Bounded non-deterministic planning for multimedia adaptation

Fernando López · Dietmar Jannach · José M. Martínez · Christian Timmerer · Narciso García · Hermann Hellwagner

Abstract This paper proposes a novel combination of artificial intelligence planning and other techniques for improving decision-making in the context of multi-step multimedia content adaptation. In particular, it describes a method that allows decision-making (selecting the adaptation to perform) in situations where third-party pluggable multimedia conversion modules are involved and the multimedia adaptation planner does not know their exact adaptation capabilities. In this approach, the multimedia adaptation planner module is only responsible for a part of the required decisions; the pluggable modules make additional decisions based on different criteria. We demonstrate that partial decision-making is not only attainable, but also introduces advantages with respect to a system in which these conversion modules are not capable of providing additional decisions. This means that transferring decisions from the multi-step multimedia adaptation planner to the pluggable conversion modules increases the flexibility of the adaptation. Moreover, by allowing conversion modules to be only partially described, the range of problems that these modules can address increases, while significantly decreasing both the description length of the adaptation capabilities and the planning decision time. Finally, we specify the conditions under which knowing the partial adaptation capabilities of a set of conversion modules will be enough to compute a proper adaptation plan.

Keywords Bounded non-deterministic multimedia adaptation · Planning · Decision-making · MPEG-7 · MPEG-21

1 Introduction

Planning is a branch of Artificial Intelligence (AI) [1] that has successfully addressed a wide range of automatic decision applications. These applications include, for example, robot control, computer games, expert systems and medical applications [1]. This paper addresses multi-step multimedia adaptation decision-making with AI planning methods. Although the multimedia community has studied this issue to some extent [2, 3], this paper will deal with new challenges that so far have not been addressed.

In MPEG-21, [4] the word *terminal* refers to the physical or logical device (e.g. iPhone, Web browser) in which the user consumes multimedia content. The terminal, the data network and the user preferences collectively define the *usage environment*. Frequently, multimedia adaptation planners [2, 3] utilize the usage environment constraints (or merely the terminal constraints) to represent the goal state of the planner. Multi-step multimedia adaptation enables the adaptation of multimedia content in several steps; for the purpose of this research work each one of these steps will be referred to as a *conversion*.[1] Software modules called *conversion modules*[2] implement these conversions. When a set of available reusable conversion modules is given, it is possible that none of these is capable of adapting multimedia content to the usage environment in one single step. However, multi-step conversions might reach this goal in several steps. For example, one of the conversion modules may be capable of transmoding a video to a set of images and another may be capable of transcoding images to satisfy the terminal image decoding and transmission constraints. Therefore, the video can only be adapted to the constraints of the terminal by executing a sequence of both conversions. In this paper, we describe an AI planner that automatically identifies such sequences together with the parameters needed to execute each conversion. This multimedia adaptation planner will be referred to as the *Planner* module.

1.1 Motivation and objectives

Multimedia adaptation offers content providers added value by increasing the range of terminals that can consume their contents. In addition, multimedia adaptation improves the quality of such content by offering both the content provider and its customers the capability to customize the content to either one's preferences. There is a large amount of research on multimedia adaptation. Special consideration has to be given to multimedia adaptation within the MPEG-21 framework [4]. In this framework a wide variety of methods—such as multi-attribute optimization methods [5], genetic algorithms [6] or knowledge-based methods [1, 3]—have been evaluated to determine the best way to modify media resources in order to make them available in the consumer's usage environment. For the purpose of multi-step multimedia adaptation, mainstream research [2, 3, 7] has relied on MPEG-21 to describe the multimedia adaptation *problem* and on different AI planning techniques to find the *solutions*, i.e., the computational processes that transform the multimedia content so that the terminal can consume it. This paper is also inspired by AI planning and extends the ideas that we introduced in [8].

The techniques described in this paper have been developed to fulfill the objectives of CAIN-21 (*Content Adaptation INtegrator in the MPEG-21 framework*) [9]. CAIN-21 is a multimedia adaptation engine[3] that is compliant with the MPEG-21 framework. The MPEG-21 framework proposes the notion of a *Digital Item* (DI) as a general media resources container that can represent nearly every type of multimedia content. Most adaptation techniques [5, 6] have been designed to deal with particular media (such as JPEG images or MPEG-4 Advanced Video Coding (AVC) video). Conversely, CAIN-21 is not restricted to particular media types. To this end, CAIN-21 supports the integration of third-party pluggable and reusable multimedia conversion modules referred to as *Component Adaptation Tools* (CATs) [10]. A CAT is characterized by the set of parameters describing the media to be adapted and it explicitly specifies the set of conversions that it can perform. The goal of this work is to incorporate into CAIN-21 an automatic decision mechanism. This mechanism will identify which conversions must be executed to adapt a DI and which parameters must be supplied to the conversion modules. For better readability of the paper, we start with a high-level description of CAIN-21 in Sect. 3. In our previous work [11], we developed a mechanism that performs multimedia adaptation decision-making by solving a Constraint Satisfaction Problem (CSP). The main limitation encountered, and the motivation for our current work, is the difficulty in applying this mechanism to problems requiring more than one conversion step. Therefore, this paper focuses on the development and evaluation of the *Planner* module which, provided with an MPEG-21 description of the adaptation problem, computes all the sequences of conversions that adapt a DI (the media resources and corresponding metadata) to the usage environment. This paper does *not* address decisions regarding which of these sequences is best. However, some approaches dealing with this issue will be introduced in Sect. 7.3 and proposed as future work. This current paper includes a set of innovations with respect to previous work [7, 8, 11, 12]:

- *Multimedia properties*. All information required in the adaptation process is consistently described by using so-called *multimedia properties*. Multimedia properties refer to all the elements of the multimedia system, including the media resource, the usage environment, and the adaptation capabilities. To access these properties, CAIN-21 implements a highly efficient addressing mechanism by means of XPath [13] expressions [10]. Section 3 describes this mechanism and Sect. 7.1 summarizes the advantages of using multimedia properties.
- *Domain-specific planner*. The *Planner* uses multimedia properties to make quick multimedia adaptation decisions with a small memory footprint. Standard planners such as the one that we used in previous work [7] or other planning representation schemes such as PDDL [14] were deliberately *not* used in this research; the rationale of this decision is discussed in more detail in Sect. 4.3.

[1] MPEG-21 Part-7 defines a *conversion* as the process that changes the characteristics of a resource. In general a conversion performs the act as defined by the MPEG-21 Part-6 term *adapt*. One *conversion* can be roughly seen as one *set of actions* in a Graphplan-like [20] planner. Sections 2.4 and 4 explain the notion of conversions in detail.

[2] Throughout this paper, the term *module* refers to software modules.

[3] The source code of CAIN-21 together with an online demo of its functionalities is available at http://cain21.sourceforge.net.

- *Tolerating partial description*. In order to allow for a compact description of the CAT adaptation capabilities, the *Planner* includes a mechanism that provides semantics for absent properties.
- *Multi-valued properties*. The *Planner* can manage terminals with multi-valued properties. Multi-valued properties represent alternative values (e.g., $screen\_size = \{320 \times 240, 640 \times 480\}$). Although AI research has addressed this problem (e.g. [15]), previous multimedia adaptation research [1, 2, 7, 12] has not taken into account this condition, which characterizes many practical scenarios.
- *Internal decisions*. CAIN-21 allows for the integration of third-party pluggable CATs capable of making *internal decisions* regarding which adaptation to perform. Thus, the decision process is partially transferred from the *Planner* to the CATs. Depending on these internal decisions, the CATs can produce different outcomes. The multi-valued properties of the CATs have also been transferred to the terminal. Consider, for instance, a specific CAT capable of receiving $format = \{mpeg\text{-}1, mpeg\text{-}2\}$ and producing a video with $format = \{mpeg\text{-}2, mpeg\text{-}4, flv\}$. If the *Planner* has selected these parameters, the CAT can produce any of these outputs and the subsequent CAT/terminal will accept all these video formats.

1.2 Methodology and structure of the paper

Section 2 reviews the state of the art of multi-step multimedia adaptation and the existing AI concepts and techniques that have been selected to make decisions in our adaptation engine. Section 3 provides a high-level view of the modular architecture of CAIN-21 and the control flow through its modules. Section 4 concentrates on its decision phase and describes the conceptual elements of the *Planner*. Section 4 also sketches practical problems that may arise with a complete and rigid multimedia description model and proposes semantics that make it possible to tolerate a partial description. Section 5 discusses the algorithms of the *Planner*, and proves (using a theoretical analysis) that the proposed algorithm is *sound* (i.e. the *Planner* always terminates) and that the produced plan is *finite* and *complete*. Section 6 provides experiments, demonstrations and performance measurements. The major findings as well as their limitations and our future work are summarized in Sect. 7, and lastly Sect. 8 concludes the paper.

2 State of the art

2.1 Multimedia background technology

In the last two decades, at least two different communities have addressed multimedia: the coding community and the metadata community. The main target of the first community is to represent multimedia content compactly and efficiently with standard multimedia formats. One of the important aims of standardization is to reduce the manufacturing costs of terminals capable of consuming multimedia. For instance, the *Joint Photographic Experts Group* (JPEG) is well-known for image compression standards. Examples of audio compression standards are *MPEG-1 Audio Layer 2* (MP2), *MPEG-1 Audio Layer 3* (MP3) and *Advanced Audio Coding* (AAC). The *Moving Picture Experts Group* has defined widely used video compression standards such as MPEG-1, MPEG-2 or MPEG-4 [16]. The MPEG standards divide video formats into profiles and levels. A *profile* defines how complex the encoding is. Technically, a profile defines a subset of the syntax of the specification. For each profile there are a series of levels. A *level* defines a set of constraints on the values that may be taken by the parameters of the specification within a profile. For instance, DVD uses the MPEG-2 *MainProfile@MainLevel* video format. Frequently, uncompressed video is referred to as RAW video and the Waveform Audio Format (WAV) has been widely used to represent uncompressed audio.

Video *containers* such as AVI, MPEG, or WMV usually consist of one *visual stream* and one *audio stream*. For instance, iPhone and Nokia N810 mobile phones use the MPEG-4 video container with an AVC visual stream and an AAC audio stream. Additional audio streams are frequently used to provide support for different languages (e.g., in DVDs). From now on, the term *media resource* will be used in this paper to refer to both media (e.g., MP3 audio) and multimedia resources (e.g., MPEG-4 video container with MP3 audio stream and AVC visual stream).

Further work initiated by the coding community includes automatic multimedia content analysis techniques (such as voice or face recognition) to extract information not explicitly represented in the media. Roughly speaking, the low-level multimedia features are automatically obtained by means of signal analysis techniques. Subsequently, inference techniques are used to obtain high-level descriptions. Multimedia research tries to fill the semantic gap between the low-level and high-level multimedia descriptions (see for instance [17]). As not all these descriptions can be automatically extracted, the metadata community has proposed the semi-automatic or manual annotation of complementary multimedia descriptions. The MPEG-7 [18] standard provides automatically and manually generated descriptions of the multimedia content. In addition, the MPEG-21 [4] standard provides metadata to describe an entire multimedia system.

2.2 Existing multimedia adaptation decision-making techniques

As stated above, the work presented in this paper focuses on multimedia adaptation within the MPEG-21 framework. MPEG-21 Part 2 [4] provides a set of *description tools* for representing media resources and their metadata. Multimedia elements are referred to as *Digital Items* (DIs). MPEG-7 Part 5 [18] specifies description tools which are frequently incorporated in the metadata of such DIs. For the adaptation of DIs to the usage environment, MPEG-21 Part 7 [4] specifies a set of tools referred to as *Digital Item Adaptation tools* (*DIA tools*).[4] *DIA* tools address the problem of maximizing the adapted content quality [19]. The term *DIA description* is used to refer to instances of one or more *DIA* tools. Within MPEG-21 Part 7, the adaptation engines themselves are not normatively specified. The scope of standardization of MPEG-21 Part 7 is limited to the *DIA* tools. Based on this scheme, authors can implement different MPEG-21 Part 7 compliant mechanisms to adapt the existing DIs.

MPEG-21 Part 7 provides, among other tools, two groups of *DIA* tools primarily intended to define the constraints of the environment. The first group is the *Usage Environment Description tools* (*UED* tools) that describe the terminal capabilities, the network characteristics, the user preferences and other environment characteristics (such as location and time zone). The second group is the *Universal Constraints Description tools* (*UCD* tools), which allow for the multimedia consumers and multimedia providers to further constrain the adaptation of the DI.

Typically, multimedia adaptation is done in two phases and executed in a sequential manner [2, 5, 7, 12, 19]. First, a *decision phase* is used to evaluate which adaptation best suits the constraints expressed in the *UED* and *UCD*. Secondly, in the *execution phase*, these adaptation actions are performed on the media and metadata conveyed in the DI. In the decision phase, two main methods exist:

- *Optimization-based methods* [5, 19] aim at finding the adaptation parameters that maximize the quality of the content of the DI after its adaptation to the constraints of the terminal, the network and sometimes the user preferences. This group of methods operates by solving an optimization problem. Frequently this problem is described by metadata where the *UED* tools and *UCD* tools are used together to describe the constraints of the problem. The MPEG-21 Part 7 *AdaptationQoS tools* have been used in this case to describe the possible adaptations that can be performed on a media resource.
- *Knowledge-based methods* [2, 3, 7] have been used primarily to determine whether an action can be used and which parameters must be supplied to adapt the content. These types of methods usually evaluate the concatenation of several actions in a sequence.

Knowledge-based methods frequently rely only on metadata describing the content to make multimedia adaptation decisions. In contrast, optimization-based methods utilize the content of the resource to make decisions. In particular, optimization-based methods measure the quality of the media content after adapting the content in several ways. The variation of the content that better fulfils the constraints of the *UED* and *UCD* and has the highest quality will be selected during the decision process.

Conversely, knowledge-based methods operate on the basis of a description of the media format in the DI (e.g., via the MPEG-7 *MediaProfileType* DS). During the decision phase, these methods search for actions that enable the media format of the DI to be consumable in the target environment. Unlike optimization-based methods, knowledge-based methods usually do not consider the content of the DI resources until the execution phase starts.

Although in theory both the *UED* tools and the *UCD* tools may be used to describe the constraints of the environment, knowledge-based approaches (such as the ones described in [2] and [7]) have used only the *UED* tools to define the constraints of the adaptation problem. Indeed, there exists a coupling between the *UCD* elements that define the constraints of the problem and the *AdaptationQoS* elements that span a solution space making references to the constraints in the *UCD*. The *Planner* presented in this paper makes use of only metadata (DI and *UED* tools), that is, it does not access to the media resource during the knowledge-based decision phase.

Based on the above observations, both optimization and knowledge-based adaptation methods may be combined in sequence. First, the knowledge-based method uses the media format to decide which actions adapt the content to the *UED*. Second, as further explained in Sect. 4.1, "intelligent" actions use optimization-based methods to select their output. At the moment, CAIN-21 includes several CATs capable of performing such optimization decisions.

In this paper, we propose an improvement to the knowledge-based methods. Specifically, we are going to allow for the integration of conversion modules that make decisions during the execution phase.

2.3 Existing AI planning techniques

Previous multi-step multimedia adaptation approaches have frequently used AI planning techniques to select which actions adapt the content to the constraints of the terminal [2, 3, 7]. These AI techniques have been summarized in the following subsections.

[4]MPEG-21 capitalises and italicises XML description tools. This paper adopts this rule.

2.3.1 Planning

In AI, *planning* is the decision-making process that precedes acting [14]. Formally, a planning problem is made up of a finite and recursively enumerable *set of states* $S = \{s_1, s_2, \ldots\}$, a finite and recursively enumerable *set of actions* $A = \{a_1, a_2, \ldots\}$, and a *state transition function* $\gamma(s, a) : S \times A \rightarrow S$, which, given a specific state s_i and a specific action a_j, take us to a different state $s_{i+1} \in \gamma(s_i, a_i)$. In addition, each action is associated with a *set of preconditions pre*(a_i) that must be true before the action can be executed and a *set of effects effects*(a_i) that describes how the state changes when the action is executed. Under such conditions, planning algorithms commits to finding the cumulative effects of these actions to search for sequences of actions that lead from an initial state to a goal state.

Moreover, the set of effects *effects*(a_i) can be further divided into a *set of postconditions post*(a_i) that represent changes in properties of the state and a *set of invariants invariants*(a_i) that represent properties of the state that must not change, i.e., *effects*$(a_i) =$ *post*$(a_i) \cup$ *invariants*(a_i). Traditionally, preconditions are represented as predicates that must be true before the action starts, postconditions are represented as predicates that must be true when the action terminates, and invariants are represented as predicates that keep their true value from the beginning to the end of the executing action.

2.3.2 Neoclassical planners

In the 1980s the computational costs needed to solve the above-described planning problems using classical planners apparently could not be further reduced. However, in the 1990s the computational costs of planning systems were reduced with the rise of techniques that have been qualified as *neoclassical planners* [14]. The most remarkable approach was Graphplan [20]. The main difference between classical planning and neoclassical planning is that in classical planning every node of the search space corresponds to single state in a partial plan, whereas in neoclassical planning nodes correspond to the union of a set of postconditions that can be seen as a set of partial plans. The *planning graph* serves to gather similar actions forming a partially defined *set of actions*. In classical planning, actions were analyzed individually and fully instantiated. Conversely, neoclassical planning analyzes a partially defined set of actions. Additionally, Graphplan proposes to build a reachability graph instead of a reachability tree to reduce the computational costs of the planning algorithm. Even though the first implementation of this idea used forward search, further advances in this area included backwards search, i.e., with the goal state evaluated first (see for instance [21]).

2.3.3 Non-deterministic planning

This subsection introduces the notion of a *non-deterministic planning* and focuses on the difference between bounded and unbounded non-deterministic planners.

Classical and neoclassical planners make two restrictive assumptions [14]:

- *Deterministic actions*: In a given state, actions always produce the same effects. That is, for each action a_i, if the action is applicable for the s_i state, it will lead to no more than a single state s_{i+1}, i.e., $|\gamma(s_i, a_i)| \leq 1$. The planner terminates the expansion of search paths in those states s_i in which for every action a_i it holds that $|\gamma(s_i, a_i)| = 0$.
- *Full observability*: The planner can monitor all the relevant features of the world, meaning that it can recognize all the properties of the states.

Non-deterministic planning relaxes these assumptions. Specifically, non-deterministic planning introduces:

- *Non-deterministic actions*: Actions that under the same conditions (receiving the same input state) produce dissimilar outcomes, i.e., the exact outcome that is going to be produced is unknown before executing the action, i.e., $0 \leq |\gamma(s_i, a_i)| \leq n$, where n is any natural number. For instance, during a manufacturing process the equipment may fail, or throwing a dice has several possibilities, none of them are certain. Deterministic actions are a particular case of the non-deterministic actions in which $|\gamma(s_i, a_i)| \leq 1$.
- *Partial observability*: In some applications the state of the world is only partially observable, and as a consequence different states of the system become indistinguishable. Full observability is a specific case of partial observability in which all the states of the world are distinguishable.

Probably the main problem of non-deterministic planning is that this may result in different execution paths. The usual way to address this uncertainty follows three basic rules:

- *Outcome probabilities*. Non-deterministic actions are modeled by associating probabilities with the outcomes of the actions. This rule allows taking into account the fact that some outcomes are more probable than others.
- *Belief states*. States are replaced by *belief states*, which associate a probability distribution across the state space.
- *Utility function*. Goals are represented via a *utility function*, i.e., numeric values that indicate the level of preference of each possible goal state. Under these circumstances, planning under uncertainty can be seen as an optimization problem where the objective of the planner is to maximize the utility function.

A non-deterministic planner can be *unbounded* or *bounded*. A *bounded non-deterministic planner* is one that can control the parameter of the action to limit the outcome to a subset of its potential instantiations. This paper will focus on the implementation of a neoclassical non-deterministic bounded planner. Specifically, Sect. 4.2 describes that preconditions and postconditions partially define a group of states. Section 4.2 introduces the notion of *conversion states*. Section 4.3 explains that uncertainty is handled using non-deterministic conversion states. Section 4.3 also explains that such conversion states are bounded using so called *source and target parameters*.

2.4 Conversions vs. planning actions

A *conversion* is an action that changes the characteristics of a multimedia resource, represents its partial or complete state before and after its execution, and may perform internal decisions. The notion of *conversion* is the basic operation within the extensible *Planner* of CAIN-21. This term has been chosen following the MPEG-21 Part-7 nomenclature. Conversions have three main features:

(1) *Internal decisions*. Conversions include internal decisions; one conversion corresponds to a group of related actions. Graphplan-like planners also include this idea and produce a planning graph in which each node corresponds to a set of related actions. However, in Graphplan-like planners, these sets of actions are partially instantiated states that will be fully instantiated before the planner terminates. In the case of conversions, the parameters are partially instantiated and the *Planner* never fully instantiates these parameters because these decisions are postponed until the execution phase.

(2) *Bounded non-deterministic actions*. Conversions are bounded non-deterministic actions, which are a specific type of action. In the *Planner*, a conversion state can represent a set of possible states. In this case, the parameter values of the conversion are multi-values.

(3) *Tolerating partial description*. Conversions incorporate semantics for absent properties that will be explained in Sect. 4.5.

3 Architecture and control flow within CAIN-21

This section describes the CAIN-21 software interfaces, the internal architecture and the control flow. Further details can be found in our previous publications [9, 10]. Subsequently, Sects. 4 and 5 focus on the *Planner* module.

3.1 External interfaces

CAIN-21 serves adaptation requests through two external software interfaces (see Fig. 1 below): (1) The *media level transcoding interface* performs *blind adaptation* (i.e., semantics-less adaptation) of a media resource. In addition to the media level, this interface can also perform system level adaptation, i.e., videos composed of one or more audio and visual streams. The media level transcoding operations are implemented inside the *Tlib* module. This module includes conventional software libraries such as *ffmpeg*, *imagemagick* as well as Java Native Interface custom libraries. (2) The *DI level adaptation interface* is responsible for performing system level (semantic or blind) adaptations. In this case, metadata is exploited during the adaptation. During this DI level adaptation, all the modules from (2) to (7) described below are used.

The DI level adaptation interface deals with three types of MPEG-21 DIs: (1) the *Content DI* conveys the media resource together with its metadata; (2) the *Context Repository* encloses usage environment information together with information that the adaptation engine employs on deciding and executing the most suitable set of adaptations. Three *Context DIs* implement the *Context Repository*: the *Usage Env DI* describes the available terminals, networks and user preferences. The *CAT Caps DI* describes the conversion capabilities for each CAT in the *CATs Repository*. The *Properties DI* defines the set of properties to be taken into account and provides the addressing mechanism described in the following subsection; (3) the *Configuration DI* states which *UED*—from the ones available in the *Context DI*—must be used to perform the adaptation. The purpose of the *Configuration DI* is to decouple the *Content DI* and the *Context DIs*. Usage examples and further explanations of this mechanism can be found in [10].

In CAIN-21, the adaptation is performed through the DI level interface and at the *Component* level. An MPEG-21 *Component* includes a media resource (in the *Resource* element) and its metadata (in the *Descriptor* element). The DI level adaptation interface provides two different operations. The first one modifies the existing *Component* and the second operation adds a new *Component* element to the DI. More specifically: (1) the *transform*() operation takes a *Component* from the *Content DI* and modifies its media resource and metadata to adapt it to the usage environment; (2) the *addVariation*() operation takes a *Component* from the DI and creates a new *Component* ready to be consumed in the usage environment. At the end of the *addVariation*() operation, CAIN-21 adds this adapted *Component* to the *Content DI*.

3.2 Architecture

This section provides a detailed description of the CAIN-21 modules. Figure 1 depicts CAIN-21 functional modules and the control flow along the adaptation process. The *Manager* module is responsible for coordinating the whole DI level adaptation process. The modules depicted below the *Manager* perform different tasks initiated by the *Manager*.

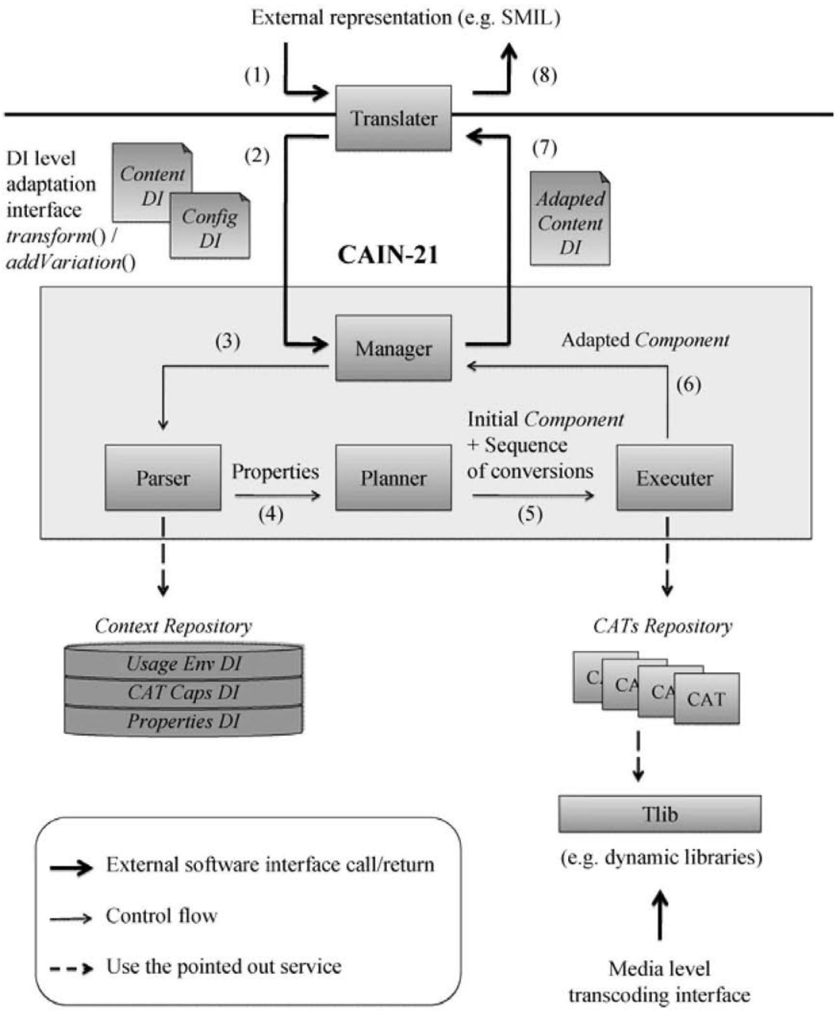


Fig. 1 Modules and control flow within CAIN-21

The distinction between the notion of adaptation module and execution module has been frequently proposed in the multimedia adaptation literature [5, 6]. CAIN-21 also includes this distinction with the *Planner* and the *Executer* modules. The *Planner* uses metadata to decide on the sequence of conversions and parameters that should be executed over a *Component* element of the *Content DI* [8]. Subsequently, the *Executer* is responsible for executing the corresponding sequence of CATs on the initial *Component*. If CAIN-21 receives multiple requests to adapt the same content to the same usage environment, a caching mechanisms speed up this process by bypassing the execution of the *Planner* and *Executer* several times.

The *Context Repository* includes three types of *Context DI*s (see Fig. 1). The *Usage Environment DI* describes the available usage environments using standard *UED* elements (i.e., instances of the MPEG-21 *UED* tools). The *CAT Capabilities DI* describes de adaptation capabilities of the pluggable CATs. There are as many *CAT Capabilities* DIs as CATs are installed in the system. Each *CAT Capabilities* DI contains one or more *ConversionCapabilities* elements. Each *ConversionCapabilities* element describes one conversion that the CAT can perform. Each *ConversionCapabilities* element has a set of valid input and output parameters: each parameter has an associated set of allowed values (multi-valued properties). The relationships among these elements are described in more detail in [10].

CAIN-21 includes an addressing mechanism [10] based on XPath [13] expressions. After parsing the different DIs, all the metadata is represented as a set of properties. With this mechanism, gathering the set of properties of the multimedia adaptation elements is straightforward and highly efficient. The Xalan processor [23] is used to efficiently gather these properties. This set of XPath expressions is collected in the *Properties DI*. The XPath expressions in the *Properties DI* reference data stored in the other DIs. Changes in the set of multimedia properties under consideration do not imply alterations of the underlying source code of the decision algorithm; in fact, these changes must only be done in the *Properties DI*. The *Parser* module resolves the values of the aforementioned properties. Firstly, the *Parser* accesses the *Properties DI* to obtain the set of property keys and corresponding XPath/XPointer expressions. Secondly, after resolving these expressions, the values of these properties are generated. During this step, metadata is loaded from the *Content DI*, *Configuration DI*, *Usage Environment DI* and *CAT Capabilities DI*. The value of the properties that this mechanism obtains can be multi-valued (e.g., *bitrate* = [1000..200000], *audio_format* = {*aac*, *mp3*}). Representing the information by means of properties has additional advantages that will be described in Sect. 7.1.

A wide range of multimedia representation standards exists and CAIN-21 can be integrated into heterogeneous multimedia systems. The *Translater* module is the gateway to other multimedia systems that may be using external technology (i.e., non-MPEG-21 technology) to represent multimedia content (e.g., HTML, SMIL, NewsML, MPEG-4 BIFS). The *Translater* enables the integration of CAIN-21 adaptation services into such heterogeneous multimedia systems. With this purpose, this module transforms the external representation of multimedia into an MPEG-21 compliant input *Content DI* that afterward CAIN-21 processes. In addition, the *Translater* is responsible for transforming the adapted output *Content DI* into its external representation. Different instances of the *Translater* are interchangeable modules created to interact with external multimedia systems.

3.3 Control flow

The numbers in Fig. 1 indicate the tasks control flow in the adaptation process. (1) When interacting with external multimedia systems, the *Translater* transforms the external multimedia representation into a *Content DI* that CAIN-21 can process. (2) The *Content DI* and *Configuration DI* arrive to CAIN-21 via the DI level interface *transform*() or *addVariation*() operations. The *Manager* is in charge of coordinating the whole DI level adaptation process. Although Fig. 1 does not explicitly show it, the *Manager* is in charge of transferring control to the *Parser*, *Planner* and *Executer*. Specifically, (3) the *Manager* initiates the adaptation transferring the control to the *Parser* so that all the metadata can be collected as properties. Other modules use the *Parser* (e.g. to create the adapted *Content DI*), but for simplicity, the figure shows it used only to extract the relevant properties. (4) The *Planner* receives these properties to make a decision on the adaptation to perform. Subsequently, (5) the *Executer* receives the initial *Component* to adapt and the calculated sequence of conversions (together with the corresponding parameters). The *Executer* uses the CATs services to execute the sequence of conversions. The CATs may also change or append information to the *Descriptor* element of the *Component* so that the subsequent CATs may use it. (6) When all the conversions of the sequence have been executed, the *Executer* returns the adapted *Component*. (7) The *Manager* replaces or appends the *Component* to the adapted *Content DI*, depending on the DI level interface operation (i.e., *transform*() or *addVariation*()). (8) Frequently, the adapted *Content DI* may need to be transformed to an external representation and in this case, the *Translater* will perform this transformation.

4 Planning with multimedia conversions

This section provides a conceptual view on the *Planner* and explains how to incorporate partial description to this model. Section 5 both contains algorithm descriptions and a theoretical analysis of the *Planner*.

4.1 Conversion modules that make decisions

In previous multi-step adaptation engines [2, 3, 7], classical and neoclassical AI planners have been used to make all the decisions regarding the actions to perform and the parameters to use in order to achieve a goal state. Frequently, the goal in multi-step multimedia adaptation is determined from the constraints of the terminal. After the planning phase, the actions are then executed in sequence. The primary novel contribution of our approach is to transfer parts of these decisions from the decision phase (*Planner*) to the execution phase (*Executer* and CATs). In this approach, the *Planner* uses the *CAT Capabilities* to determine the constraints that the CATs must obey during their *internal decisions*. The optimization-based methods described in Sect. 2.1 are an example of this class of conversion modules. In [24] we reported several experiments that make internal decisions in order to select optimal parameters settings that maximize the Quality of Service (*AdaptationQoS* tools), satisfying the constraints imposed by the terminal (*UED* tools) and the constraints of the content providers and content consumers (*UCD* tools). In another publication [25], we evaluated the automatic execution of sequences of conversions in which the *Planner* first decides on the input image format, the output visual stream format and the output frame size. In a second step, an *Image2Video* CAT uses *Regions Of Interest* (ROIs) descriptions stored in the *Content DI* to make additional internal decisions that focus on the adapted scene on the ROIs (e.g. faces of the people) to improve the result of the adaptation. The *OnDemandVideoTranscoder* CAT is yet another example of a conversion module that makes internal decisions. This CAT embeds the *ffmpeg* transcoding tool. When certain parameters (such as the frame rate or the bitrate) are not explicitly provided, the *ffmpeg* tool chooses default values, which usually depend on the transcoding operation that will be carried out. More details of these internal decisions will be provided in Sect. 6.

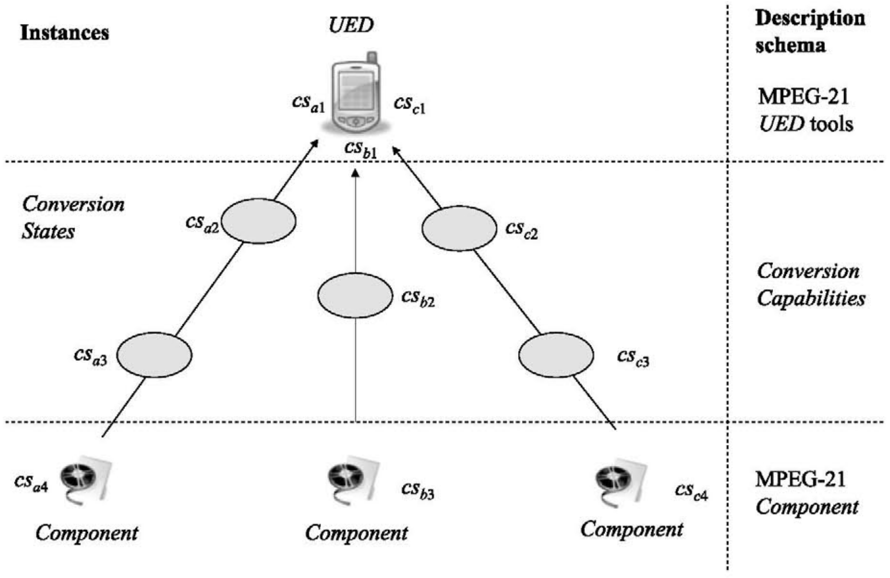


Fig. 2 Conceptual view of the *Planner*

4.2 Conceptual view of the Planner

In Sect. 1 we explained that a *conversion module* comprises one or more actions that modify the media resource. In Sect. 3.2 we then explained that in the context of the CAIN-21 system, a CAT is a pluggable software tool that implements one or more conversion modules. The *CAT Capabilities* and *ConversionCapabilities* documents describe the CATs and the conversions that each CAT implements, respectively. This section provides a conceptual view of the *Planner* and its elements.

The input to the *Planner* are the multimedia *Component* to be adapted, the *ConversionCapabilities* describing the conversion modules installed in the system, and the *UED* providing the current usage environment. We refer to the properties of these three description elements as *conversion states*. The conversion states will be further formalized in the next section. The *Planner* uses conversion states to represent the properties of a *Component* both before and after converting it. Figure 2 shows the conceptual view of the *Planner* together with the description schema used to represent each type of conversion state.

The term *sequence of conversions* soc_i will be used to refer to any sequence of conversion states $cs_n, cs_{n-1}, \dots, cs_1$ that leads from the initial content cs_n (i.e., the *Component* to be adapted) to the adapted content cs_1 (i.e., the *UED*). Since our *Planner* is a backward planner, indices appear in reverse from n to 1. As there may be several ways of adapting the input media resource to the usage environment, the *Planner* builds a *virtual tree of conversions*. In the virtual tree of conversions, the nodes correspond to the conversion states and the arrows correspond to changes in the conversion states. The term *set of sequences of conversions* **SSOC** will refer to all of these sequences of conversions, i.e., $\mathbf{SSOC} = \{soc_1, \dots, soc_k\}$. For convenience reasons, our *Planner* creates several instances of the conversion state that represents the initial content (i.e., all of these instances represent the same *Component* that is going to be adapted). For example, in Fig. 2 $soc_1 = \{cs_{a4}, cs_{a3}, cs_{a2}, cs_{a1}\}$, $soc_2 = \{cs_{b3}, cs_{b2}, cs_{b1}\}$, $soc_3 = \{cs_{c4}, cs_{c3}, cs_{c2}, cs_{c1}\}$ and $\mathbf{SSOC} = \{soc_1, soc_2, soc_3\}$. The output of the *Planner* is the **SSOC** that produces content that is ready for consumption on the terminal.

4.3 Multimedia conversion states

Since these CATs take internal decisions during the execution phase, their outputs depend on these internal decisions and therefore are non-deterministic from the point of view of the planning algorithm, i.e., the *Planner* cannot anticipate the outcomes of these CATs. In contrast to previous multimedia adaptation research [2, 3, 7], we use a bounded non-deterministic planner, i.e., the *Planner* binds the output of the internal decisions to a subset of its potential outputs. The existence of bounded internal decisions is a fundamental reason for the development of a model for the conversion states that we will further described in the subsequent subsections.

4.3.1 Properties-based representation of the conversion states

Preconditions, postconditions and invariants have traditionally been represented with first order logic predicates that have shown to be sufficiently expressive to model many planning problems [26, 27]. This research evaluates an alternative representation of predicates based on properties (i.e., 0-ary propositional predicates). The term *single-valued property* will be used to refer to a label (variable assignment) with only one value (e.g., $width = 320$).

Similarly, the term *multi-valued property* will refer to a label with multiple homogeneous values (e.g., $format = \{mpeg\text{-}1, mpeg\text{-}2, mpeg\text{-}4\}$ or $bitrate = [16000..780000]$).

4.3.2 Conversion capabilities and conversion states

As explained in Sect. 2.3.3, non-deterministic planning has addressed uncertainty in the output of the actions with belief states. In this work, we propose an alternative approach to address non-deterministic actions with conversion states. Specifically, the term *conversion capabilities* cc_i[5] will refer to the range of properties accepted and produced by the conversion module. The term *selected conversion state* cs_i (or *conversion state*, for short) will refer to the subset of properties that the *Planner* is considering for the execution in a sequence of conversions. The term *realized conversion state* $realized(cs_i)$ will refer to the result (set of single-valued property valuations) of executing a non-deterministic conversion. Therefore, given any conversion module the following relation holds: $cc_i \subseteq cs_i \subseteq realized(cs_i)$. In the *Planner*, only the properties in $realized(cs_i)$ have to be single-valued. For example, when given a conversion capabilities element cc_i that accepts $format = \{mpeg\text{-}1, mpeg\text{-}2, divx\}$ and produces $format = \{mpeg\text{-}1, mpeg\text{-}2, mpeg\text{-}4, divx\}$, the *Planner* may generate a conversion state cs_i that accepts $format = \{mpeg\text{-}1, mpeg\text{-}2\}$ and produces $format = \{mpeg\text{-}1, mpeg\text{-}2, mpeg\text{-}4\}$. In this example, the values accepted by cs_i are a subset of the values accepted by cc_i. Similarly, the values produced by cs_i are a subset of the values produced by cc_i. After its execution, the conversion module may end up receiving $format = \{mpeg\text{-}2\}$ and producing $format = \{mpeg\text{-}4\}$. The result of executing the conversion corresponds to $realized(cs_i)$.

4.3.3 Preconditions, postconditions, source and target parameters

Section 2.3.1 described the difference between preconditions, effects, postconditions and invariants. Section 3.2 introduced the *ConversionCapabilites* element of the *CAT Capabilities*. A *ConversionCapabilities* element contains two subelements: the *Preconditions* element describes the conditions (using multi-valued properties); the *Planner* must select one or more of its values before the conversion module can be executed The *Postconditions* element describes the changes (represented as properties) that occur after executing it. These description elements correspond to the conversion capabilities cc_i, preconditions object $pre(cc_i)$ and the postconditions object $post(cc_i)$ respectively. Invariants are not directly represented. Section 4.5 explains that if a property of the conversion module is not described in the preconditions and in the postconditions, the property is invariant, i.e., its value is not altered in the conversion.

The term *selected source parameters* $s\_params(cs_i)$ (or *source parameters*, for short) will be used to refer to the subset of preconditions $s\_params(cs_i) \subseteq pre(cs_i)$ that the *Planner* selects for executing a conversion. Similarly, the *selected target parameters* $t\_params(cs_i)$ (or *target parameters*, for short) will be used to refer to the subset of postconditions $t\_params(cs_i) \subseteq post(cs_i)$ that might be obtained during a specific conversion execution. The selected source and target parameters will be computed during the decision phase. When the target parameters are multi-valued, the *Planner* transfers decisions to the conversion modules. In this case, they may decide which output to produce. In the same manner, the terms *realized source parameters* $s \in s\_params(cs_i)$ and *realized target parameters* $t \in t\_params(cs_i)$ (which names are not shortened in this paper) will be used to refer to the single-valued properties that the conversion module receives and produces during the execution phase.

Figure 3 shows the elements that take part in a bounded non-deterministic multimedia conversion and Fig. 4 shows an example in the CAIN-21 demo. In Fig. 3, the conversion state is shown as an instance of the conversion capabilities: the source parameters are a subset of the preconditions and the target parameters are a subset of the postconditions. The source and target parameters in Fig. 3 will be selected parameters at the end of the decision phase and realized parameters at the end of the execution phase. Note that the conversion states are represented with only one object whose properties (in the case of selected conversion states) may be multi-valued.

The example in Fig. 4 shows a conversion state (source and target parameters) together with its conversion capabilities (preconditions and postconditions). The source parameters must fulfill the preconditions, and therefore the source parameters must be a subset of the preconditions, i.e., $s\_params(cs_i) \subseteq pre(cs_i)$. Similarly, the target parameters must be a subset of the postconditions $t\_params(cs_i) \subseteq post(cs_i)$. Target parameters are always associated with the postconditions and not with the invariants, because invariant properties by definition cannot be changed and are not explicitly modeled. In Fig. 4, the conversion labeled as *mime_image_formats_transcoder* defines in its postconditions that $mime\_type = \{image/jpeg, image/bmp, image/gif, image/ppm, image/png, image/tiff\}$, which means that the conversion module can produce these image formats. In the example, the *Planner* has decided that the output format must be *image/jpeg*, and therefore in the selected target parameter is $mime\_type = \{image/jpeg\}$. In general,

[5]In this document the term *conversion capabilities* (shortened as *cc*) makes reference to the range of properties accepted and produced by the conversion module whereas the term *ConversionCapabilities* makes reference to its description using the MPEG-21 XML elements.

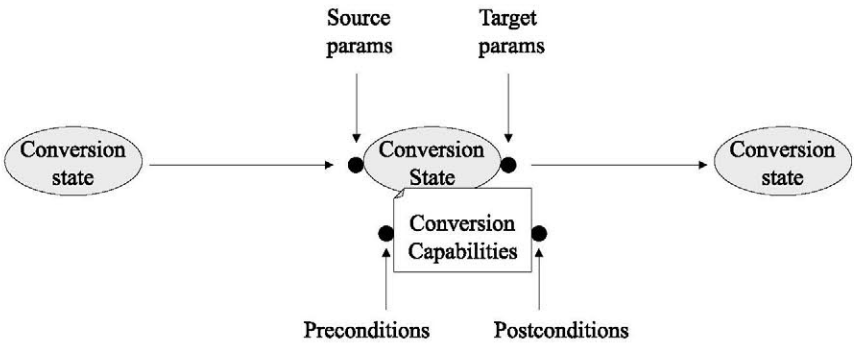
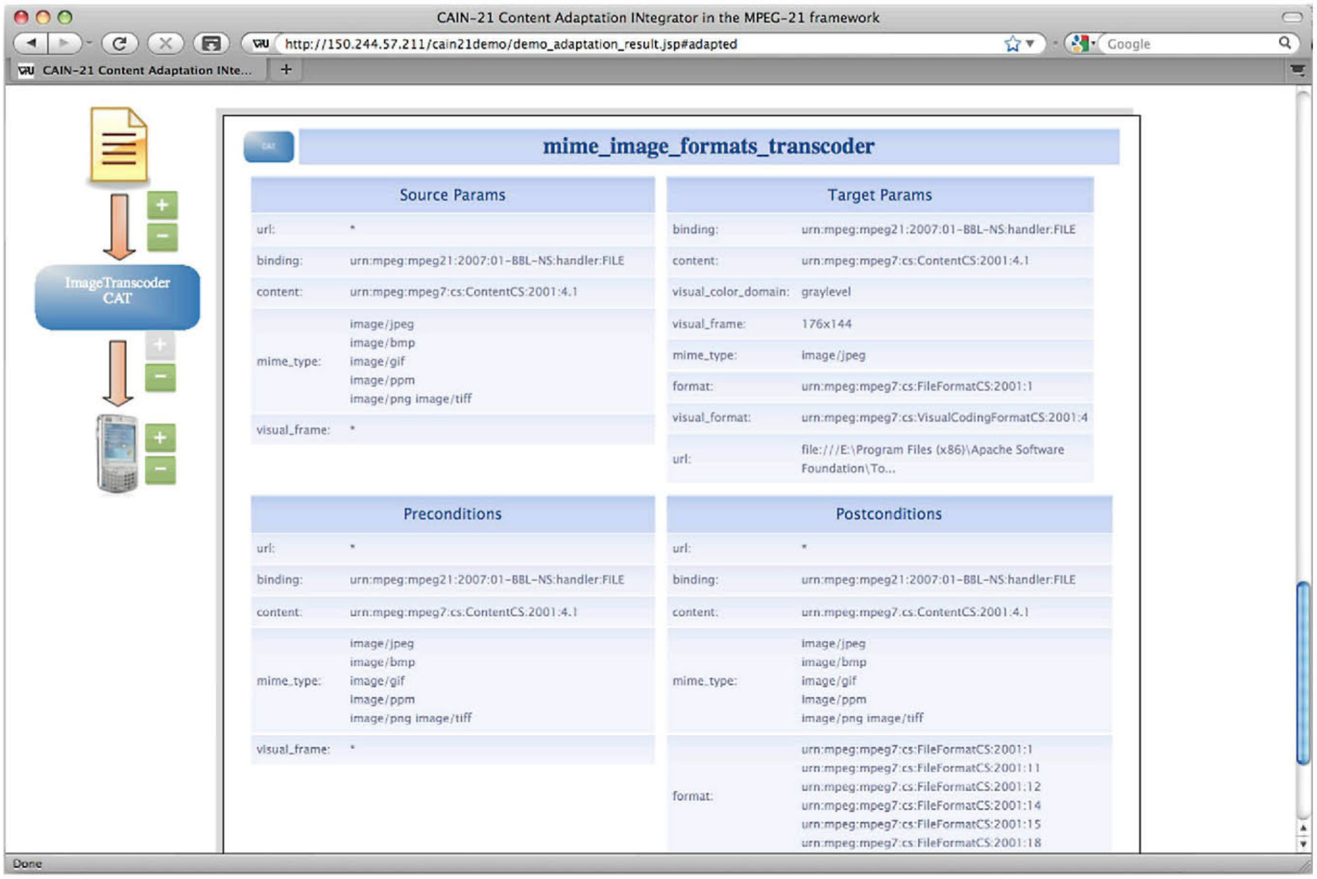


Fig. 3 Elements of a bounded non-deterministic conversion


CAIN-21 Content Adaptation INtegrator in the MPEG-21 framework
http://150.244.57.211/cain21demo/demo_adaptation_result.jsp#adapted
Google
CAIN-21 Content Adaptation INte...
ImageTranscoder CAT
Done

mime_image_formats_transcoder

Source Params	
url:	*
binding:	urn:mpeg:mpeg21:2007:01-BBL-NS:handler:FILE
content:	urn:mpeg:mpeg7:cs:ContentCS:2001:4.1
mime_type:	image/jpeg image/bmp image/gif image/ppm image/png image/tiff
visual_frame:	*

Target Params	
binding:	urn:mpeg:mpeg21:2007:01-BBL-NS:handler:FILE
content:	urn:mpeg:mpeg7:cs:ContentCS:2001:4.1
visual_color_domain:	graylevel
visual_frame:	176x144
mime_type:	image/jpeg
format:	urn:mpeg:mpeg7:cs:FileFormatCS:2001:1
visual_format:	urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:4
url:	file:///E:\Program Files (x86)\Apache Software Foundation\To...

Preconditions	
url:	*
binding:	urn:mpeg:mpeg21:2007:01-BBL-NS:handler:FILE
content:	urn:mpeg:mpeg7:cs:ContentCS:2001:4.1
mime_type:	image/jpeg image/bmp image/gif image/ppm image/png image/tiff
visual_frame:	*

Postconditions	
url:	*
binding:	urn:mpeg:mpeg21:2007:01-BBL-NS:handler:FILE
content:	urn:mpeg:mpeg7:cs:ContentCS:2001:4.1
mime_type:	image/jpeg image/bmp image/gif image/ppm image/png image/tiff
format:	urn:mpeg:mpeg7:cs:FileFormatCS:2001:1 urn:mpeg:mpeg7:cs:FileFormatCS:2001:11 urn:mpeg:mpeg7:cs:FileFormatCS:2001:12 urn:mpeg:mpeg7:cs:FileFormatCS:2001:14 urn:mpeg:mpeg7:cs:FileFormatCS:2001:15 urn:mpeg:mpeg7:cs:FileFormatCS:2001:18

Fig. 4 Conversion state in the CAIN-21 demo available at *cain21.sourceforge.net*

during the decision phase the target parameters can be assigned several values. In this case, the *Planner* is transferring the decision of what value to produce to the conversion module. For instance, if the terminal accepts GIF and JPEG images, the target parameter would be $mime\_type = \{image/gif, image/jpeg\}$.

4.4 Bounded non-deterministic conversions and planning

According to their outcomes, conversions can be divided into three groups: (1) *deterministic conversions*, which are always bound to single-valued properties, (2) *unbounded non-deterministic conversions*, where the outcome is not always the same and the planner cannot select the outcome, and (3) *bounded non-deterministic conversions*, where the outcome can vary and the planner selects a subset of the postconditions. In this paper, multimedia conversions are modeled as bounded non-deterministic conversions where a conversion can be executed by providing a set of target parameters. Subsequently, the execution of each conversion may produce these parameters (a subset of the set of the postconditions).

An important difference between the bounded non-deterministic planner (our *Planner*) and other multimedia planners (such as the one in [3]) is that the bounded non-deterministic planner must determine the source and target parameters settings to execute the conversions.

Finally, the bounded non-deterministic planner should not be confused with other techniques such as continuous planning [22] or planning under uncertainty [14]. These latter two techniques implement unbounded non-deterministic planners wherein the actual outcome of an action is not known before its execution. Such situations can for example arise, when the execution of an action fails due to an unforeseen change in the environment that has occurred since the planning phase. Therefore, this other group of techniques requires the existence of *contingency decisions*, that is, alternative decisions that are chosen depending on the outcome of executing an action (such as to repeat the action or to select an alternative action). Contingency decisions are needed in continuous planning and planning under uncertainty because if contingency decisions were not used, the plan would not always lead to a goal; the goal would be only reached when there was no "failure" in the sequence of actions. Moreover, these contingency decisions have the effect that no linearly ordered sequence of conversions exists. Bounded non-deterministic planners, however, can build linear sequence of actions that always succeed. The planner developed in this paper uses conversions that—by definition—always produce one of the selected solutions. Therefore, this planner can calculate all the feasible sequences of conversions before the conversions are executed.

4.5 Tolerating partial description

Multimedia adaptation systems such as CAIN-21 can be used in large real-world multimedia platforms, which might involve a relatively large number of multimedia properties. For the viewpoint of both the CAT implementer (i.e., the third-party that implements a pluggable CAT) and the user that needs to adapt his/her multimedia content, supplying accurate values for the entire set of properties may become tedious and error prone. For these reasons the *Planner* was designed to operate with partial description. Specifically, the CAT implementer is not forced to provide detailed description of all the properties of the *CAT Capabilities*. Besides, the *Content DI* (i.e., the *Component* to be adapted) and *Context DI* (i.e., the *UED*) may arrive at the adaptation engine without all their properties values. The notion of partial description relates to the notion of partial observability introduced in Sect. 2.3.3. When only a partial description is available, the *Planner* assumes default meanings for the properties of the conversions capabilities. The following subsections describe the mechanisms that tolerate partial description.

4.5.1 Semantics of the properties of the conversion capabilities

We propose the following *semantics* for the properties of the preconditions, postconditions and invariants of the conversion capabilities. These semantics apply in extended form also to the corresponding properties in the conversion states:

- *Required preconditions*. If a property appears in a precondition of a conversion capability cc_i, this indicates that the conversion *requires* this property in the corresponding source parameter of the conversion state cs_i. Specifically, such a source parameter (represented as a property) must be a subset of the corresponding precondition property values.
- *Produced postconditions*. If a property appears in a postcondition of a conversion capability cc_i, this must be interpreted in the sense that the corresponding conversion state cs_i *produces* such a property. In this case, the conversion state may create the property (represented as a target parameter) if it does not exist in the source parameter, or modify it if it exists in the source parameters.

4.5.2 Incompleteness semantics

In addition to providing semantics to existing properties, *incompleteness semantics* have been implemented. These semantics define how to deal with absent or partially defined properties in the conversion capabilities and corresponding conversion states. Specifically:

- *Optional properties*. If a property does not appear in the preconditions of a conversion capability cc_i, this must be interpreted in the sense that *every value is acceptable, including the situation where it is not provided at all.*
- *Wildcard properties*. If a property is marked with a wildcard in the preconditions of a conversion capability cc_i, this configuration must be interpreted in the sense that *every value is acceptable, but it must be provided.*
- *Preserved properties*. If a property appears neither in the preconditions nor in the postconditions of a conversion capability cc_i, this situation must be interpreted in the sense that the conversion state cs_i *preserves* the value of such a property, that is, the target parameters take the property values of the corresponding source parameters without changes.

Note that the preserved properties semantic forces the *Planner* to assume that whenever a property does not appear in the description of the conversion capabilities, this property is not modified in the conversion. Therefore, the CAT that implements such a conversion must refrain from modifying this property. Note that this is a risky assumption, as the CAT implementers are assumed to be careful and precise in their design. The other option would have been to force the CAT implementers to annotate all the invariant properties that the conversion module does *not* modify, which would become tedious for the CAT implementers. Alternatively, the algorithm that searches for the plan would have to systematically discard all the conversion modules that are not completely annotated.

4.5.3 Ignored properties and accumulated effects

In the incompleteness semantics there is a peculiar situation when a property appears in the preconditions of a conversion module, but does not appear in the postconditions or in the invariants of the conversion. In this case, it is not defined what the conversion does with this property and we say that the conversion capability *ignores* the property. Therefore, it is impossible to make any assumption involving the value of this property in the target parameters. The term "ignore" must not be interpreted in the sense that the execution of the conversion module necessarily "loses" the property, but in the sense that the conversion module does not provide information about what is going to happen with this property. An example of this is a conversion capability that defines the maximum frame rate that a conversion module accepts, but does not define the maximum frame rate that it can produce. This situation may arise simply because the media produced by the conversion does not have a frame rate at all (e.g., it produces an image as a summary of a video clip), or because it does not specify the output frame rate (although the conversion module produces a video).

To avoid ignoring properties, we introduced the notion of *accumulated effects,* which refer to the set of properties that are produced or modified through a sequence of conversions. Given a conversion state cs_i, an algorithm can compute these accumulated effects as the union of the properties in the source and target parameters of the conversion states that appear along the path from cs_i to cs_1. During each step of the sequence of conversions, the *Planner* imposes that the accumulated effects must always increase or remain, that is, the *Planner* does not allow the existence of conversion capabilities that ignore properties (according to the above-mentioned *ignore* incompleteness semantic). In CAIN-21, the *Parser* currently detects this condition and reports it to the user.

5 Algorithm and theoretical analysis

5.1 The Planner algorithm

Figure 2 shows that the conversion states of a multimedia adaptation problem are instances of the MPEG-21 *Component* to be adapted, the existing *ConversionCapabilities* elements and the *UED* tool. The *Planner* algorithm carries out a backwards search that finds all the sequences of conversion states that are capable of adapting the *initial conversion state* (the *Component*) to the *goal conversion state* (the *UED*). Section 4.2 proposes to represent those sequences of conversions soc_i as a *set of sequences of conversions* **SSOC** $= \{soc_1, soc_2, \ldots, soc_k\}$. Algorithm 1 is the top-level control structure of the *Planner* that computes these sequences. It starts by extracting the properties of the goal conversion state from the *UED*. Subsequently, *getSetOfSequenceOfConversions*() (explained in Sect. 5.2) determines the sequences of conversions that lead to the goal conversion state. As the goal conversion state is not part of the sequences of conversions that *getSetOfSequenceOfConversions*() produces, Algorithm 1 adds the goal conversion state to the tail of each sequence.

Every sequence of conversion states must have at least two conversion states: the initial conversion state and goal conversion state. Thus, if the *Component* to be adapted (the *initial* conversion state) fulfils the *UED* constraints (the *goal* conversion state) without further changes, the corresponding sequence of conversion states would only include these conversion states. Conversely, if there is no sequence of conversions that can adapt the *Component* to the *UED*, Algorithm 1 would return an empty **SSOC**. In this case, CAIN-21 reports this circumstance to the user using plain English explanation messages (e.g., "There is no sequence of CATs capable

Algorithm 1 Main structure of the *Planner* algorithm

```
Inputs: problem // The Component to be adapted, the CAT Capabilities and the UED
Outputs: SSOC // Set of sequences of conversions that the Planner has found.
Vars: goal_cs // A conversion state with the properties of the UED
      empty_soc // Starts backwards search with an empty sequence of conversion states
SSOC planner(problem)
  goal_cs = problem.getUEDConversionState();
  SSOC = getSetOfSequenceOfConversions(problem,goal_cs,[]);
  for (each soc in SSOC)
    soc = soc + {goal_cs} // Add goal_cs to the tail of soc
  return SSOC;
```

Algorithm 2 Set of sequences of conversions

```
Inputs: problem                       // The Component to be adapted, the CAT Capabilities and the UED
        subgoal                       // The goal or subgoal in each recursive step
        visited_conversion_states     // List of conversion state objects that have already been evaluated
Outputs: SSOC                         // Set of sequences of conversions that the Planner has found.
Vars: prospective                     // The set of conversion states (instance of either a ConversionCapabilities element or
                                      // the Component element) that matches with the current subgoal
      feasible                        // Subset of prospective conversion states that Algorithm 4 has selected
      sub_ssoc                        // The SSOC that reach the current subgoal
SSOC getSetOfSequencesOfConversions(problem, subgoal, visited_conversion_states)
  prospective = getProspectiveConversionStates(problem, subgoal);
  feasible = {};
  for (each cs in prospective)
    if (isFeasibleConversion(cs, subgoal, visited_conversion_states))
      feasible.add(cs);
  SSOC = {};
  for (each cs in feasible)
    if (cs isa Component)
      soc = {cs};
      SSOC.add(soc);
    else
      visited_conversion_states.add(cs);
      sub_ssoc = getSetOfSequencesOfConversions(problem, cs, visited_conversion_states);
      visited_conversion_states.remove(cs);
      for (SequenceOfConversions soc : sub_ssoc)
        soc.addTailStep(cs);
        SSOC.add(soc);
return SSOC;
```

of converting DivX files to MPEG-4"). This message helps the adaptation engine administrator to configure the CATs installed in the adaptation engine.

5.2 Computing the set of sequences of conversions

Algorithm 2, which gives details of the *getSetOfSequenceof-Conversions*() function, is a recursive process that produces a virtual tree of conversions (introduced in Sect. 4.2). An example of this virtual tree of conversions is given in Fig. 5. This example shows the changes in the source and target parameters of the virtual tree of conversions' nodes. Thus, what Algorithm 2 produces is a set of sequences of conversions **SSOC** $= \{soc_1, soc_2, \ldots, soc_k\}$ that corresponds to the different paths.

The algorithm receives in the *subgoal* parameter either the goal conversion state (i.e., instance of the *UED*) or any subgoal conversion state (i.e., instance of either a *ConversionCapabilities* element or the *Component* element) in the virtual tree of conversions. In order to prune the search, the algorithm also receives a list of visited conversion states. Algorithm 2 has two parts:

(1) The first loop determines which conversion states can precede the current goal conversion state. The term *prospective conversion states* refers to the conversion states (instances of the conversion capabilities available in the system) that match (according to Algorithm 6 explained in Sect. 5.5) the current goal conversion state. The term *feasible conversion states* refers to those prospective conversion states that contribute to the adaptation. They are explained in Sect. 5.3.
(2) The second loop recursively composes the set of sequences of conversions that go from each feasible conversion to the current goal. The visited conversion states are also used to avoid infinite loops through the same group of states.

5.3 Prospective and feasible conversion states

The prospective conversion states are obtained from both the *Component* to be adapted and the *ConversionCapabilities* of the CATs currently installed in the system. In Algorithm 3, the first part determines whether the conversion capabilities that correspond to the *Component* to be adapted matches (according to Algorithm 6) with the current goal conversion

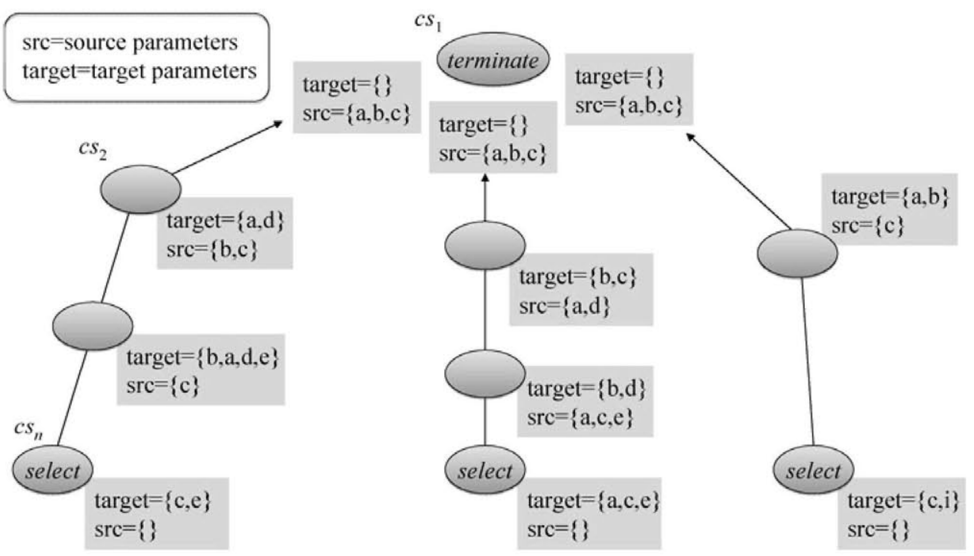


Fig. 5 Example of a virtual tree of conversions

Algorithm 3 Prospective conversion states

```
Inputs: problem // The Component to be adapted, the CAT Capabilities and the UED
        subgoal // The goal of subgoal in each recursive step
Outpus: cs_set  // Set of prospective conversion states.
Vars: child_cs  // Conversion state that matches with the current goal conversion state
SSOC getProspectiveConversionStates(problem, subgoal)
  cs_set = {};
  child_cs = match(subgoal, problem.getComponentConversionCaps());
  if (child_cs ≠ null)
    cs_set.add(child_cs);
  for (each cc in problem.getCATsConversionCaps())
    child_cs = match(subgoal, cc);
    if (child_cs ≠ null)
      cs_set.add(child_cs);
  return cs_set;
```

state. The second part of Algorithm 3 obtains the set of conversion capabilities that match the current goal.

The set of feasible conversion states is a subset of the prospective conversion states that contribute to the adaptation. Algorithm 4 shows the three conditions that make a prospective conversion state become a feasible conversion state. (1) It must not be previously visited, (2) it must contribute to the progress of the adaptation (according to Algorithm 5), and (3) the initial conversion states must not have unsatisfied preconditions. Unsatisfied preconditions mean that properties exist which have not been adapted through the sequence of conversions according to the constraints of the *UED*.

5.4 Obtaining the conversion states that contribute to the adaptation process

In each step within the backwards search process through the virtual tree of sequences of conversions of Fig. 5, the *Planner* must determine which conversion states contribute to the adaptation process. The criterion to determine their contribution is as follows:

- If the intersection between the property list of the target parameters of the cs_{i+1} conversion state and the property list of the target parameters of the cs_i goal conversion state is a set with one or more empty property values, then this configuration means that cs_{i+1} contributes (with the empty properties in the intersection) to the progress of the adaptation. In this case, the cs_{i+1} conversion state is maintained.
- Otherwise, it means that the same outcome can be reached with cs_i, and the cs_{i+1} conversion state is discarded.

Consider, for example, that the target parameters of cs_i are *visual_format* = {*mpeg*-1, *mpeg*-2} and *audio_format* = {*mp2*, *mp3*} and the target parameters of cs_{i+1} are *visual_format* = {*mpeg*-4} and *audio_format* = {*mp2*, *mp3*}. In this case, the intersection of the *visual_format* values is

Algorithm 4 Feasible conversion states

```
Inputs: cs                          // Prospective conversion state to be determine as feasible conversion state
        subgoal                     // The goal of subgoal in each recursive step
        visited_conversion_states   // List of conversion state objects that have already been evaluated
Outputs: Returns whether the prospective conversion state is feasible or not
boolean isFeasibleConversion (cs, subgoal, visited_conversion_states)
  if (visited(cs, visited_conversion_states))
    return false;
  if (!contribute(cs, subgoal))
    return false;
  if (cs isa Component AND NOT cs.hasUnsatisfiedPreconditions())
    return false;
  return true;
```

Algorithm 5 Conversion states that contribute to the adaptation

```
Inputs: cs      // Prospective conversion state. The algorithm determines whether it contributes to the adaptation
        subgoal // The goal of subgoal in each recursive step
Outputs: Whether the cs conversion state contributes to the adaptation (that produces the subgoal conversion state) or not
boolean contribute(cs, subgoal)
  intersected_props_values = cs.getTargetParams() ∩ subgoal.getTargetParams();
    for (each p intersected_props_values)
      if (p.isEmpty())
        return true;
    return false;
```

an empty set. This means that *mpeg-4* visual format can be adapted by adding cs_{i+1} to the virtual tree of conversions. Note that this discarding condition never occurs when the cs_i conversion state is the goal conversion state. This happens because *getUEDConversionState*() always returns a conversion state with an empty set of target parameters and in this case the source parameters corresponds to the properties of the *UED* (see Fig. 5).

Algorithm 5 starts by determining the intersection between the target parameters of cs_i and cs_{i+1}. Subsequently, a property with empty values denotes the existence of states that cannot be reached without cs_{i+1}. This means that cs_{i+1} contributes to the adaptation.

5.5 The matching process

Classical planning algorithms represent changes in the state of the system with a state transition function (introduced in Sect. 2.3.1). Figure 6(a) shows the elements involved in the state transition function. The state transition function $s_{i+1} = \gamma(s_i, a_i)$ evaluates the preconditions of an action $a_i \in A$ and determines if the action can be applied to the input state $s_i \in S$ and the output state $s_{i+1} \in S$. Accordingly, classical planning algorithms are restrained by three rules:

- States are fully observable. The system has complete knowledge of the world, and therefore it observes the outcomes in a single state, i.e., the current unique state of the system.
- Actions are deterministic. Actions have single-valued states, i.e., if applicable during the s_i state, each action a_i leads to a single new state s_{i+1}, so that $|\gamma(s_i, a_i)| \leq 1$.
- Actions are always unbounded, i.e., they have no source and target parameters. As the result of executing an action is always the same and it makes no sense to use parameters to select these single-values properties.

This research proposes that relaxing these traditional assumptions allows modeling and solving a wider range of problems. Thus, our proposal for a bounded non-deterministic planner replaces the state transition function $\gamma(s_i, a_i)$ with a *matching process* $\gamma(cs_i, cc_{i+1})$. Figure 6(b) shows the elements of the matching process. The matching process is a function that receives as input the i-th conversion state cs_i together with the $(i+1)$-th conversion capabilities cc_{i+1} and produces as output the $(i+1)$-th conversion state cs_{i+1}. In other words, the matching process identifies the cs_{i+1} conversion state whose target parameters are acceptable by the cs_i conversion state. In this case, the conversion capabilities cc_{i+1} *matches* with the conversion state cs_i. The rationale behind this change is further developed in Sect. 7.1. This algorithm is invoked from Algorithm 3 to obtain the prospective conversion states. The arrows in Fig. 6 show that

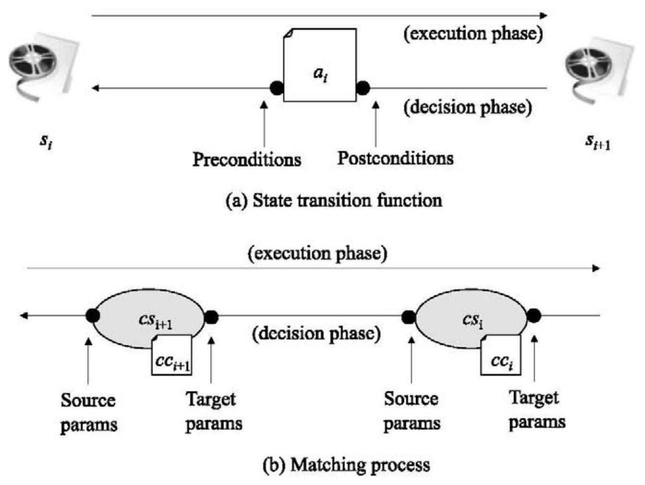


Fig. 6 Elements of the state transition function and of the matching process

the decision phase progresses from the target to the source, whereas the execution phase progresses from the source to the target. In this work, indices are always assigned to the states according to the order in which the decision phase evolves.

The state transition function used in classical planning algorithms creates an explicit intertwined sequence of states and actions as shown in Fig. 6(a). On the other hand, as Fig. 6(b) shows, the implied state of the *Component* that is being adapted can be removed from the sequence of conversion states without losing information because the conversion state impliedly represent the state of such a *Component* through its source and target parameters. Specifically, the source parameters represent the *Component* before executing the conversion module and the target parameters represent the *Component* after executing the conversion module. In Sect. 4.3.3 the source and target parameters of a conversion state cs_{i+1} have been defined as a subset of, respectively, the preconditions and postconditions of its conversion capabilities cc_{i+1}. Therefore, the new conversion state object cs_{i+1} would include its corresponding conversion capabilities cc_{i+1} (represented as preconditions and postconditions) together with the source and target parameters that the matching process has selected for the conversion state. The details of the matching process algorithm are given in Algorithm 6.

5.6 Analysis of the Planner algorithm

As the algorithm progresses from the goal conversion state to the initial conversion state, the source parameters are removed and the target parameters are added. Therefore, at the end of Algorithm 1 the goal conversion state must only contain source parameters. Likewise, the initial conversion state must only contain target parameters (see Fig. 5). That being said, note that:

(1) The number of properties in the goal conversion state is always the same, regardless of the sequence of conversions that reaches the goal conversion state, i.e., in Fig. 5 all the sequences must lead to the same set of source parameters ($\{a, b, c\}$ in this example). However, since the properties of the goal conversion state could be multi-valued, the values of the properties that each sequence of conversions produces may vary. This representation is consistent with the capability of the *UED* to accept alternative properties. For instance, suppose that in Fig. 5 the goal state accepts $a = \{mpeg\text{-}2, mpeg\text{-}4\}$, what means that the a property must be produced by every sequence of conversions. However some of these conversions might produce $a = \{mpeg\text{-}2\}$, other sequences might produce $a = \{mpeg\text{-}4\}$, and even another group of conversions might produce both $a = \{mpeg\text{-}2, mpeg\text{-}4\}$.

(2) It is not guaranteed either that the source parameters will be monotonically removed, or that the target parameters will be monotonically added. However, the *accumulated effects* characteristic introduced in Sect. 4.5.3 guarantees that the accumulated effects (the union of the source and target parameters) of the cs_{i+1} conversion state are a superset of the source parameters of the cs_i conversion state. For instance, in Fig. 5, the goal conversion state has the source parameters $src = \{a, b, c\}$ and the cs_2 conversion state has the accumulated effects $src = \{a, d\} \cup target = \{b, c\} = \{a, b, c, d\}$, which is a superset of the source parameters of the goal conversion state. The next section will make use of the accumulated effects to prove that the *Planner* algorithm always terminates.

5.7 Soundness and completeness

This subsection develops three theorems that prove that the *Planner* algorithm given in Algorithm 1 is *sound* (always terminates). In addition, this subsection proves that the plan is *finite* (always terminates) and *complete* (contains all the available solutions).

Theorem 1 *Algorithm* 1 *is sound* (*always terminates*).

Proof Given any goal conversion state cs_i, representing either the *UED* (cs_1) or any subgoal conversion state, it can be derived that:

(1) According to the prerequisite given in Sect. 4.5.3 and further described in the previous subsection, the set of accumulated effects remains stable or increases in size in each step from cs_i to cs_{i+1}. Therefore, in each conversion step going backwards from cs_1 (the goal conversion state) to cs_n (the initial conversion state), the number of accumulated effects never decreases (see Fig. 5). This

Algorithm 6 Matching process

```
Inputs: cs0   // Conversion to be reached (subgoal)
        cc1   // The ConversionCapabilities or Component to be considered
Outputs: cs1 // Conversion state (an instance of cc1) that matches with cs0 or null if there is no matching
Vars: key     // The key of a property. Properties contains one key and a set of values
        p     // Used to refer to each postcondition of cs1
        q     // Used to refer to each source parameter of cs0
ConversionState match(cs0, cc1)
  cs1 = new ConversionState(cc1);
  // Step 1: Gather source parameters that come from cs0
  for (each q in cs0.getSourceParams())
    key = q.getKey();
    p = cc1.getPostcondition(key);
    if (p $\neq$ null)
      r = p $\cap$ q;
      if (r.isEmpty())
        // Fails since cc1 is not capable of producing a property requested by cs0
        return null;
      cs1.addTargetParam(r);
    else
      cs1.addSourceParam(q);
  // Add the preconditions of cc1 to the source parameters of cs1 if they have not been taken from cs0 in step 1
  for (each p in cc1.getPreconditions())
    key = p.getKey();
    if (cs1.getSourceParam(key) = null)
      cs1.addSourceParam(p);
  // Add the postconditions of cs1 to the target params of cs1 if they have not been selected during step 1
  for (each p in cc1.getPostconditions())
    key = p.getKey();
    if (cs1.getTargetParam(key) = null)
      cs1.addTargetParam(p);
  return cs1;
```

idea is similar to the one proposes in [27] where "delete lists" are removed from a STRIPS-like planner in order to guarantee decidability.

(2) As the number of *ConversionCapabilities* elements installed in CAIN-21 is finite, Algorithm 3 always expands a finite number of prospective conversion states (as defined in Sect. 5.2). In practice, cs_1 always has a finite number of source parameters (number of properties of the *UED*), which must be reached by its subsequent conversion states during each step of the sequence of conversions from cs_i to cs_{i+1}. In this case, only two situations may occur: (a) the number of accumulated effects increases from cs_i to cs_{i+1}, or (b) the number of accumulated effects from cs_i to cs_{i+1} remains the same. However, in the second situation Algorithm 2 cannot use conversion states stored in *visited_conversion_states* again. Therefore, Algorithm 2 always terminates (and thus Algorithm 1). In the worst-case scenario, Algorithm 2 will terminate when all the conversion states are stored in *visited_conversion_states*. □

Theorem 2 *The plan is finite* (*always terminates*).

Proof As, according to Theorem 1, Algorithm 3 always expands a finite number of conversion states, and because the *visited_conversion_states* guaranties that the virtual tree of conversions has no cycles, the plan reaches all the feasible conversion states in a finite number of steps. □

Theorem 3 *The plan is complete*, *i.e.*, *the virtual tree of conversions covers all the feasible conversion states.*

Proof As, according to Theorem 1, all the conversion states that reach the goal conversion state are expanded and as according to Theorem 2 the plan is finite, it can be concluded that the plan reaches all the feasible conversion states. □

The result of this subsection formalizes the sufficient conditions that allow the *Planner* algorithm to compute a finite and complete plan. This computation can be executed even when the adaptation capabilities are only partially defined. In fact, the only condition that the conversion capabilities have to comply with is that the accumulated effects always increase or remain unchanged. That is, the *Planner* does not allow for the existence of conversion capabilities that ignore properties (according to the semantic explained in Sect. 4.5.3).

6 Experiments and demonstrations

6.1 Claims and hypotheses

In the subsequently reported experiments and demonstrations, we focus on providing evidence for the following claims:

(Claim 1) A partial description of the adaptation capabilities suffices for computing all the feasible adaptation plans.
(Claim 2) Requiring only partial description of the conversion modules significantly reduces the description length of the adaptation capabilities.
(Claim 3) Requiring only partial description of the conversion modules significantly reduces the decision time.
(Claim 4) Partial decision-making provides advantages for decision-making with respect to systems in which the conversion modules do not perform additional decisions.

Claim 1 has been theoretically proven in Sect. 5.7. Claim 4 is not suitable for an empirical study, but it is suitable for an informal demonstration that has been accomplished in Sect. 6.5. Based on Claim 2 and Claim 3, three hypotheses are going to be tested in this section.

(H1) The average-case computational cost of the *Planner* is significantly lower than the theoretical worst-case computational cost.
(H2) The adaptation capabilities description size decreases significantly when partial description is allowed.
(H3) The decision time decreases significantly when partial description is allowed.

6.2 Theoretical worst-case computational costs of the Planner

This section analyzes the theoretical worst-case computational cost of Algorithm 2 and Algorithm 6, which are the core algorithms of the *Planner*. The other algorithms described in Sect. 5 serve as subfunctions.

Let C be the number of conversion capabilities elements existing in the available CATs, and N the number of properties of the conversions capabilities to be considered in the matching process of Algorithm 6. In the worst-case (assuming that the same conversion capabilities are not instantiated more than once), Algorithm 2 would be invoked $C!$ times. In the worst-case Algorithm 6 would be invoked C times in Algorithm 2, i.e., in the worst-case Algorithm 6 is invoked $C \cdot C!$ times, which is of the order of $(C+1)!$. Assuming that N is the upper bound of properties in a conversion capabilities element, Algorithm 6 would perform in the worst-case N^2 property comparisons during each invocation. Thus, the theoretical worst-case computational cost is of the order of $N^2 \cdot (C+1)!$ with respect to the number of comparisons between properties.

6.3 Empirical methodology, dataset and metrics

This subsection justifies the empirical methodology, dataset and metrics used in the experiments. The following subsections perform the empirical analysis and present the results.

To increase the objectivity of the experiments, we have selected adaptation tests aiming to cover different media transcoding operations. In this way, different sets of properties would be involved in the experiments. Specifically, we have used most of the adaptation tests available in the CAIN-21 demo, but distributing them into four groups: I→I (Image to image), I→V (Image to video), V→V (Video to video), SVC (Scalable video coding). Tables 1 and 3 show these 24 adaptation tests distributed in four groups: 6 image to image adaptations, 6 image to video adaptations, 6 non-scalable video adaptations and 6 scalable video adaptations. In Tables 1 and 3, the ID of the terminal corresponds to each adaptation test of the dataset. In Table 3, the second column shows the type of these tests. The adaptation tests use a different number of *CATCapabilities* elements and therefore a different number of *ConversionCapabilities* elements. In Tables 1 and 3 C represents the number of *ConversionCapabilities* elements used in each adaptation test. In addition, each *ConversionCapabilities* element contains multiple properties. Therefore, the number of properties involved in the experiments is higher than the number of adaptation tests. Table 2 shows the specific number of properties, N, for each *ConversionCapabilities* element. The tests were implemented and executed in the JUnit testing framework [28]. The source code of these tests is publicly available at *cain21.sourceforge.net*. All these tests were executed in the same hardware, a Mac Book Pro with a 2.4 GHz Intel Core 2 duo and 4 GB of RAM.

To study H1, we have measured the average-case computational cost of the *Planner* for each test and compared it with the corresponding theoretical worst-case computational cost. Specifically (see Table 1), we counted the number of invocations to the functions that implement Algorithm 2 and

Table 1 Number of invocations of the algorithms

ID of the terminal	C	Invocations				Theoretical worst-case		
		Min steps	Algorithm 2	Algorithm 6	Cmps	Algorithm 2 $C!$	Algorithm 6 $(C+1)!$	Cmps $N^2 \cdot !(C+1)$
gray_images_viewer	4	1	4	20	96	2.40e+01	1.20e+02	1.08e+05
jpeg_images_viewer	4	1	4	20	94	2.40e+01	1.20e+02	1.08e+05
images_viewer_without_resolution	4	1	4	20	91	2.40e+01	1.20e+02	1.08e+05
audiovisual_mobile_1	4	1	2	10	47	2.40e+01	1.20e+02	1.08e+05
bmp_image_viewer	4	1	4	24	108	2.40e+01	1.20e+02	1.08e+05
png_image_viewer	4	1	4	24	106	2.40e+01	1.20e+02	1.08e+05
image_viewer_several_formats	4	1	4	24	108	2.40e+01	1.20e+02	1.08e+05
mpeg4_mp2_online_web	7	1	2	16	199	5.04e+03	4.03e+04	3.62e+07
mpeg1_big_adapted_online_web	9	1	108	1080	3648	3.62e+05	3.62e+06	3.26e+09
mpeg1_desktop	11	1	54	590	1850	3.99e+07	4.79e+08	4.31e+11
flash_player	11	1	30	360	1013	3.99e+07	4.79e+08	4.31e+11
iphone	11	1	30	360	1017	3.99e+07	4.79e+08	4.31e+11
h264_desktop	11	3	30	360	1400	3.99e+07	4.79e+08	4.31e+11
svc_no_audio_176×144_15fps	13	1	231	3756	14745	6.22e+09	8.71e+08	7.84e+11
mp4_mobile_audio	13	2	297	4158	16826	6.22e+09	8.71e+08	7.84e+11
mpeg2_without_audio	13	5	173	2595	10174	6.22e+09	8.71e+08	7.84e+11
mpeg1_without_audio	13	5	212	3870	13131	6.22e+09	8.71e+08	7.84e+11
mpeg1_with_audio	13	5	281	4131	14937	6.22e+09	8.71e+08	7.84e+11
svc_with_audio_352×288_15fps	16	1	321	5457	21740	2.09e+11	3.55e+12	3.20e+17
mpeg2_medium_desktop	19	3	131	2620	8162	1.21e+17	2.43e+18	2.18e+21
mpeg2_big_desktop	19	3	232	3575	10456	1.21e+17	2.43e+18	2.18e+21
h263_offline	19	3	308	5231	18341	1.21e+17	2.43e+18	2.18e+21
h263_online	19	3	312	6040	19837	1.21e+17	2.43e+18	2.18e+21
mpeg2_small_desktop	20	3	312	6552	20712	2.43e+18	5.10e+19	4.59e+22

Table 2 Number of properties in each *ConversionCapabilities* element with partial description

ConversionCapabilities element	*N*	*ConversionCapabilities element*	*N*
ondemand_mpeg1_http_video_server	18	*visual_format_image_formats_transcoder*	17
ondemand_mpeg4_http_video_server	18	*ondemand_mpeg_transcoder*	25
online_mpeg1_http_video_server	18	*ondemand_mp4_transcoder*	25
online_mpeg2_http_video_server	18	*mpeg2_online_transcoder*	23
online_mpeg4_http_video_server	18	*raw_video_combiner*	21
online_h264_http_video_server	18	*online_resource_loader*	14
big_image_2_video	25	*svc_without_audio_transcoder*	22
medium_image_2_video	25	*svc_with_audio_transcoder*	22
small_image_2_video	25	*svc_to_mp4*	20
mime_image_formats_transcoder	17	*visual_to_svc*	21
image_formats_transcoder	17	*audiovisual_to_svc*	23

Algorithm 6 as well as the number of property comparisons within Algorithm 6. To obtain the theoretical worst-case costs, in Table 1 we used the formulas explained in Sect. 6.2. In order to provide an upper bound for the theoretical worst-case (assuming partial description), we assumed $N = 30$ properties (therefore, $N^2 = 900$) as the larger number of properties in each conversion capabilities element. In addition, Table 1 has a column with the minimum number of steps in the sequence of conversions needed to perform the adaptation.

To study H2 and H3, we performed an ablation study with two different sets of *Content DIs* and *CAT Capabil-*

Table 3 Time and number of comparisons with partial and with total description

ID of the terminal	Type	C	With partial description		With total description	
			Time	Comparisons	Time	Comparisons
gray_images_viewer	I → I	4	173 ms	96	320 ms	245
jpeg_images_viewer	I → I	4	106 ms	94	315 ms	230
images_viewer_without_resolution	I → I	4	108 ms	91	257 ms	201
audiovisual_mobile_1	V → V	4	85 ms	47	112 ms	97
bmp_image_viewer	I → I	4	106 ms	108	220 ms	340
png_image_viewer	I → I	4	177 ms	106	208 ms	340
image_viewer_several_formats	I → I	4	184 ms	108	371 ms	340
mpeg4_mp2_online_web	V → V	7	260 ms	199	487 ms	483
mpeg1_big_adapted_online_web	V → V	9	1327 ms	3648	2012 ms	3834
mpeg1_desktop	V → V	11	1443 ms	1850	3456 ms	4304
flash_player	V → V	11	664 ms	1013	2178 ms	3201
iphone	V → V	11	518 ms	1017	1678 ms	2278
h264_desktop	I → V	11	1134 ms	1400	2976 ms	3023
svc_no_audio_176×144_15fps	SVC	13	7539 ms	14745	28678 ms	48675
mp4_mobile_audio	SVC	13	8791 ms	16826	33457 ms	52567
mpeg2_without_audio	SVC	13	4593 ms	10174	22331 ms	32870
mpeg1_without_audio	SVC	13	6131 ms	13131	25312 ms	35322
mpeg1_with_audio	SVC	13	6012 ms	14937	25240 ms	38731
svc_with_audio_352×288_15fps	SVC	16	7124 ms	21740	36593 ms	66457
mpeg2_medium_desktop	I → V	19	8163 ms	8162	27964 ms	17586
mpeg2_big_desktop	I → V	19	9134 ms	9852	34574 ms	19356
h263_offline	I → V	19	10131 ms	18341	33420 ms	34132
h263_online	I → V	19	11005 ms	19837	38432 ms	39962
mpeg2_small_desktop	I → V	20	10087 ms	20712	59570 ms	86793

ities. In the first set we did not allow absent properties, i.e., all the properties were provided. In the second set the *Content DI* and *CAT Capabilities* only provided some of these properties. Whenever incompleteness semantics (explained in Sect. 4.5.2) apply, they are used and the property was removed from the description. In reference to H2, it is straightforward to demonstrate that the size of the adaptation capabilities description decreases with partial description. This is demonstrated by observing that the number of properties of the *Content DI* and *CAT Capabilities* with total description must be longer. The number of properties with partial description N varies for each *ConversionCapabilities* element. Table 2 shows the number of properties for each of the *ConversionCapabilities* elements with partial description. The number of properties with total description $N_{\max} = 36$ is fixed and determined by the number of properties in the *Properties DI*. In order to check that this number decreases significantly, we have implemented a significance test in Sect. 6.4.

In reference to H3, Table 3 shows the time and number of comparisons needed to execute the experiments with partial and with total description. As CAIN-21 allows specifying the *CAT Capabilities* to be considered during the decision phase (and therefore the *ConversionCapabilities*), the number of *ConversionCapabilities* elements C is not fixed through the tests. Section 6.4 proves that the time and number of comparisons also decrease significantly with partial description.

6.4 Statistical significance of the experiments

This subsection studies the statistical significance for the reduction in the average-case computational cost (H1), size of the partial description (H2), and time needed to compute the decision with partial description (H3).

The average-case number of properties to be compared in each experiment is a statistical variable that depends on the number of *ConversionCapabilities* elements C involved in the experiment. If we increase C for a given experiment, the number of properties to be compared would also increase. Therefore, in this study we assume a normal distribution in this statistical variable only when the adaptation tests have the same C number. Formally, the number of comparisons in the theoretical-worst case is not a statistical variable but a fixed upper bound for the given N and C. However, for the purposes of the following significance tests it can be seen as a statistical variable for which, given N and C, its mean is the fixed upper bound and its variance is zero.

The other statistical variables in this study, (i.e., the average-case number of invocations of Algorithm 2, the average-case number of invocations of Algorithm 6, the time needed to execute Algorithm 2 with partial description and with total description) can be seen as different views of how the average-case number of comparisons varies with the number of available properties. To demonstrate that these statistical variables are aligned with the average-case number of property comparisons, we have calculated their correlation. The correlation coefficient between the number of invocations of Algorithm 2 and the number of property comparisons is 0.987. The correlation coefficient between the number of invocations of Algorithm 6 and the number of property comparisons is 0.990. The correlation coefficient between the time needed to compute the plan with partial description and the number of property comparisons is 0.931. The correlation coefficient between the time needed to compute the plan with total description and the number of property comparisons is 0.945. For these reasons, when the adaptation tests have the same C number, we also assume a normal distribution on these statistical variables.

On the other hand, as the samples come from different adaptation tests, we assume independence between the samples of the independent variables to be compared in the significance tests. Specifically, among the samples of the average-case computational cost and the samples of the theoretical worst-case computational cost (H1), among the samples of the size of the adaptation capabilities with partial and with total description (H2), and among the samples of the decision time with partial and with total description (H3).

In addition, as we have a relative small number of samples, we are going to validate these hypotheses by testing the difference between the two independent variables means using the Student's t-test. In this significance test, the t-score (t) is calculated as:

$$t = \frac{m_1 - m_2}{SE} \tag{1}$$

Where m_1 is the mean of the first independent variable, m_2 is the mean of the second independent variable, and SE is the *standard error*, which is calculated as:

$$SE = \sqrt{\frac{v_1}{n_1} + \frac{v_2}{n_2}} \tag{2}$$

In formula (2), n_1, v_1 are the number of tests and variance in the first independent variable, and n_2, m_2 are the number of tests and variation of the second independent variable.

For the three tests, we define two alternative hypotheses: the null hypothesis is that $m_1 \geq m_2$ and the alternative hypothesis is that $m_1 < m_2$. The critical value of t (t_c) depends on the significance level (which is always 0.05 in this study) and on the *degree of freedom* (DF) of the adaptation tests.

6.4.1 Computational cost

To study the reduction in the computational cost for H1, we have divided the tests into four groups with the same C number in Table 1. In this way, as has been discussed above, we can assume that the independent variables (i.e., the average-case and worst-case number of invocations) in each *group of tests* follow normal distributions. Specifically, we have selected four groups of tests corresponding to $C = 4$, $C = 11$, $C = 13$, $C = 19$. The other tests (i.e., tests with $C = 7$, $C = 9$, $C = 16$, $C = 20$) were discarded because there is only one test per group and therefore these tests have zero *degrees of freedom* (*DF*). For the selected four groups, the means to be compared are the average-case computational cost m_1 and the worst-case computational cost m_2. To analyze the number of invocations of Algorithm 2, Table 4 collects the means, variances, standard error, degrees of freedom used to calculate the t-critical value, t-score and t critical value for the independent variables in each group. Likewise, Tables 5 and 6 show the same information for, respectively, the number of invocations of Algorithm 6 and for the number of property comparisons. The worst-case computational cost is the second independent variable. As the second independent variable is a theoretical upper value, its mean m_2 is constant for each group and therefore $v_2 = 0$.

In all cases, the null hypothesis is rejected because the t-score is lower than the t critical value (i.e. the t-score is in the region of rejection). Therefore, we conclude that the

Table 4 Significance test for Algorithm 2

Group	$n_1 = n_2$	m_1	m_2	v_1	v_2	SE	DF	t	t_c
$C = 4$	7	3.71	2.4e+01	0.57	0.00	0.28	6	−7.10e+01	−1.94
$C = 11$	4	36.00	3.99e+07	144.00	0.00	6.00	3	−6.65e+06	−2.35
$C = 13$	5	238.80	6.23e+09	2569.20	0.00	22.66	4	−2.74e+08	−2.13
$C = 19$	4	245.75	1.21e+17	7206.91	0.00	42.44	3	−2.85e+15	−2.35

Table 5 Significance test for Algorithm 6

Group	$n_1 = n_2$	m_1	m_2	v_1	v_2	SE	DF	t	t_c
$C = 4$	7	20.28	1.20e+02	24.57	0.00	1.87	6	−5.32e+01	−1.94
$C = 11$	4	417.50	4.79e+08	13225.00	0.00	57.50	3	−8.33e+06	−2.35
$C = 13$	5	3702.00	8.71e+08	412141.50	0.00	287.10	4	−3.03e+06	−2.13
$C = 19$	4	4366.50	2.43e+18	2408232.33	0.00	775.92	3	−3.13e+15	−2.35

Table 6 Significance test for the number of comparisons between properties

Group	$n_1 = n_2$	m_1	m_2	v_1	v_2	SE	DF	t	t_c
$C = 4$	7	92.85	1.08e+05	4.58e+02	0.00	8.09	6	−1.33e+04	−1.94
$C = 11$	4	1320.00	4.31e+11	1.57e+05	0.00	198.61	3	−2.17e+09	−2.35
$C = 13$	5	13962.60	7.84e+11	6.20e+06	0.00	1113.68	4	−7.03e+08	−2.13
$C = 19$	4	14199.00	2.18e+21	3.31e+07	0.00	2878.05	3	−7.57e+17	−2.35

Table 7 Significance test for H2

Group	$n_1 = n_2$	m_1	m_2	v_1	v_2	SE	DF	t	t_c
All	22	20.45	36.00	11.21	0.00	0.71	21	−21.77	−1.72

Table 8 Significance test for the decision time with partial and with total description

Group	$n_1 = n_2$	m_1	m_2	v_1	v_2	SE	DF	t	t_c
$C = 4$	7	134.14	2.57e+02	1.75e+03	75.20e+02	36.39	6	−3.39	−1.94
$C = 11$	4	939.75	2.57e+03	1.81e+05	6.33e+05	451.29	3	−3.61	−2.35
$C = 13$	5	6613.20	2.70e+04	2.56e+06	1.80e+07	2031.37	4	−10.04	−2.13
$C = 19$	4	9608.25	3.36e+04	1.51e+06	1.86e+07	2247.80	3	−10.67	−2.35

average-case computational cost of the *Planner* is significantly lower than the theoretical worst-case computational cost.

6.4.2 *Size of the descriptions*

To study H2, the first independent variable is the number of properties in the *ConversionCapabilities* elements with partial description N (gathered in Table 2) and the second independent variable is the number of properties in the *ConversionCapabilities* elements with total description (assuming $N_{\max} = 36$ as explained in Sect. 6.3). We compare the means of the first independent variable m_1, with the means of the second independent variable m_2. In this significance test, we assume that the number of properties in the *ConversionCapabilities* elements follows a normal distribution and thus Table 7 has only one group of tests. The number of properties with total description m_2 is constant ($m_2 = N_{\max} = 36$) and therefore $v_2 = 0$.

The null hypothesis is rejected because the t-score is lower than the t critical value. Thus, we conclude that the size of the *ConversionCapabilities* elements decreases significantly when partial description is allowed.

6.4.3 *Decision time*

This subsection studies the significant reduction in the decision time that H3 hypothesizes. Table 3 gathers the decision time for the tests with partial and with total description. Again, to assume a normal distribution for the independent variables of each group of tests we have created four groups of tests corresponding to $C = 4$, $C = 11$, $C = 13$, $C = 19$. Table 8 shows the result of computing the t-score for the decision time with partial and with total description. In contrast to the previous experiments, in this experiment the second independent variable is not theoretical, and thus $v_2 > 0$. It can be observed that in all cases, the t critical value is lower than the t-score, and therefore we conclude that the decision time decreases significantly when partial description is allowed.

Even though H3 only states that the decision time decreases significantly, in Table 9 we have analyzed the num-

Table 9 Significance test for the number of comparisons with partial and with total description

Group	$n_1 = n_2$	m_1	m_2	v_1	v_2	SE	DF	t	t_c
$C = 4$	7	92.85	2.56e+02	4.58e+02	8.37e+03	35.52	6	−4.59	−1.94
$C = 11$	4	1320.00	3.20e+03	1.57e+05	7.00e+05	463.10	3	−4.06	−2.35
$C = 13$	5	13962.60	4.16e+04	6.20e+06	7.35e+07	3993.67	4	−6.92	−2.13
$C = 19$	4	14048.00	2.78e+04	3.47e+07	1.21e+08	6243.81	3	−2.19	−2.35

ber of comparisons to prove that it also decreases significantly. In this way, we aim to demonstrate that comparing properties is the upper bound and more costly operation in our *Planner*. To provide more evidence for the alignment of the decision time and the number of comparisons, we have also computed the correlation coefficient between the time and number of comparisons in Table 3, which are 0.944 with partial description and 0.901 with total description. The intuition behind this statement is that comparing properties is the inner operation and therefore, the operation that is repeated more often.

6.5 Increase in the range of addressable problems with internal decisions

To demonstrate Claim 4, the CAIN-21 demo implements two CATs that include internal decisions. We assume that it is very difficult (or impossible) to make these internal decisions during the planning phase. Subsequently, we demonstrate that if we postpone these decisions to the execution phase, these decisions can be performed inside the CAT.

The first CAT with internal decisions is called *Image2Video*. This CAT uses *Regions Of Interest* (ROIs) descriptions stored in the *Content DI* so that additional decisions can be performed. Specifically, the ROIs are used (see Fig. 7) to focus on the faces in the photo instead of the whole subjects in the image. This process is further explained in previous work [25].

The second CAT with internal decision is the SVCCAT. This CAT performs scalable video coding adaptation. A scalable video is divided into one or more layer. The *base layer* contains a rough representation of the video suitable for low bandwidth networks. Additional layers, called *enhancement layers*, provide additional details in several dimensions (typically spatial dimension, frame-rate dimension and quality dimension). If the bandwidth supports the delivery of more information, the SVCCAT implements a mechanism to decide which enhancement layer to transmit. Specifically, the SVCCAT uses *Peak Signal to Noise Ratio* (PSNR) metrics stored in the *AdaptationQoS* description (explained in Sect. 2.2) to make this decision.

The remainder of this subsection describes in detail three internal decision tests involving SVC and three internal decision tests involving image to video adaptation. The first SVC-adaptation to discuss is labeled *svc_no_audio_176 ×*

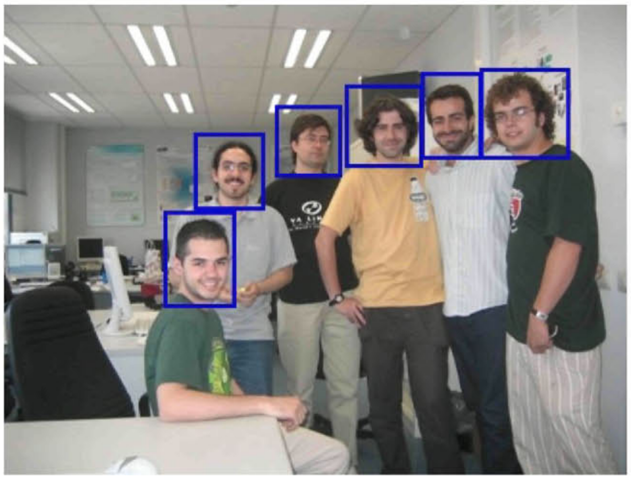


Fig. 7 Example of image resource to be adapted

144_15fps in Table 3. With this test, the *Planner* algorithm produces the sequence of conversions *initial* → *svc_without_audio_transcoder* → *goal* with. In this sequence, the *initial* conversion state corresponds to the *Content DI* to be adapted, the *svc_without_audio_transcoder* conversion state corresponds to one of the conversion modules implemented in the SVCCAT and *goal* corresponds to the properties of the terminal labeled *svc_no_audio_176 × 144_15fps*. The *svc_without_audio_transcoder* conversion state is executed with the following source and target parameters (for better readability, only the relevant parameters from the standpoint of this discussion are shown):[6]

Source parameters:
visual_frame = {176 × 144, 288 × 352, 704 × 576}
visual_frame_rate = {1.875, 3.75, 7.5, 15, 30}
visual_bitrate = [500..100000]

Target parameters
visual_frame = {176 × 144}
visual_frame_rate = {15}
visual_bitrate = [0..200000]

The *visual_frame* source parameter indicates that the scalable input video can be adapted at three different frame

[6]The complete details of these tests can be obtained in the public demo publicly available at http://cain21.sourceforge.net.

resolutions ($176 \times 144, 288 \times 352, 704 \times 576$) and the target parameter indicates that the *Planner* algorithm has selected the frame resolution 176×144. This happens because 176×144 is the frame size of the current terminal (labeled *svc_no_audio_176* × *144_15fps*). Similarly, the *Planner* algorithm has selected 15 fps, which corresponds to the frame rate of the terminal. Regarding the visual bitrate target parameter, the *Planner* algorithm does not make a complete decision for this value. The terminal supports a visual bitrate from 0 to 200000 bps, which is the *Planner* selected range. As the *Planner* has not solved the whole problem, this decision is transferred to the internal decision. Specifically, during the execution phase the SVC-CAT (using the *AdaptationQoS* description) decides that the maximum quality layer that can be delivered has a bitrate of *visual_bitrate* = 98000 fps. The second test labeled *svc_with_audio_352* × *288_15fps* in Table 3 produced similar results, but including the audio stream.

The third test labeled *mp4_mobile_audio* in Table 3 is another didactic experiment. In this test, the *Planner* produces the sequence of conversions *initial* → *svc_without_audio_transcoder* → *svc_to_mp4* → *goal*. Note that the *Planner* adds the *svc_to_mp4* conversion state at the end of the sequence to transcode SVC content to MP4. This occurs because the *Content DI* to be adapted contains SVC video, but the target terminal is not a SVC-compliant terminal. In the *svc_without_audio_transcoder* conversion state, the selected source and target parameters are:

Source parameters:
visual_frame = $\{176 \times 144, 288 \times 352, 704 \times 576\}$
visual_frame_rate = $\{1.875, 3.75, 7.5, 15, 30\}$
visual_bitrate = [500..100000]

Target parameters
visual_frame = $\{176 \times 144, 288 \times 352, 704 \times 576\}$
visual_frame_rate = $\{1.875, 3.75, 7.5, 15, 30\}$
visual_bitrate = [0..200000]

Again, the *Planner* does not completely specify the target parameters for this conversion state and therefore offloads the decision to the SVCCAT (which implement the *svc_without_audio_transcoder* conversion module). The SVCCAT, then uses the *AdaptationQoS* description to make a decision, which selects a layer that fulfils the constraint *visual_bitrate*< 200000 maximizing the quality of the adaptation. During this internal decision, the SVCCAT selects the layer with properties *visual_frame* = 288×352, *visual_frame_rate* = 15 and visual_bitrate = 185000.

In the I→V tests in Table 3, the *Planner* always selects the sequence of conversions *initial* → *image_formats_transcoder* → *medium_image_2_video* → *ondemand_video_transcoder* → *goal*. The *image_formats_transcoder* conversion state transcodes the image to the format and size that the *medium_image_2_video* conversion state requests in its preconditions (JPEG image format and 3:4 aspect ratio). The *medium_image_2_video* conversion module is only able to produce MPEG-2 video and the *ondemand_video_transcoder* conversion state transcodes this video to the (format and size) constraints of the terminal. Note that even though the sequence of conversions is always the same, the parameters of the conversion states were different for each test. This happens because each test was executed with a different *Content DI*. Specifically, in the first test the *Content DI* (named *birds_eye_view_di.xml*) had no ROIs descriptions. In the other two cases that we have studied, the *Content DI*s were annotated with ROIs. The ROIs were provided to the *Image2VideoCAT* in the property labeled *roi*. Specifically, during the second test the *Content DI* (named *group_roi_di.xml*) was adapted with *roi* = $\{62$ 139 76 156, 93 151 107 165, 115 128 128 145, 129 177 143 $193, \ldots, 403$ 146 415 $16\}$. During the third test, the *Content DI* named *people_roi_di.xml* was adapted with *roi* = $\{95$ 125 116 149, 110 76 127 95, 151 56 169 84, 189 57 206 76, 219 53 234 71, 251 51 266 $69\}$. The image resource used in the third test is shown in Fig. 7.

6.6 Analysis of the results

H1 states that in practical scenarios the average-case computational cost of the *Planner* algorithm is significantly lower than the theoretical worst-case computational cost. Section 6.4.1 has confirmed the significant difference that H1 proposes. The intuition behind these results is that Algorithm 2 only succeeds in expanding a few conversion states (the feasible conversion states according to Algorithm 4). Besides, in practice, most of the properties are optional and rarely used. To empirically see it, we can observe that when C increases, the ratio between the theoretical worst-case number of invocation of the algorithms and the real number of invocations also increases. The correlation coefficient between C and this ratio is 0.704 for Algorithm 6 and 0.666 for Algorithm 2. This can also be observed in Tables 4, 5 and 6 in which when C increases the t-score increases as well. This suggests that H1 becomes more important with large values of C. The reason behind this observation is that with a large number of conversion capabilities elements, Algorithm 4 eliminates a larger number of expansions in the virtual tree of conversions.

H2 states that a partial description of the conversion modules has the benefits of reducing the size of the adaptation capabilities and H3 states that partial description also significantly reduces the decision time. To study H2 and H3 we have performed an ablation study. The first group uses a partial description of the *ConversionCapabilities* and the second group uses a total description of the *ConversionCapabilities*. To prove H2, Table 2 collects from the CAIN-21 software the number of properties for each *ConversionCapabilities* description element with partial description. It is obvious that a partial description is smaller than a total description. Section 6.4.2 validates H2 because the size of the conversions decreases significantly. To prove H3, Table 3 collects two groups of similar tests that only differ in the *Content DI* and *CAT Capabilities* description. Table 3 shows both the time and number of comparisons performed. Section 6.4.3 demonstrates that in all cases the time and number of comparisons decrease significantly when partial description was allowed. These results validate H3 and show the usefulness of partial description.

Claim 4 states that offloading decision-making from the *Planner* to the conversion modules the range of problems that can be addressed increases. The intuition behind Claim 4 is that certain decisions cannot be made during the decision phase, but can be made during the execution phase. To demonstrate Claim 4, Sect. 6.5 collects 3 tests involving internal decisions for SVC layer selection and 3 tests involving internal decisions for ROIs-based adaptation. For these tests, there are no methods for adaptation decision-making during the planning phase. However, Sect. 6.5 shows how the range of problems that can be addressed increases by making these decisions during the execution phase. The divide and conquer design paradigm recommends to divide complex solutions into smaller and more specific solutions. In CAIN-21, the *Planner* makes the general decisions that only rely on metadata (properties) and transfers the particular problem domain decisions to the CATs.

6.7 Comparison

In [2], the authors report experiments in which the planner selects and performs multi-step image adaptation. However, they do not provide a dataset, but instead report the times needed to construct the Directed Acyclic Graph (DAG), which determines the multi-step sequence. The behavior of the *Planner* algorithm in this paper can be compared with the behavior of the planning algorithm in [2]. The *Planner* builds a virtual tree of conversions whereas [2] has reported the construction of a DAG that is similar. The main difference is that in the virtual tree of conversions (as explained in Sect. 4.2) the *Planner* creates several instances of the conversion state representing the initial *Component*.

In reference to the computational cost, the main difference that we have identified between our *Planner* and the planner in [2] is that in our implementation, the computation time substantially depends on the number of conversion capabilities elements C involved in the decision. Conversely, in the results from [2], the computation time only depends on the length of the sequence obtained and does not depend on the number of conversion capabilities (services in their terminology). Using the data in Table 3, Fig. 8 compares the number of conversion capabilities evaluated C against the time needed to compute the plan with partial and with total description. In the *Planner*, both seem to increase rapidly, although the decision time with total description is significantly higher. In our understanding, the number of nodes that have to be analyzed in each node expansion must influence the decision time and this is the reason why in our work the computational time increases when C increases.

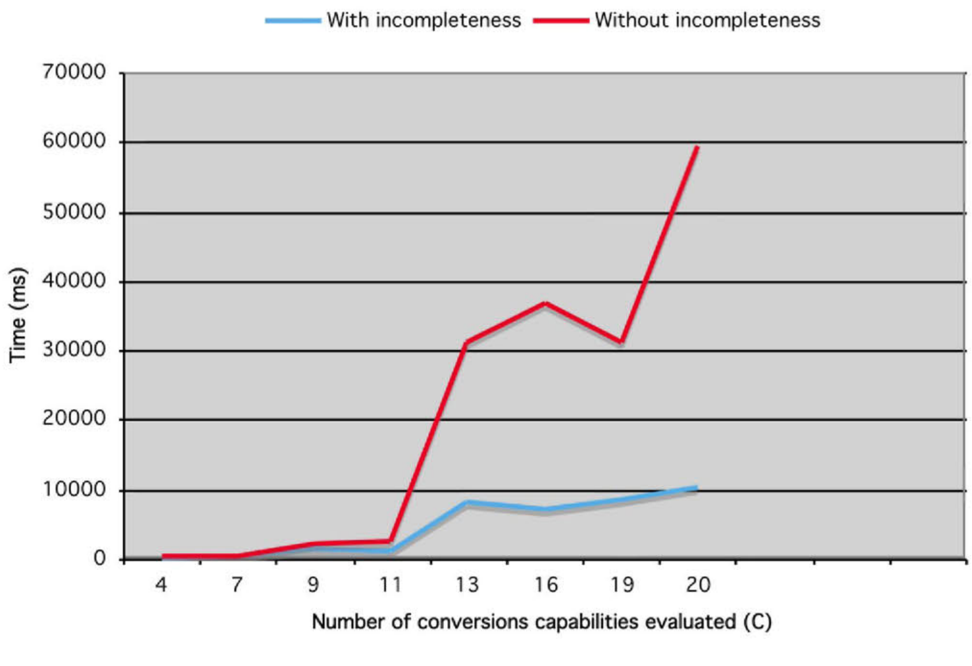


Fig. 8 Plan construction time with partial and with total description

7 Summary of contributions, limitations and future work

7.1 Contributions

Initial studies in multi-step multimedia content adaptation have employed forward chaining [1] in conjunction with breadth-first search and goal regression (and some heuristics) to search for a sequence of actions that lead from the initial state to a goal state [2, 3, 7]. As explained in Sect. 2.3.2, further advances in AI [20, 21] combined planning graphs with backwards chaining techniques to search, beginning from a goal state and finishing in an initial state. This paper has applied these ideas to multi-step multimedia content adaptation. The algorithms in Sect. 5 combine ideas from both modi operandi (graph-based planning and backwards searching) and introduce new ideas, which are not present in the neoclassical backwards-chaining algorithms. In particular, our paper contributes the following:

(1) *Sound multimedia adaptation planning.* This research has proven that AI planning techniques can make sound automatic decisions in the multimedia domain. The *Planner* algorithm is sound and produces a finite and complete plan. These features, in general, however, do not hold when a planning algorithm permits the removal of effects. For these reasons, Sect. 5.7 has presented an algorithm that never removes effects.

(2) *Multimedia properties.* The use of multimedia properties presents several advantages with respect to using standard XML representation: (a) all the decision-related information can efficiently be held in memory; (b) changes in the underlying XML files do not imply changes in the source code of the whole adaptation engine (Sect. 3.2 explained that it only implies changes in the *Properties DI* document); (c) the multimedia properties directly represent the information that the *Planner* requires; (d) this information is represented homogeneously; and lastly, (e) alternatives are easily represented by means of multi-valued properties.

(3) *Bounded non-deterministic planning.* The *Planner* is well suited to represent parameterized conversion states and conversion capabilities of the conversion modules. We have replaced the notion of action with the notion of conversion in such a way that different parameters lead to different actions. Multi-valued parameters make it possible to gather related actions in a single conversion state. The source parameters (that may be multi-valued representing the selected inputs) represent the input to the conversion. The target parameters (that also may be multi-valued allowing the CAT to make internal decisions) control the output of the conversion. One conversion can be comparable to a set of actions in a Graphplan-like planner. Section 4.3.2 explains peculiarities of conversions such as using selected conversion states to determine the source and target parameters. The partial description is another characteristic of the conversion modules.

(4) *Matching process.* As a side effect, the matching process developed in Sect. 5.5 represents an easy and more efficient way to check the relations between the MPEG-7 and MPEG-21 descriptions of the *Component* states (both selected and realized) and also of the conversion states that modify the *Component.* To check these relations, the classical and neoclassical planning approach—of modeling preconditions and effects with first order logic predicates—has been replaced by properties. The matching process described in Sect. 5.5 has replaced the state transition function that performs unifications between predicates. Instead of computing intertwined states and actions, this replacement is the basis for the computation of conversion states (see Fig. 6) and uses the parameters to impliedly represent the state of the *Component* being adapted. Additionally, traditional bound or unbound planning values have been replaced by general multi-valued properties, where a property is always bound to one or more values.

(5) *Alternatives.* We have developed a model that allows representing alternative constraints in the terminal (e.g., the terminal accepts several media formats). Additionally, this model allows for representing alternatives in the internal decisions of the CATs.

(6) *Partial decisions.* The *Planner* partially decides the adaptation constraints that the conversion modules must comply with and postpones other decisions to the conversion modules. These conversion modules can use dissimilar decision techniques referred to as internal decisions. For instance, the *Image2Video* CAT can make decisions (using the ROIs [25] stored in the *Content DI*) to show the faces in the image being adapted, instead of the whole subjects in the image.

(7) *Internal decisions.* Most planners make all the decisions during the decision phase. The *Planner* transfers parts of the decision to the CATs and in this way, the decision and the execution phases are intertwined. In contrast to continuous planning (introduced in Sect. 2.3.3), the conversions of the *Planner* are bounded non-deterministic actions. As a result, the *Planner* does not perform further decisions that depend on the result of the internal decisions transferred to the CAT. That is, the *Planner* computes all the sequences of conversions before the *Executer* can start executing the CATs. The notion of internal decisions cannot be found in existing multi-step multimedia adaptation planning algorithms [2, 3, 7].

(8) *Parameters for optimization.* Previous multi-step multimedia adaptation planning algorithms seldom search for more than one sequence of actions. Conversely, the bounded non-deterministic planner developed in this work searches for all the sequences of conversions capable of adapting the content to the usage environment. This feature allows further decisions in order to pick the sequence that optimizes some criterion (such as execution time or resulting screen resolution). In addition, in contrast to a neoclassical planner, the bounded non-deterministic planner must find the source and target parameters that must be supplied to the non-deterministic conversions.

(9) *Tolerating partial description.* Traditional decision systems only rely on actions whose adaptation capabilities are completely known. Planning under uncertainty uses belief states (explained in Sect. 2.3.3) that associate a probability distribution over the state space in order to represent the manifold states that non-deterministic actions can produce. This work has proposed the use of selected and realized states (instead of belief states) to address non-deterministic conversions.

(10) *Absent properties.* The *Planner* deals with absent properties, which are useful in practical applications. Furthermore, the absent properties, in conjunction with multi-valued properties allow the *Planner* to navigate through a set of conversion states. These are only partially determined using the same technique as the neoclassical planners. The work in [20] demonstrated that these sets reduce the number of states that must be evaluated and therefore speed up the decision process. Previous multimedia adaptation planners do not take advantage of this idea. Absent properties must not be confused with un-instantiated action and goal attributes within a classical planner: the former corresponds to information that is never given, while the latter are unbound attributes that must be bound after producing a plan. Absent properties must also not be confused with flexible planning [21]. Absent properties correspond to lack of information; flexible planning introduces soft constraints in the classical planning domain definition.

7.2 Limitations

Two main limitations have been identified in the multimedia adaptation decision system proposed in this research. First, the *Planner* considers the terminal properties mandatory constraints, i.e., the sequence of conversions must produce all of them. In contrast to the *Content DI* and *CAT Capabilities,* the properties of the terminal cannot be ignored/optional. More precisely, the terminal properties are a conjunction of preconditions where all the preconditions must be "produced" at a certain step of the sequence of conversions. For instance, if a terminal accepts visual and audio streams, the *Planner* would consider non-consumable by this terminal a video composed of just a visual stream (without an audio stream). Optional properties cannot be included either in the standard MPEG-21 *UED* or in the *Planner* developed in this paper. In addition, the semantics of MPEG-21 *UED* tools do not provide a mechanism to prohibit properties in the media to be consumed. According to these semantics, if the MPEG-21 terminal does not declare the audio stream format, it does not mean that the media cannot include audio; rather it means that the *Component* will be accepted independently of the existence of an audio stream.

Second, the matching process described in Sect. 5.5 makes possible to define static relations between conversion states (e.g., determining that cs_i produces a video format that cs_{i-1} accepts). Dynamic relations between preconditions and postconditions (i.e., properties whose values depend on other properties) cannot be expressed in this declarative approach. For instance, the output bitrate of a video transcoder depends, in general, on the input frame size. Although it is theoretically possible to express a property as a function of other properties, it is not always easy or possible to find a function that provides the exact value of a property as a function of other properties (e.g., the exact output bitrate might not be represented as a function of the input frame width and height properties). As a result, the output properties (the postconditions) can be related to the input properties using ranges of values more easily. Whenever the CAT implementers have to express this relation, it is easy to create several conversion capabilities with different "profiles" for the input and output property ranges. For instance, one *ConversionCapabilities* element might describe that frame sizes between 44×36 and 177×144 produce a bitrate between 2000 bits/s and 8000 bits/s. Another *ConversionCapabilities* element might describe that the frame sizes between 177×144 and 704×576 produce a bitrate between 8000 bits/s and 48000 bits/s. For instance, due to this limitation, the *Image2VideoCAT* has three *ConversionCapabilities* elements (namely, the *big_image_2_video*, *medium_image_2_video* and *small_image_2_video* in Table 2).

Besides the above two limitations, the following difficulties in the implementation or use of the adaptation engine have been identified:

(1) The input *Content DI*, where the *Component* to be adapted is located, cannot impose constraints involving properties that the CATs or the usage environment does not consider. That is, those properties that appear in the preconditions of the conversion states must be fulfilled by the *Component*. For example, if a conversion state

requests *audio_stream* = {*mp*2, *mp*3, *aac*}, the *Component* must contain one of these audio streams in order to enable the evaluation of the conversion state. Otherwise, the *Planner* will not evaluate this conversion state. Conversely, it can be observed that properties of the *Component* that do not appear in the conversion state preconditions are always allowed and ignored and these properties impliedly become postconditions of the conversion state. The cause of this effect is the *preserved properties* semantic defined in Sect. 4.5.2, which tolerates the existence of partial description in the *CAT Capabilities* and *UED*. Therefore, the terminal may end up receiving an adapted *Component* with properties that the terminal does not understand. To avoid such problems, the terminals must not use properties that have not been declared in its *UED*.

(2) There is a gap between the MPEG-7 description of the *Component* and the MPEG-21 *UED*. These difficulties have been hidden behind the *getUEDConversionState*() function in Algorithm 1. An example of this gap is a mismatch between the MPEG-7 description of the color domain that can take the values *color*, *colorized*, *graylevel*, *binary* and the MPEG-21 description of the terminal color display capabilities, which can take the values *true*, *false*. To address this problem, the *getUEDConversionState*() function transforms the MPEG-21 properties associated with the *UED* into MPEG-7 properties.

(3) It is not always possible to describe the conversions that can be performed if the capabilities of a CAT are not represented with several *ConversionCapabilities* elements. Therefore, this increases the verbosity of the *CAT Capabilities*. Specifically, it is cumbersome for a conversion capabilities element to express that it preserves the value of the property, but the property must exist and take (at least) one value from a set of selected values. In this scenario, the *ConversionCapabilities* element must be divided into several *ConversionCapabilities* elements, so that the preconditions of each *ConversionCapabilities* element accept only one value and produce the same value. Another problematic situation arises whenever a *ConversionCapabilities* element describes several preconditions, as it represents an implied conjunction of preconditions. To express disjunction of preconditions, these must be listed in different *ConversionCapabilities* elements so that all the preconditions and postconditions describe conjunctive conditions. Still another situation that is difficult to express in a *ConversionCapabilities* element is that it does not accept input parameters with certain combinations of properties; for example, a *ConversionCapabilities* element that accepts JPEG and PNG images, where JPEG images are accepted in color and grayscale and PNG images are only accepted in grayscale. Once again, in this situation the capabilities must be split into two separate *ConversionCapabilities* elements: one stating that JPEG images are accepted in both color and grayscale and another in which PNG images are accepted only in grayscale. If the *CAT Capabilities* and the enclosing *ConversionCapabilities* element have several of these restrictions, the list of *ConversionCapabilities* elements will quickly become long and unwieldy. This difficulty could be handled through a Graphic User Interface (GUI) managing these descriptions. In addition, the partial description semantics developed in Sect. 4.5 also help.

(4) The correct operation of the system depends on the precise conformance with the semantics of the *CAT Capabilities* that represent the preconditions and postconditions of the conversion capabilities elements. This means that the whole system usefulness depends on judicious and correct "advertisement" of the *ConversionCapabilities* by the CAT authors, i.e., according to the semantics of the parameters. If different CATs describe the capabilities in different ways or have different policies with respect to how properties should be specified, the system would not work effectively. For instance, consider MPEG-4 videos that have different levels and profiles. One CAT might be capable of processing all the MPEG-4 video file levels and never specifies the levels in its capabilities description. Another CAT might only be capable of processing one level and advertise that it can accept only this level. In this case, the two CATs might not work together effectively on MPEG-4 media, i.e., the output of one CAT cannot be used as input of the other CAT. This may happen if, due to incompleteness semantics, the output of the first CAT does not specify the level and its output level cannot be used as input by the second CAT. To answer this problem, the CAT implementer must pay special attention to describing the preconditions and postconditions of the conversion capabilities elements according to their semantics. Making use of a set of standardized classification schemes is very useful in this case. MPEG-7 Part-5 classification schemes such as the *ContentCS*, *FileFormatCS*, *VisualCodingFormatCS* and *AudioCodingFormatCS* (see Annex B of the MPEG-7 Part 5 standard [18]) can be used in this case.

7.3 Future work

There are different opportunities for future work. First of all, it is important to consider the details of how to selects a sequence of conversions after the *Planner* has produced a set of sequences of conversions **SSOC** in which more than one sequence satisfies the adaptation requirements. How does the *Planner* decide which sequence of conversions to make? How can the cost of each sequence be measured in order to select the "cheaper" sequence? Which criterion is more feasible, lower memory usage or lower computing time? Can hybrid models be applied? Perhaps the simplest approach is to take the shortest sequence. Instead of a shorter sequence, perhaps it is better to choose online adaptations first? That is, sequences whose conversion modules are annotated in the *CAT Capabilities* as online conversions (i.e., conversion modules that start serving the media before it has been totally adapted) could be preferred over sequences that include offline conversion modules. Also, the computational costs may be annotated in the *ConversionCapabilities*. Dynamic measures of the past performance of each CAT could be considered as well. Other approaches might be the use of memory and, depending on the terminal, specific properties like battery lifetime could be evaluated. The user preferences can be considered in this case to answer all these questions.

A second opportunity for further development is related to the conversion states. If there are several sequences that contain the same conversion states, but in a different order, what order is the most appropriate? Here it should be evaluated whether different orders provide the same outcome (the same outcome may mean the same adaptation quality) and/or whether different orders yield "cheaper" sequences. For example, if the last conversion state is lossy and the remaining ones are lossless, then other sequences of conversion states may be cheaper than using the lossy conversion state after all the lossless conversion states.

A third research area lies in the improvement of the performance of the reachability analysis (see Sect. 2.3.2). As Fig. 2 shows, our *Planner* builds a reachability tree that has been proven finite and complete in Sect. 5.7. Graphplan introduced reachability ideas to reduce the computational spatial costs by avoiding the expansion of duplicate states.

A fourth research direction is dealing with flexible planning [21]. Having a set of decoding and transmission constraints, how should the *Planner* maximize the number of desirable constraints satisfied? One way is to maximize the number of desirable constraints (each desirable constraint has the same weight), but another option is to establish a ranking of constraints (each constraint may have a different weight).

8 Conclusion

This paper has dealt with the applicability of AI planning methods for the computation of multi-step multimedia adaptations. To adapt multimedia, some extensions to standard AI planning methods have been proposed. Traditionally, multi-step adaptation has been implemented with an AI planner that makes all the decisions before beginning the adaptation. Taking into account that there are decisions that can only be made, or are easy to made, during the execution phase (i.e. when the media resource is available), this paper has proposed the inclusion of these decisions into the adaptation process. To accomplish this goal, we have modeled multimedia conversions as bounded non-deterministic conversions and we have developed a bounded non-deterministic planning algorithm. The *Planner* allows for dealing with decision-making problems in which the conversions to be performed can be controlled (i.e., are bound), even though under some circumstances they may produce different outcomes (i.e., are non-deterministic). These outcomes may also only be partially observable by the *Planner*. Overall, the proposed planning algorithm is capable of computing all sequences of conversions that adapt an MPEG-21 *Component* to the constraints of the terminal. Deciding which of these sequences of conversions is the best with respect to some plan quality metrics is part of our future work. In addition, mechanisms that deal with partial observability tolerating partial description have been proposed. The theoretical analysis has proven the soundness of the *Planner* and its applicability has been proven as well. Finally, the most important findings, limitations and difficulties pertaining to multi-step multimedia adaptation have been discussed.

References

1. Russell S, Stuart J (2003) Artificial Intelligence: a modern approach, 2nd edn. Prentice Hall, New York
2. Girma B, Brunie L, Pierson JM (2006) Planning-based multimedia adaptation services composition for pervasive computing. In: Proceedings of 2nd international conference on signal-image technology internet based systems (SITIS'2006), pp 132–143
3. Soetens P, De Geyter M (2005) Applying domain knowledge to multistep media adaptation based on semantic web services. In: Proceedings of workshop on image analysis for multimedia interactive systems (WIAMIS 05) (CD-ROM Proc), 4 pp
4. Burnett IS, Pereira F, de Walle RV, Koenen R (eds) (2006) The MPEG-21 Book. Wiley, New York
5. Mukherjee D, Delfosse E, Kim JG, Wang Y (2005) Optimal adaptation decision-taking for terminal and network quality-of-service. IEEE Trans Multimedia 7(3):454–462
6. Sofokleous AA, Angelides MC (2008) DCAF: an MPEG-21 dynamic content adaptation framework. Multimed Tools Appl 40(2):151–182
7. Jannach D, Leopold K, Timmerer Ch, Hellwagner H (2006) A knowledge-based framework for multimedia adaptation. Int J Appl Intell 24(2):109–125
8. López F, Jannach D, Martínez JM, Timmerer Ch, Hellwagner H, García N (2008) Multimedia adaptation decisions modelled as non-deterministic operations. In: Proceedings of 9th international workshop on image analysis for multimedia interactive services WIAMIS 2008, May 2008, pp 46–49

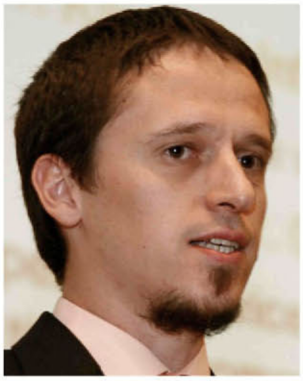
9. Martínez JM, Valdés V, Bescos L, Herranz J (2005) Introducing CAIN: a metadata-driven content adaptation manager integrating heterogeneous content adaptation tools. In: Proceedings of WIAMIS'05 (CD-ROM Proc), 4 pp
10. Lopez F, Martínez JM, García N (2009) CAIN-21: an extensible and metadata-driven multimedia adaptation engine in the MPEG-21 framework. In: 4th international conference on semantic and digital media technologies (SAMT 2009), Graz, Austria, 2–4 December. Lectures Notes in Computer Science, vol 5887. Springer, Berlin, pp 114–125
11. López F, Martinez JM (2007) Multimedia content adaptation modelled as a constraints matching problem with optimisation. In: Proceedings of the 8th international workshop on image analysis for multimedia interactive services, WIAMIS'2007, June 2007, pp 82–85
12. Plan T, Zorpas G, Bagrodia R (2002) An extensible and scalable content adaptation pipeline architecture to support heterogeneous clients. In: Proceedings of ICDCS, pp 507–516
13. WWW Consortium (W3C) (1999) XML Path Language (XPath) version 1.0, November 1999
14. Ghallab M, Nau DS, Traverso P (2004) Automated planning: theory and practice. Morgan Kaufmann, San Mateo
15. Chapman D (1987) Planning for conjunctive goals. Artif Intell 32:333–379
16. Chiariglione L (1995) MPEG: a technological basis for multimedia applications. IEEE Multimed 2(1):85–89
17. Iftikhar N, Qadir MA, Hamid OA (2007) Group profile and ontology-based semantic annotation of multimedia data for efficient retrieval. In: Proceedings of the 2nd international workshop on context-based information retrieval 2007 in conjunction with sixth international and interdisciplinary conference on modeling and using context, August 2007
18. ISO/IEC 15938-5:2003 (2003) Information technology—multimedia content description interface—part 5: multimedia description schemes
19. Timmerer C (2008) Generic adaptation of scalable multimedia resources. Verlag Dr. Muller, Saabrücken
20. Blum A, Furst M (1995) Fast planning through Planning Graph analysis. Artif Intell 90:1636–1642
21. Xu L, Gu W-X, Zhang X-M (2006) Backward-chaining flexible planning. Lecture notes in computer science, vol 3960. Springer, Berlin, pp 1611–3349
22. Myers KL (1999) A continuous planning and execution framework. AI Mag 20(4):63–69
23. Apache Java Xalan XSLT Processor. Available online at http://xml.apache.org/xalan-j/
24. López F, Nur G, Dogan S, Arachchi HK, Mrak M, Martínez JM, García N, Kondoz A (2010) Improving scalable video adaptation in a knowledge-based framework. In: Proceedings of the 11th international workshop on image analysis for multimedia interactive services (WIAMIS 2010), under publication, April 2010
25. López F, Martinez JM, García N (2009) Automatic adaptation decision making using an image to video adaptation tool in the MPEG-21 framework. In: Proceedings of the 10th international workshop on image analysis for multimedia interactive services (WIAMIS'09), 6–8 May 2009
26. Pednault E (2007) Synthesizing plans that contain actions with context-dependent effects. Comput Intell 4(3):356–372
27. Erol K, Nau D, Subrahmanian VS (1995) Complexity, decidability and undecidability results for domain-independent planning. Artif Intell 72(1–2):75–88
28. JUnit. Available online at http://junit.org/

Fernando López received a Computer Science degree in 2004 from the Universidad Autónoma de Madrid (UAM), Spain. He obtained a fellowship grant from the Ministerio de Educación y Ciencia of the Spanish Government. Since 2004 he has been working in multimedia adaptation research within the Video Processing and Understanding Lab (VPU Lab) of the UAM wherein he is currently pursuing his Ph.D. degree in Computer Science and Telecommunications. In this period, he has published one journal paper, ten international conference papers and one book chapter. In addition, he is working as assistant professor in the Knowledge Engineering and Multimedia laboratories in the UAM. His research interests lie in the areas of multimedia adaptation and knowledge engineering.

Dietmar Jannach is a full professor at Technische Universität Dortmund, Germany and the head of the e-Services Research Group. His research interests include interactive recommender systems and conversational preference elicitation, engineering of knowledge-based systems and web applications as well as the application of Artificial Intelligence in industry. Dietmar Jannach has authored and co-authored more than 100 scientific papers in these areas and published papers in journals such as Artificial Intelligence, AI Magazine, IEEE Intelligent Systems and on conferences such as IJCAI and ECAI.

José M. Martínez received the Ingeniero de Telecomunicación degree (six years engineering program) in 1991 and the Doctor Ingeniero de Telecomunicación degree (Ph.D. in Communications) in 1998, both from the E.T.S. Ingenieros de Telecomunicación of the Universidad Politécnica de Madrid. From 1991 till 2002 he was a member of the Grupo de Tratamiento de Imágenes of the E.T.S. Ingenieros de Telecomunicación. In 2002 he moved to Universidad Autónoma de Madrid were he started the Video Processing and Understanding Lab. Besides working as Research Assistant at the Grupo de Tratamiento de Imágenes, from 1995 until 1998 he was Assistant Lecturer at the Escuela Técnica Superior de Informática of the Universidad Autónoma de Madrid and since 1998 he was Associate Professor at the Department of Signals, Systems, and Radiocommunications of the Universidad Politécnica de Madrid. Since 2002 he is Associate Professor at the Escuela Politécnica Superior of the Universidad Autónoma de Madrid. From 2003 to 2004 he was also Subdirector for Postgraduate and Doctoral Studies at the Escuela Politécnica Superior of the Universidad Autónoma de Madrid, and from 2005 to 2008 Subdirector for Postgraduate Studies and Research. His professional interests cover different aspects of multimedia in-

formation systems, focusing on content analysis, understanding and description, content adaptation and personalization for Universal Multimedia Access, and video sumarización. Besides his participation in several Spanish national projects (both with public and private funding) he has bein actively involved in European projects (Race, Acts, Telematics) dealing with multimedia information systems applied to the cultural heritage (e.g., RACE 1078 EMN, European Museum Network; RACE 2043 RAMA, Remote Access to Museums Archives), education (e.g., ET 1024 TRENDS, Training Educators Through Networks and Distributed Systems), multimedia archives (e.g., ACTS 361 HYPERMEDIA, Continuous Audiovisual Digital Market in Europe) and semantic multimedia systems (e.g., IST FP6-001765 acemedia, IST FP6-027685 Mesh).
He is author and co-author of more than 80 papers in international journals and conferences, and co-author of the first book about the MPEG-7 Standard published 2002. From 1998 to 2004 he as actively involved in the development of the MPEG-7 standard, acting as contributor, chair and co-chair in different AHGs and editor of some documents, among them the "Multimedia Description Schemes Committee Draft" (part 5 of the ISO/IEC 15938 standard) and the "Overview of MPEG-7". He has also followed and contributed to MPEG-21.
He has acted as auditor and reviewer for the EC for projects of the 5th, 6th and 7th framework program for research in Information Society and Technology (IST). He has acted and acts as reviewer for journals and conferences, and has been Technical Co-chair of the International Workshop VLBV'03 (Madrid, September 2003), Special Sessions Chair of the International Conference SAMT 2006 (Athens, December 2006), Special Sessions Chair of the 9th International Workshop on Image Analysis for Multimedia Interactive Services WIAMIS 2008 (Klagenfurt, May 2008) and Program co-chair of the 7th International Workshop on Content-based Multimedia Indexing CBMI 2009 (Crete, June 2009).



Christian Timmerer received his M.Sc. (Dipl.-Ing.) in January 2003 and his Ph.D. (Dr.techn.) in June 2006 (for research on the adaptation of scalable multi-media content in streaming and constraint environments) both from the Klagenfurt University. He joined the Klagenfurt University in 1999 and is currently a Assistant Professor (Ass.-Prof.) at the Department of Information Technology (ITEC)—Multimedia Communication Group. His research interests include the transport of multimedia content, multimedia adaptation in constrained and streaming environments, distributed multimedia adaptation, and Quality of Service/Quality of Experience. He has published more than 50 scientific papers in these areas and he was the general chair of WIAMIS 2008. Additionally, he is an editorial board member of IEEE Computer Science Computing Now. He has been actively participating in several EC-funded projects, notably DANAE, ENTHRONE, and P2P-Next. Furthermore, he participated in the work of ISO/MPEG for several years, notably in the area of MPEG-21, MPEG-M (MXM), and MPEG-V. Publications and MPEG contributions can be found under http://research.timmerer.com.

Narciso García received the Ingeniero de Telecomunicación degree (with honors) in 1976 and the Doctor Ingeniero de Telecomunicación degree (Ph.D. in communications) with summa cum laude in 1983, both from the Universidad Politécnica de Madrid, Spain. Since 1977, he has been with E.T.S. Ingenieros de Telecomunicación of the Universidad Politécnica de Madrid (tenure as Associate Professor in 1984), and, since 1990, as a Professor of Signal Theory and Communications in the Department of Signals, Systems, and Communications. In addition, he leads the Image Processing Group (Grupo de Tratamiento de Imágenes). From 1978 to 1988, he was also a Scientific Advisor at the Image Processing Department, IBM Madrid Scientific Center, Madrid, Spain. He was Coordinator of the Spanish Evaluation Agency from 1990 till 1992 and has been evaluator, reviewer, auditor, observer, and analyst of European programs since 1990. He has been Director of the Spanish delegation at the Management Committee of the Information Society Technologies Program of the Fifth Framework Program of the European Union. His professional interests cover digital image processing, digital television, computer vision, and telecommunication systems.

Hermann Hellwagner (S'85–M'95) received the M.S. and Ph.D. degrees in Informatics from the University of Linz, Austria, in 1983 and 1988, respectively. He has been a full professor of Informatics in the Institute of Information Technology (ITEC), Klagenfurt University, Austria, for ten years. His current research areas are distributed multimedia systems, multimedia communications, quality of service, and MPEG-21. He has received many research grants from national (Austria, Germany) and European funding agencies as well as from industry. Dr. Hellwagner is the editor of several books and has published more than 100 scientific papers on parallel computer architecture and parallel programming and, more recently, on multimedia communications and adaptation. He has organized several international conferences and workshops. He is a member of the IEEE, ACM, GI (German Informatics Society) and OCG (Austrian Computer Society) and member of the Scientific Board of the Austrian Science Fund (FWF).