}?(t)\ ; \\ \mathit{info}!(''\ '') \Diamond \mathit{premiere}?()\ ; \\ \mathit{watch}!(m) \Diamond \mathit{view}?(m), \mathit{play}?()\ ; \\ \mathit{store}!(m) \Diamond \mathit{view}?(m), \mathit{record}?()\ ; \end{array} \right\}$$

Let us suppose that as a reply to the specification S_2, the client receives from the server the following contract C_A:

$$C_A = \left\{ \mathit{info}!(t) \Diamond \mathit{search}?(t)\ ; \right\}$$

The straightforward reading of the contract proposal C_A is that while the server commits to let the client search the movies database, for some reason it will

[3] For the sake of simplicity, we omit here the correspondences concerning the data transfer actions (*data*, *abort*, *stream*, *eof*) already described in specification S_1.

not feature the adaptation required to let the client watch or store such movies. (For instance, the server might decide to feature a partial adaptation even if a full adaptation would be feasible, in order to balance its current workload or for other internal service policies.)

Hence, given an adaptor specification S, the type of component adaptation that we have described so far:

- either it yields a (possibly partial) adaptation proposal C (where C is a subset of S),
- or it fails when no adaptation is possible.

The sole possibility of removing some rules from the initial specification obviously limits the success possibilities of yielding a (partial) adaptation. Indeed there are many situations in which more flexible ways of weakening the initial specification may lead to deploying a suitable partial adaptor, as we shall discuss in the next section.

2.2 Soft adaptation

The methodology for *hard* adaptation described in [2] has been extended in [3] to feature forms of *soft* adaptation. One of the key notions introduced in [3] is the notion of *subservice*. Intuitively speaking, a subservice is a kind of surrogate of a service, which features only a limited part of such service. Formally, subservices are specified by defining a partial order $\sqsubset$ over the actions of a component:

$$b_j \quad \sqsubset \quad b_i$$

indicating that the service b_j is a subservice of b_i.

It is important to observe that adding subservice declarations to component interfaces paves the way for more flexible forms of adaptation. Indeed, subservice declarations support a flexible configuration of components in view of their (dynamic) behaviour, without having to modify or to make more complex the protocol specification of component interfaces.

As one may expect, the introduction of subservices increases notably the possibilities of successful adaptations, as an initial specification can be suitably weakened by providing (when needed) subservices in place of the required services. A direct consequence of enabling soft adaptation is that a client that submits an adaptor specification may now receive a rather different contract proposal, in which the server may declare its intention both to feature only some of the services requested and to subservice some of them. Hence, the process of soft adaptor generation in presence of subservice declarations can be described as follows.

1. The initial adaptor specification S is actually interpreted as the specification S^* obtained by expanding every rule r in S with a new set $subs(r)$ of correspondence rules that are obtained by replacing one or more (server) actions in r with a corresponding subservice. That is, $S^* = S \cup subs(S)$, where $subs(S) = \cup_{r \in S} subs(r)$.

2. The process of adaptor construction generates (if possible) a partial adaptor that satisfies a subset C of the extended specification S^*.

(Notice that while both C and S are subsets of S^*, in general C is not a subset of S since possibly some service requests have been subserviced.)

For instance, in the Video-on-Demand service, offering a clip preview of a movie can be considered a typical subservice of offering the whole movie, while starting to view a movie could be considered as a subservice of downloading it into the hard disk of the client. These subservice definition are expressed as follows:

$$\begin{array}{lcl} \mathit{preview}?(m) & \sqsubset & \mathit{view}?(m)\ ; \\ \mathit{play}?() & \sqsubset & \mathit{record}?()\ ; \end{array}$$

The previous adaptor specification S_2 is hence actually interpreted by the server as the following expanded specification S_2^*:

$$S_2 = \left\{ \begin{array}{r} \mathit{info} \Diamond \mathit{search}\ ; \\ \mathit{info} \Diamond \mathit{premiere}\ ; \\ \mathit{watch} \Diamond \mathit{view}, \mathit{play}\ ; \\ \mathit{store} \Diamond \mathit{view}, \mathit{record}\ ; \end{array} \right\} \qquad S_2^* = \left\{ \begin{array}{r} \mathit{info} \Diamond \mathit{search}\ ; \\ \mathit{info} \Diamond \mathit{premiere}\ ; \\ \mathit{watch} \Diamond \mathit{view}, \mathit{play}\ ; \\ \mathit{watch} \Diamond \mathit{preview}, \mathit{play}\ ; \\ \mathit{store} \Diamond \mathit{view}, \mathit{record}\ ; \\ \mathit{store} \Diamond \mathit{view}, \mathit{play}\ ; \\ \mathit{store} \Diamond \mathit{preview}, \mathit{record}\ ; \\ \mathit{store} \Diamond \mathit{preview}, \mathit{play}\ ; \end{array} \right\}$$

where for simplicity we have omitted signs and parameters in the actions of both components.

Depending both on the actual protocols of the components, and on the server's policy, the server may return different contract proposals, for instance:

$$C_B = \left\{ \begin{array}{r} \mathit{info} \Diamond \mathit{search}\ ; \\ \mathit{watch} \Diamond \mathit{view}, \mathit{play}\ ; \\ \mathit{watch} \Diamond \mathit{preview}, \mathit{play}\ ; \\ \mathit{store} \Diamond \mathit{preview}, \mathit{record}\ ; \end{array} \right\}$$

where the contract proposal C_B indicates that some *watch* requests will be adapted into previews, while all *store* requests will be adapted into previews.

As in the case of the partial adaptation described in the previous section, it is worth noting that a server may decide to subservice some of the client requests even if this is not strictly necessary in order to achieve a successful interoperation of the two components. Hence, the contract proposal returned may present relevant differences with respect to the original client requests, and some additional information would be of help for deciding whether or not to accept a proposed adaptation. Thus, together with the contract proposal, the server should provide the client with some explanation of why some requests have not been satisfied, such as for instance "temporarily unavailable service", "insufficient access rights", or "unsolved protocol mismatch", as discussed in [3].

However, this kind of "qualitative" information is usually insufficient and most of the times the client needs a "quantitative" estimation on how close to the original request is a certain contract proposal with respect to an alternative one (either provided by the same server or by a different one). The main aim of this work is exploring different possibilities for achieving a quantitative measure of adaptation.

3 Measuring adaptation

In this section we develop two initial metrics to compare an initial specification S with a contract proposal C.

3.1 Rule satisfaction

A first way to measure the distance between an initial specification S and a contract proposal C is to inspect how many rules of S are "satisfied" by C.

For the case of hard adaptation, where C is a subset of S, one may simply count the number of correspondence rules of S that are contained in C, that is consider the ratio:

$$\frac{\sharp\, C}{\sharp\, S}$$

where $\sharp\, X$ denotes the number of rules in a specification X.

However, the above ratio does not make much sense in the context of soft adaptation, where in general the rules of C are not a subset of the rules of S. Indeed, as we have already mentioned, both S and C are subsets of S^*, but C may even contain more rules than S.

In the general context of soft adaptation, one should measure how many rules of S are "satisfied" by C, either because they are included *verbatim* in C or because a subserviced version of them is included in C. More precisely, we can single out three possible cases for each rule r in S:

1. *r is fully satisfied by C.* Namely, r belongs to C, but no subserviced version of r belongs to C. Formally:

$$r \in C \ \wedge \ subs(r) \cap C = \emptyset$$

2. *r is partly satisfied by C with some subservicing.* Namely, some subserviced version of r belongs to C. Formally:

$$subs(r) \cap C \neq \emptyset$$

3. *r is not satisfied by C.* Namely, neither r nor any subserviced version of r belongs to C. Formally:

$$r \notin C \ \wedge \ subs(r) \cap C = \emptyset$$

Based on the above observation, we define the metric $m_1(S, C)$ which, given a specification S and a contract C, returns two values. The first value indicates the percentage of rules of S that are fully satisfied by the contract proposal C. The second value indicates the percentage of rules of S that are partly satisfied, that is subserviced, by the contract proposal C. Formally:

$$m_1(S, C) \;=\; \left\langle \frac{\sharp\,\{r \in S \cap C \mid \mathit{subs}(r) \cap C = \emptyset\}}{\sharp\, S} \;,\; \frac{\sharp\,\{r \in S \mid \mathit{subs}(r) \cap C \neq \emptyset\}}{\sharp\, S} \right\rangle$$

For instance, consider again the specification S_2 and the contract C_B already presented, together with the new contract proposal C_C:

$$C_C = \left\{ \begin{array}{c} \mathit{info} \lozenge \mathit{search} \;; \\ \mathit{watch} \lozenge \mathit{view}, \mathit{play} \;; \\ \mathit{store} \lozenge \mathit{view}, \mathit{play} \;; \end{array} \right\}$$

We have that $m_1(S_2, C_B) = \left\langle \frac{1}{4}, \frac{2}{4} \right\rangle$, while $m_1(S_2, C_C) = \left\langle \frac{2}{4}, \frac{1}{4} \right\rangle$, from which we could deduce that the contract proposal C_B is worse than C_C with respect to the specification S_2, since C_B presents less rules fully satisfied and more subserviced rules than C_C.

3.2 Action enabling

The simple inspection of the percentage of rules satisfied by a contract proposal does not however deal adequately with specifications that contain nondeterministic correspondence rules. For instance, consider once more the specification S_2 and the contract C_B already presented, together with the new contract proposal C_D:

$$C_D = \left\{ \begin{array}{c} \mathit{info} \lozenge \mathit{search} \;; \\ \mathit{info} \lozenge \mathit{premiere} \;; \\ \mathit{watch} \lozenge \mathit{preview}, \mathit{play} \;; \end{array} \right\}$$

We observe that, according to metric m_1, $m_1(S_2, C_B) = \left\langle \frac{1}{4}, \frac{2}{4} \right\rangle$, while on the other hand $m_1(S_2, C_D) = \left\langle \frac{2}{4}, \frac{1}{4} \right\rangle$, from which we should conclude that the contract proposal C_D is closer to the specification S_2 than C_B, since in C_D the number or rules of S_2 fully satisfied are greater than in C_B. However, the adaptor corresponding to the new contract C_D will allow the client to perform a smaller number of actions than C_B (viz., action *store* is not present in C_D). Hence, a finer metric should consider the percentage of client actions (instead of rules) whose execution will be actually enabled by the adaptor proposal. Again, for the case of hard adaptation, one may simply consider the ratio between the number of client actions occurring in C and in S, formally:

$$\frac{\sharp\, [C]_{cl}}{\sharp\, [S]_{cl}}$$

However, the above ratio is a bit coarse in the general context of soft adaptation, since it does not consider the subservicing possibly operated by the adaptor. More precisely, we can single out three possible cases for a client action a occurring in S (viz., for each $a \in [S]_{cl}$):

1. *a is fully enabled by C.* Namely the execution of a is enabled by C without introducing subservicing, that is, a only occurs in rules of C that were in S too. Formally:
$$a \in [C]_{cl} \;\wedge\; a \notin [C \cap subs(S)]_{cl}$$
2. *a is partly enabled by C with some subservicing.* Namely the execution of a is enabled by C via some subservicing, that is, a occurs in some subserviced version of rules of S. Formally:
$$a \in [C \cap subs(S)]_{cl}$$
3. *a is not enabled by C.* Namely a does not occur in (any rule of) C. Formally:
$$a \notin [C]_{cl}$$

Based on the above observation, we define a new metric $m_2(S, C)$ which, given a specification S and a contract C, returns two values. The first value indicates the percentage of client actions in S that are fully enabled by C. The second value indicates the percentage of client actions in S that are partly enabled by C. Formally:

$$m_2(S, C) = \left\langle \frac{\sharp\ ([C]_{cl} \backslash [C \cap subs(S)]_{cl})}{\sharp\ [S]_{cl}}\ ,\ \frac{\sharp\ [C \cap subs(S)]_{cl}}{\sharp\ [S]_{cl}} \right\rangle$$

For instance, considering again the specification S_2 along with the contracts C_B and C_D, we observe that $m_2(S_2, C_B) = \langle \frac{1}{3}, \frac{2}{3} \rangle$ while $m_2(S_2, C_D) = \langle \frac{1}{3}, \frac{1}{3} \rangle$, from which we should now conclude that C_B —in which all client actions are being adapted— represents a better adaptation than C_D —in which no adaptation for action *store* is featured. In other words, if we only take into account the number of (partially) satisfied rules (as given by metric m_1), the contract proposal C_D presents a greater level of satisfaction for the specification S_2. Nevertheless, from an intuitive point of view, C_B seems closer to S_2 than C_D, which is correctly reflected by metric m_2.

4 Adding quantitative information

So far we have assumed that contract specifications are plain sets of correspondence rules, some of which may have been introduced because of subservicing. We now consider the case in which contract specifications are enriched so as to contain additional quantitative information denoting usage percentage of the individual rules during the construction of the adaptor. For instance, consider again the specification S_2 already presented, together with the following annotated version of the contract proposal C_B:

$$C_{B'} = \left\{ \begin{array}{cl} info \lozenge search & : 1; \\ watch \lozenge view, play & : .8; \\ watch \lozenge preview, play & : .2; \\ store \lozenge preview, record & : 1; \end{array} \right\}$$

The intended meaning of the numbers annotated on the right of the correspondence rules is to provide an estimation of the degree of subservicing employed in the adaptation. For instance, the first rule of $C_{B'}$ indicates that the first rule of S_2 is kept as is in C_B, i.e., without subservicing it (while the second rule of S_2 has not been used at all during the construction of the adaptor). On the other hand, the second and third rules of $C_{B'}$ state that the third rule of S_2 has been used 80% of the times during the adaptor construction to match client action *watch*, while its subserviced version has been employed the remaining 20% of the times. Finally, the fourth rule of $C_{B'}$ states that the fourth rule of S_2 has been always subserviced.

Hence, we can refine metric m_1 so as to measure the *percentage* with which the rules of S are satisfied without or with subservicing by C:

$$m_3(S,C) \;=\; \left\langle \frac{\sum_{r\in S}\{p \mid \langle r:p\rangle \in C\}}{\sharp\, S} \;,\; \frac{\sum_{r\in subs(S)}\{p \mid \langle r:p\rangle \in C\}}{\sharp\, S} \right\rangle$$

For instance, for the contract $C_{B'}$ above we have that $m_3(S_2, C_{B'}) = \langle\, \frac{1.8}{3} \;,\; \frac{1.2}{3} \,\rangle$, which offers a more precise information on the adaptation proposal than that provided by $m_1(S_2, C_B) = \langle \frac{1}{4}, \frac{2}{4} \rangle$ for the non-annotated version C_B of the same contract proposal.

The metric $m_3(S,C)$ above allows to exploit the information provided by annotated contracts in order to measure the percentage of *overall* subservicing of the original specifications. However, subservices may not be all the same for clients. Typically, a client may consider a subservice b_j to be an acceptable surrogate of b_i, while a different subservice b_k of b_i may not be of interest or valuable for the client. Therefore, a finer metric should try to combine the information provided by an annotated contract together with an assessment of subservices performed by the client.

We hence consider that a client may wish to set its own criteria to assess the quality of a soft adaptor obtained in response to its request for adaptation. As soft adaptors employ subservicing, clients may wish to assess how much the employed subservicing will affect the quality of the adaptation proposed. For instance, in our VoD system, the client may wish to weight the subservice relation presented by the server on the basis of its own assessment. For instance, the client may annotate the above subservice relation as follows:

$$\begin{array}{lll} \textit{preview} & \sqsubset_{.5} & \textit{view}\ ; \\ \textit{play} & \sqsubset_{1} & \textit{record}\ ; \end{array}$$

where the first annotation ($\sqsubset_{.5}$) indicates that the client will be half satisfied if the *preview* subservice will be provided in place of the *view* service. The second annotation ($\sqsubset_1$) instead indicates that the subservicing of *record* with *play* is not relevant for the client. Annotations must maintain some kind of consistency. In fact, they must satisfy the following property:

$$a \sqsubset_u b \quad\wedge\quad b \sqsubset_v c \quad \textit{implies} \quad a \sqsubset_{u\times v} c$$

which ensures that $\sqsubset_{\cdot}$ is also a partial order.

The annotated subservice relation can be lifted to rules as follows:

$$r' \sqsubset_v r \quad \textit{iff} \quad v = \begin{cases} \min_{a' \in [r']_{sr}, a \in [r]_{sr}} \{x \mid a' \sqsubset_x a\} & \textit{if} \quad r' \in \textit{subs}(r) \\ 1 & \textit{if} \quad r' = r \end{cases}$$

We can hence refine metric m_3 to take into account the subservicing assessment made by the client:

$$m_4(S, C) \;=\; \frac{\sum_{r \in S} \{p \times v \mid \langle r' : p \rangle \in C \;\wedge\; r' \sqsubset_v r\}}{\sharp\, S}$$

For instance, for the above specification S_2 and the annotated contract $C_{B'}$, we have that $m_4(S_2, C_{B'}) = (1 \times 1 + .8 \times 1 + .2 \times .5 + 1 \times .5) \div 3 = .8$, which can be considered as a measure of the optimality of the contract for the client, i.e., of the degree of satisfaction achieved during adaptor construction.

The introduction of quantitative annotations (both in contracts and in subservice relations) paves the way for a more precise assessment of the actions or services whose execution will be enabled by an adaptor proposal. However, such an assessment should consider the actual interaction protocols followed by the components and the adaptor. Consider for instance the following specification S_3, and the contract proposal C_E:

$$S_3 = \big\{\, \textit{watch} \Diamond \textit{view}, \textit{play}\ ; \big\} \qquad C_E = \left\{ \begin{array}{ll} \textit{watch} \Diamond \textit{view}, \textit{play} & : .5; \\ \textit{watch} \Diamond \textit{preview}, \textit{play} & : .5; \end{array} \right\}$$

where as before *preview* $\sqsubset_{.5}$ *view*. The quantitative annotations in C_E denote the usage percentage of the rules *during the construction* of the adaptor, resulting in a satisfaction measurement $m_4(S_3, C_E) = (1 \times .5 + .5 \times .5) \div 2 = .375$.

However, let us suppose now that the contract C_E corresponds for instance to the actual adaptor:

$$A \;=\; \textit{watch}.\ (\ \textit{preview}.\ A \;\;+\;\; \textit{view}.\ A\ )$$

It is easy to observe that the real percentage of subservicing employed by the server to enable the execution of client actions depends on the actual protocols of the components involved (in the example, the frequency with which the server drives the adaptor to perform action *preview* instead of *view*). Furthermore, a simple inspection of the protocols of the two components may not suffice. Suppose for instance that the protocol of the server is:

$$\textit{Server} = \;\textit{preview}.\ \textit{Server} \;+\; \textit{view}.\ \textit{Server}$$

From that, we cannot conclude that the adaptor will perform the actions *preview* and *view* with the same frequency, since the choice between the two branches of the alternative may depend on internal policies of the *Server* not represented at the protocol level. Hence, the amount of times that action *watch* will be subserviced when using the adaptor A above will not depend merely on the protocol describing the adaptor, but mainly on the actual behaviour of the *Server* and the *Client* which determines the frequency with which these components present the actions complementary to those offered by the adaptor.

Hence, a quantitative assessment of the actual, possibly subserviced, execution of client actions should take into account also probabilistic information associated with protocols (e.g., see [9, 14]). For instance a probabilistic definition of the previous process *Server*, as:

$$Server \;=\; preview.\, Server \;\; {}_{.9}{+}_{.1} \;\; view.\, Server$$

indicates that the left branch of the alternative will be executed with probability .9. Hence, most of the times the subservice *preview* will be offered to the client instead of the service *view* originally requested. This information should be taken into account in order to refine metrics like m_3 and m_4 in order to yield a more precise information on the adaptation proposal.

5 Concluding remarks

When heterogeneous software components have to be combined in order to construct complex systems, one of the crucial problems to be solved is the mismatching of interaction behaviours they may present. To the best of our knowledge, in spite of the relevance of this topic, not many efforts have been devoted to develop well-founded component adaptation theories.

A number of practice-oriented studies have analysed different issues encountered in (manually) adapting a third-party component for using it in a (possibly radically) different context (e.g., see [10, 11, 19]). The problem of software adaptation from a more formal point of view was specifically addressed by the work of Yellin and Strom [21], which constitutes the starting point for our work. They used finite state grammars to specify interaction protocols between components, to define a relation of compatibility, and to address the task of (semi)automatic adaptor generation. The main advantage of finite state machines is that their simplicity supports an efficient verification of protocol compatibility. However, such a simplicity is a severe expressiveness bound for modelling complex open distributed systems. On the contrary, process algebras feature more expressive descriptions of protocols and enable more sophisticated analysis of concurrent systems. For this reason, several authors propose their use for the specification of component interfaces and for the analysis of component compatibility [1, 7]. A technique for automatically synthetising connectors is discussed in [13] by focussing on avoiding deadlocks caused by mismatching coordination policies among COM/DCOM components that present compatible interfaces. A different approach is that of [20], where software composition is addressed in the context of category theory. The connection between components is obtained by *superposition*, defining a morphism between actions in both components. Morphisms are similar to our specifications, though the kind of adaptation provided is more restrictive, limiting adaptation to a kind of name translation similar to that provided by IDL signature descriptions. Finally, in [17], type systems and finite state automata are used for describing component interfaces, and a game-theoretic framework is defined for the automatic generation of adaptors avoiding component deadlock. Their proposal is quite appealing, although it shares most of the limitations of those based on finite automata.

Following the above comparison with significant related works in the literature, we argue that our approach [2–4] facilitates the adaptation of components by combining expressiveness and effectiveness in a formally grounded methodology. Our proposal is based on the idea of deriving a concrete adaptor, given its specification and the behaviour description of the two components to be adapted. As full adaptation is not always possible (e.g., because behavioral protocols cannot be matched, or because the requested services are not —fully— available), soft adaptation features a flexible way of adapting components, and calls for mechanisms to assess the appropriateness of proposed adaptors.

Component adaptation is also a relevant issue in the context of mobile computation, where nomadic applications (client components) will need to require services from different hosts (server components). Again, the behaviours mismatching can be solved by adapting both client and server protocols, considering an initial adaptation (given by a set of rules) requested by the client. If the server cannot fully satisfy these initial requests, it could offer an alternative (partial) adaptation (given by a set of —possibly— revised rules) , whose appropriateness should be evaluated by using the previously mentioned mechanisms.

If the references to formal adaptation of software components are scarce in the literature, those trying to address a measurement of the results obtained are probably non-existing. There are quite a few proposals for measuring the quality of component-based systems, usually dealing with measuring the adequacy of the components in a repository w.r.t. the particular needs of a software developer (see for instance [8] for catching a glimpse on the state-of-the-art in this field). Although there are works, as [15], dealing with measuring the semantic distance between functional specifications on a formally based setting, most of the proposals in the component-based field deal with component specification and metrics using long lists of mostly qualitative attributes. Some of these metrics are proposed as an estimation of the effort required for adapting (manually) software components.

At the best of our knowledge, this work is the first proposal addressing the measurement of the (automatic) process of software adaptation and of its result (the adaptor). In this paper, we have defined a set of behaviour-oriented metrics which feature different ways of measuring the distance between an adaptor request and the adaptor which will be effectively deployed. Thus, m_1 gives a measure based on the satisfied adaptation rules, whereas m_2 defines a metric where enabled services are taken into account. On the other hand metrics m_3 and m_4 take into account additional quantitative information provided by the server and client components, respectively.

It is worth noting that the metrics m_1, m_2, and m_3 can be computed directly by server components (they only need the client protocol specification and the initial adaptor request). Thus, servers may ultimately send back to the client only metric values rather than adaptor proposal specifications, hence increasing further the non-disclosure of information.

Our plans for future work include integrating the adaptation methodology in existing CBSE development environments, so as to allow experimenting and assessing the methodology on large numbers of available components. Another interesting direction for future work is to extend the adaptation methodology to deal with probabilistic descriptions of behaviour, as suggested at the end of Section 4.

References

1. R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–49, 1997.
2. A. Bracciali, A. Brogi, and C. Canal. A formal approach to component adaptation. *Journal of Systems and Software, Special Issue on Automated Component-Based Software Engineering*, (in press). A preliminary version of this paper was published in *COORDINATION 2002: Fifth International Conference on Coordination Models and Languages*, LNCS 2315, pages 88–95. Springer, 2002.
3. A. Brogi, C. Canal, and E. Pimentel. Soft component adaptation. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 85(3), 2003.
4. A. Brogi, C. Canal, and E. Pimentel. On the specification of software adaptation. In *FOCLASA'03*, *Electronic Notes in Theoretical Computer Science (ENTCS)*, 90 (in press), 2003.
5. A.W. Brown and K.C. Wallnau. The current state of CBSE. *IEEE Software*, 15(5):37–47, 1998.
6. G.H. Campbell. Adaptable components. In *ICSE 1999*, pages 685–686. IEEE Press, 1999.
7. C. Canal, E. Pimentel, and J.M. Troya. Compatibility and inheritance in software architectures. *Science of Computer Programming*, 41:105–138, 2001.
8. A. Cechich, M. Piattini, and A. Vallecillo (Eds). Component-Based Software Quality. LNCS 2693. Springer, 2003.
9. A. Di Pierro, C. Hankin, H. Wiklicky. Approximate Non-Interference. In *CSFW'02*, IEEE Press, pages 3–17, 2002.
10. S. Ducasse and T. Richner. Executable connectors: Towards reusable design elements. In *ESEC/FSE'97*, LNCS 1301. Springer, 1997.
11. D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch: Why reuse is so hard. *IEEE Software*, 12(6):17–26, 1995.
12. D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*. 35(2):97–107. 1992
13. P. Inverardi and M. Tivoli. Automatic synthesis of deadlock free connectors for COM/DCOM applications. In *ESEC/FSE'2001*. ACM Press, 2001.
14. B. Jonsson, K.G. Larsen, and W. Yi. Probabilistic extensions of process algebras. *Handbook of Process Algebra*. Elsevier, 2001.
15. R. Mili et al. Semantic distance between specifications. *Theoretical Computer Science*, 247:257–276, Elsevier 2000.
16. R. Milner, J. Parrow, D. Walker. A calculus of mobile processes. *Journal of Information and Computation*, 100:1–77. 1992

17. R. Passerone et al. Convertibility Verification and Converter Synthesis: two faces of the same coin. In *Proc. of the Int. Conference on Computer-Aided Design*. ACM Press, 2002.
18. A. Vallecillo, J. Hernández, and J.M. Troya. New issues in object interoperability. In *Object-Oriented Technology*, LNCS 1964, pages 256–269. Springer, 2000.
19. K. Wallnau, S. Hissam, and R. Seacord. *Building Systems from Commercial Components.* SEI Series in Soft. Engineering, 2001.
20. M. Wermelinger and J.L. Fiadeiro. Connectors for mobile programs. *IEEE Transactions on Software Engineering*, 24(5):331–341, 1998.
21. D.M. Yellin and R.E. Strom. Protocol specifications and components adaptors. *ACM Transactions on Programming Languages and Systems*, 19(2):292–333, 1997.