An extensible framework for knowledge-based multimedia adaptation

Dietmar Jannach, Klaus Leopold and Hermann Hellwagner

University Klagenfurt,
9020 Klagenfurt, Austria
{dietmar.jannach, klaus.leopold, hermann.hellwagner}@uni-klu.ac.at

Multimedia content is becoming increasingly important in many areas not only for pure entertainment but also for commercial or educational purposes like, e.g., distance learning or online training. In parallel, the rapid evolution in the hardware sector brought up various new (mobile) end user devices like pocket PCs or mobile phones that are capable of displaying such content. Due to the different capabilities and usage environments of these devices, the basic multimedia content has to be adapted in order to fit the specific devices' capabilities and requirements, whereby such transformations typically include changes in the display size or quality adaptation. Based on the capabilities of the target device that can be expressed using recent multimedia standards like MPEG-21, these adaptation steps are typically carried out by the video server or a proxy node before the data is transferred to the client. In this paper, we present a software framework and implementation of such a multimedia server add-on that advances state-of-the-art technology in two ways. First, the framework supports the integration of various (already existing) multimedia transformation tools based on declarative interface and semantic capability descriptions in a way comparable to *Semantic Web Services* approaches. Second, by using the components' capability descriptions and the usage environment of the end user device, we employ a knowledge-based planning approach for dynamically constructing and executing the needed transformation program for a specific multimedia content request.

Introduction

The importance of multimedia content provision over the Internet has been steadily increasing over the last years whereby the most prominent application domains can for instance be found in the entertainment sector and in the areas of distance education and online training. Quite naturally, the field will continuously evolve as bandwidth limitations will be decreasing and new (mobile) end user devices will open new opportunities for multimedia-based applications. In a traditional multimedia content provision architecture, the content, e.g., a video, is stored on one or more servers or network nodes and streamed to the client in a certain data format and encoding. However, the variety of the new end user devices, user preferences, and other application-specific requirements like mobility show that future multimedia environments must be more flexible and adapt the content to the client's needs when transferring it over the network [3].

In a simple scenario, a high quality video stream is stored at the server in a given format and a request for this video is received. Assume that the request comes from a mobile client that uses a PDA (Personal Digital Assistant) and a low-bandwidth connection. Then, it is not reasonable to transfer the whole video in high quality or resolution, respectively, and leave the adaptation to the client. Transferring the large video file or "fat" stream would cause higher network traffic and, on the other hand, client-side transformation would consume a large portion of the limited processing power of the handheld device. Thus, an intelligent network node like, e.g., a video server or proxy will transform the video into an appropriate format before sending it to the client. The transformation steps invoked include, e.g., spatial adaptation of the display size, quality reduction, or greyscaling.

Current and emerging multimedia standards like MPEG-4 [9] and MPEG-21 [2] address the adaptation problem by providing tools to perform adaptation on multimedia streams, e.g., layered encoding, fine granular scalability, or bitstream syntax description. In this paper we describe the architecture and implementation of an intelligent multimedia server add-on that is capable of dynamically adapting a given media file to the current client's needs and capabilities. Based on declarative descriptions of the usage environment of the current session and the description of an extensible set of already existing media transformation tools, our module dynamically generates and executes "transformation plans" on the multimedia content. Due to the nature of the used algorithms and the extensibility of the description formats, the system is capable of transforming content in arbitrary ways and, on the other hand, is open for the incorporation of new tools that support, e.g., emerging multimedia standards.

The paper is organized as follows. In the next section, we give an overview of the usage scenario and describe our approach by means of an example. Next, we present implementation and algorithmic details and finally discuss how the described approach for knowledge-based program construction based on predefined modules and functions is related to the field of Semantic Web Services and Software re-use.

Example problem

Figure 1 shows the general architecture of multimedia content provision over the Internet, where different clients can access content that is stored on dedicated servers in the network where also the adaptation modules are located. In the following, we will describe a simple but realistic example in order to demonstrate the functionality of our generic adaptation module. For presentation purposes, we will utilize a pseudo notation instead of the technical internal representations (in XML). First, there is a description of the content file, in our case a video, expressed by MPEG-7 media description meta data [10]. The description contains information about, e.g., encoding, color, or resolution of the video. On the other hand, together with the client request, there is a description of the client's preferences on these parameters, which is contained in the *Usage Environment Description* which is also part of the forthcoming MPEG-21 *Digital Item Adaptation* standard (see Table 1).

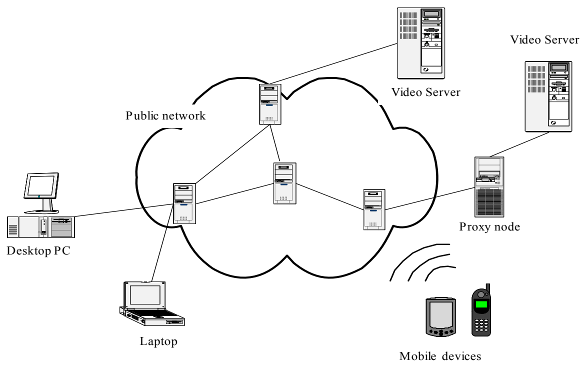

Fig. 1. Multimedia provision over the Internet

Content description:	***Usage environment description:***
type : video stream	*codec : mpeg-1*
codec : mpeg-4 Visual ES	*color : false*
color : true	*resolution : 320 x 240*
resolution : 640 x 480	

Table 1 Example for content- and usage environment descriptions

In order to transform the given video to the target state described in the *usage environment*, a set of manipulating operations have to be performed, e.g., decode the video, remove the color, change the resolution, and encode it in the required format. In fact, there are a lot of commercial and open source tools and libraries available that can perform these individual steps for different multimedia files, but in general, none of them can do all of the required transformations on different multimedia formats in one step. The main goals of our framework are to provide a mechanism to easily incorporate the functionality of different tools and to come up with a design that is general enough to cope with situations where, for instance, new description attributes or transformations are required. Therefore, we base our approach on declarative capability descriptions of the available transformation functions that are contained in libraries (plug-ins), whereby these descriptions contain the function name with the parameter descriptions as well as the preconditions and effects of the execution of the function on the content file[1]. The following example shows parts of the description of a *spatial scaler* and a *greyscaler* module.

[1] These descriptions of pre-conditions and effects are similar to action descriptions in classical state-space planning.

Spatial scaler
module : Scaler.so
function : spatialscale
Inparameters:
fb, width, height,
newwidth, newheight
Outparameters:
newfb
Preconditions:
isFrameBuffer(fb),
codec(fb,mpeg4ves),
resolution(fb,width,height)
Effects:
isFrameBuffer(newfb),
resolution(newfb,newwidth,newheight)

Grey scaler
module : Colorutils.so
function : greyscale
Inparameters:
fb
Outparameters:
newfb
Preconditions:
isFrameBuffer(fb),
color(fb,true)
Effects:
isFrameBuffer(newfb),
color(fb,false)

Given all descriptions, the goal is to come up with a sequence of adaptation steps that transform the original video into the target state. In our framework we adopt a knowledge-based approach that employs a standard state-space planner (see, e.g. [4]) for the computation of the plan. Once a plan is constructed, the meta-information in the capability descriptions is used to actually invoke the correct functions with the correct parameters from the corresponding software libraries.

A possible plan for the example problem can be described as follows, whereby the adaptation is executed on framebuffer *fb1* and the adapted framebuffer is *fb5*: In order to fit the client's usage environments, the frame buffer is first decoded, then the size is reduced to 320x240 and color is removed, and finally the framebuffer is encoded again.

Transformation sequence:
1: decode(fb1,mpeg4ves,fb2)
2: spatialscale(fb2,640,480,320,240,fb3)
3: greyscale(fb3,fb4)
4: encode(fb4,rgb,fb5)

Note, that besides the meta-data for generating the adaptation plan, the capability descriptions also contain a mapping from the symbolic names to concrete function implementations in the dynamically linked libraries (*.so or *.dll) and their respective parameter lists. Given this information, we are able to load the libraries and then dynamically invoke these transformation functions on the original multimedia stream. Figure 2 shows an overview of the architecture and the individual components of our framework.

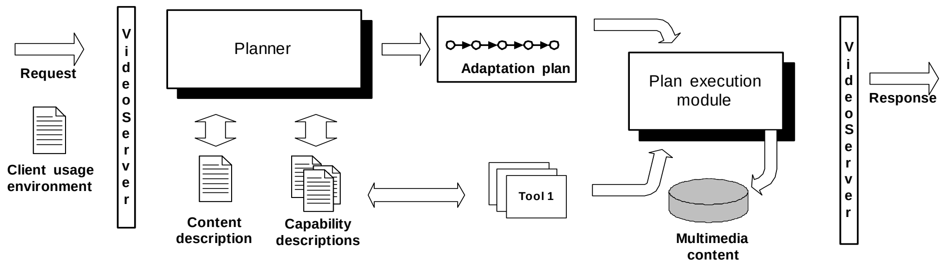

Fig. 2. Adaptation overview

The planner component

One of the major design goals for the described multimedia adaptation framework is to be as generic as possible and open for future extensions both with respect to new content adaptation tools as well as with respect to the terminology used in the capability and content descriptions. As an example, in the emerging MPEG standards, a set of terms is defined[2] that can be used to describe the client's usage environment like preferred movie genre of the user or the device's capabilities. Due to the fact that this vocabulary is subject to changes and extensions, we adopt a knowledge-based approach that is capable of operating on arbitrary sets of such symbols.

The main idea of this approach is to view the multimedia transformation task as a classical "state space planning problem" (see e.g., [4]). Such a planning problem typically consists of a set of facts (ground symbols) that describe the given situation, i.e., the start state and the target situation (the *goal state*), as well as a set of possible *actions* that change the current situation. Each of the actions is described in terms of its preconditions that must be met such that the action can be applied, and a description of the effects of the execution of the action. A *plan* consists of an ordered sequence of parameterized actions that transfer a system from the start state to the goal state. The mapping of the standard planning problem to our multimedia transformation problem is quite straightforward. The start state corresponds to the description of the existing multimedia file or stream, the target state is given by the usage environment description of the current client. The actions correspond to the available multimedia transformation steps whereby the preconditions and effects are included in the tool's capability description (see examples above).

Implementation

In a first proof-of-concept implementation, we used a light-weight Prolog realization of a means-ends planner with goal regression [4][3]. Over the last years, significant advances in the field of AI-based planning were made [1] which made the state space planning approach suitable for large real-world problems. In addition, a common language for describing planning problems (PDDL - Planning Domain Description Language [7]) was

[2] The syntax is defined using XML-Schema descriptions.
[3] AMZI! Logic Server is used as the Prolog inference engine, see http://www.amzi.com.

established in the community and is nowadays standard for most planning tools, e.g., the Blackbox [8] system. Currently, we are working on the integration of such a high-performance planner that also supports the PDDL format.

An important integration aspect for multimedia transformation planning lies in the fact that the different inputs for the planning process stem from different sources and come in different (non-PDDL) formats: Both the media description information and the usage environments are defined in an ISO-standardized XML format, i.e., they use predefined sets of tags that can be quite easily translated automatically into the required PDDL representation. At the moment, we are using a proprietary XML-based representation for the capability descriptions that is translated into the internal planner presentation. However, note that the integration of a new media transformation tool into our framework requires that a capability description file is made available. These descriptions have to be constructed by hand and it is the responsibility of the engineer that the preconditions and effects of the program execution are defined in a valid and complete manner. *Valid* means that only function symbols are used that are allowed in the MPEG standards, which can be checked using standard XML-validation; *complete* means that all of the required preconditions and effects are listed, whereby no automatic check is possible in general. Note however, that in the error case where the action description uses undefined symbols, the planner will produce no plans that include such an action but try to use other transformation plug-ins to reach the goal state. In fact, the planning mechanism works with arbitrary symbols such that any extensions of the allowed terms (e.g., a new codec) or plug-ins do not affect the implementation of the domain-independent planning algorithm and framework.

Evaluation and optimizations

The complexity of such planning problems depends heavily on the length of the computed plans and on the number of available actions. Our experiments show that the length of the required transformation sequences is rather short, i.e., it typically does not exceed five or six steps. In our test scenario with around twenty different transformation actions, plans of such a length can be computed in a few milliseconds for all cases where a plan exists. On the other hand, the un-optimized planner implementation needs less than one second to determine that no such plan exists. Note that in general there are no hard real-time restrictions for plan generation in our application scenario because once a transformation plan is computed it is valid for a long time during video streaming. A new plan might be computed if, e.g., the client's bandwidth changes. Nonetheless, besides the integration of a high-performance solver, other improvements in the planning component are possible. First, there are several possible plans to reach the same goal state and the *quality*, i.e., effectiveness of the plan can depend on the order of the plan steps. As an example, for performance reasons it might be reasonable to reduce the video's size before the color is reduced, although the inverse order would also result in the same desired video format. At the moment, this knowledge is encoded as heuristics in the knowledge base but future versions will possibly require a more elaborated approach. The second research challenge concerns situations where no plan can be found that *exactly* produces the desired format. In these cases we need mechanisms to define how an *acceptable* solution for the client looks like. Other performance improvements can be reached more simply with the means of, e.g., a *plan cache*: Once a plan is computed, we can store this plan together with the client's preferences and the media content description and access the plan immediately once a similar request is received.

Plan execution module

After successful computation of the plan, the corresponding audio, video, and image processing tools have to be invoked by the *plan execution module* (see Figure 2) on the original media files in correct order using the correct parameter sets. The main aspects for developing the plan execution module are:

- We have to incorporate and utilize already existing multimedia manipulation libraries like the FFMPEG-toolkit[4].
- We need an approach where we can easily plug in new tools without changing the core that, for instance, implements new standards or faster algorithms.
- For performance and interoperability reasons, the plan execution modules have to interface to C and C++ software components.

Typically, extensibility of a framework is reached by defining a software interface that describes the function signatures that a plug-in has to implement. At run time, the registered plug-ins are loaded and only methods defined in the interface are used. The plug-in module then performs its tasks whereby the invoking software component does not have to be aware of any internals of the newly added component. Given that there are already existing libraries to be incorporated and there are lots of vendors and software providers we cannot assume that a shared interface (using C++, in our case) is available. Consequently, we would have to implement wrapper components for each of the used modules manually that map the function calls from the interface to the concrete implementations in the libraries. In order to overcome these limitations, in our framework we specify the required details like function name and parameter lists for invoking the concrete software modules in the tool's capability descriptions as described above. The major challenge in this setting lies in the fact that the plan execution module must cope with the addition of new modules and capability descriptions and the information on the low-level function invocation is only available in textual form. When using C or C++ as programming language, however, there is no built-in means for dynamic class loading as in programming languages like Java. Consequently, a wrapper component for each new tool has to be written, compiled, and linked to the plan execution module. In our framework, however, we adopt an approach that bypasses this problem by using a combination of dynamic library loading and generic argument lists construction for C functions based on a public available C library[5] where parameter lists and return value types are externally supplied in text files, i.e., in the capability descriptions of the transformation tools. The main function invocation routine for this process is sketched in Listing 1 below.

```
(1) libname = getLibraryNameForTool(...)
```

[4] see http://www.ffmpeg.org

[5] See www.gnu.org,"ffcall" library

```
(2) funcname = getFunctionNameForTool(...)
(3) handle = loadLibrary(libname)
(4) f = getAddressForSymbol(handle,funcname)
(5) for each parameter specified
(6)     determine type and value of parameter
(7)     add param to generic param-list
(8)     callFunction(f,param-list)
```

Listing 1 Dynamic function invocation

When executing the individual steps of the plan, it is important to note that the input for one function can correspond to the output of a previously called function. Typically, the already partially adapted multimedia file or stream is handed from one plan step to another. For the implementation of argument passing, we use lookup tables that contain all the symbols that are used in the generated plan that store the actual variable values after each transformation step.

Relation to Semantic Service Mark-up with DAML/S

The idea of describing the capabilities of a software component, i.e., its specification, in a formal manner is not new. Besides the standard use of formal specifications in the requirements engineering phase, such specifications can for instance be used to (partially) automate the software *reuse* process ([13]). In these approaches, the functionality and behavior of the components of an existing software library are explicitly described using a formal language like Z [6] or in terms of, e.g., input-output pairs. Quite independently, in the last years another field emerged where formal descriptions of component or function behavior play an important role: *Semantic Web Services*. The goal of these efforts is to extend the *Web* which was once more or less a large repository of text and images − with *information-providing* and *world-altering* services accessible over the Internet. As these services are to be located and executed by computer programs or agents, they must be made available in a computer-interpretable way [12]. While the technical basis for inter-operation with standard *Web Services* is already laid with the definition of XML-based exchange formats and protocols like SOAP and WSDL[6], a really intelligent service-providing Web requires that the properties, capabilities, and effects of execution of the available services are explicitly defined. The standard example scenario for the approach [12] is from the domain of vacation planning: the agent's goal is to compose a complete travel arrangement where the individual steps include, e.g., hotel reservations or flight booking and there are several alternative providers for hotel rooms or other required resources. Ontologies [5] − simply speaking, a common understanding of the terms and their interrelationships − are the basis for any communication in distributed and heterogeneous environments. DAML-S[7] is such an ontology having its roots in the Artificial Intelligence field and is evolving as a *de facto* standard for semantic service markup that facilitates automatic discovery, invocation, composition as well as monitoring of *Semantic Web Services* [13]. DAML-S can be used to describe what a service does

[6] Simple Object Access Protocol and Web Service Description Language (www.w3c.org)
[7] www.daml.org

(*Profile*), how the service works (*Process*) and how it is actually executed (*Grounding*), whereby this information is contained in several XML resources.

Quite obviously, there is a strong relationship with our work on dynamic multimedia adaptation, particularly in the way the behavior of the services is specified (input, output, preconditions, and effects). Nonetheless, there are some important facets of the two approaches to be discussed in more detail.

*Service Discovery***.** In our approach, the complete adaptation of the multimedia content takes place on one dedicated multimedia server. Although it would in principle be possible to design simple "transformation services" as modular, distributed Web Services, we feel that such a scenario is not realistic or desirable for our application domain.

Domain ontologies. In current research efforts on Semantic Web Services the problem of establishing a common "domain ontology" is not adequately addressed. If we consider the travel arrangement planning service, it is assumed that all providers of some Web Service in the domain not only use the same vocabulary (for instance, the word/concept *reservation*) but also associate the same meaning to these terms. In the domain of video adaptation, however, we have the advantage that the terminology as well as the semantics of the involved concepts (like *resolution*) is already clearly defined in the evolving and existing MPEG multimedia standards.

Grounding. In the *grounding* part of a Semantic Web Service description, each *process* of the Web Service is related to one or more *WSDL operations* whereby the needed WSDL definitions for the low-level message exchange can be constructed from the DAML-based service descriptions. In our framework, *grounding* corresponds to the mapping of each plan step to a function call to the transformation libraries in the C/C++ libraries and the generation of adequate wrappers for module invocation.

As a result, we argue that the basic approach used for describing Semantic Web Services can be generalized and adopted in other areas of software development like software re-use based on behavior specifications or automated program construction. The main advantages of such an approach lie in the fact that with DAML and OIL (Ontology Inference Layer) an underlying mechanism and corresponding tool support for the definition and integration of domain ontologies are already available. While in our domain the application of DAML-S as core representation mechanism for describing component capabilities is quite straightforward and part of our current engineering work, our current research focuses on the generalization of the Semantic Web-Services techniques for software re-use and automated, knowledge-based program construction. We therefore see our work as a step towards this direction and the domain of multimedia adaptation as a promising field for evaluation of such an approach.

Conclusions

In this paper we have presented a framework that supports dynamic composition and execution of multimedia transformations using existing software libraries. Based on declarative descriptions of the available libraries and the requirements of the clients, a knowledge-based planner computes a sequence of required transformation steps. These transformations are then executed on the given multimedia source and the transformed content is then shipped to the client in the required format. The described approach was validated in a proof-of-concept implementation using standard multimedia manipulation libraries and a state-space planning engine.

References

[1] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[2] J. Bormans and K. Hill, editors. *MPEG-21 Overview, ISO/IEC JTC1/SC29/WG11 N5231*. October 2002. http://mpeg.telecomitalialab.com/standards/mpeg-21/mpeg-21.htm.

[3] L. Böszörmenyi, H. Hellwagner, H. Kosch, M. Libsie, and S. Podlipnig. Metadata Driven Adaptation in the ADMITS Project. *EURASIP Signal Processing: Image Communication, Special Issue on Multimedia Adaptation*, 2003.

[4] I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison-Wesley, 3rd edition, 2000.

[5] B. Chandrasekaran, J. Josephson, and R. Benjamins. What Are Ontologies, and Why do we Need Them? *IEEE Intelligent Systems*, 14(1):20–26, 1999.

[6] A. Diller. *Z: An Introduction To Formal Methods*. O'Reilly, 1996.

[7] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL, the planning domain definition language. Technical report, Yale Center for Computational Vision and Control, 1998.

[8] H. A. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *Intl. Joint Conference on Artificial Intelligence*, pages 318–325, Stockholm, 1999.

[9] R. Koenen, editor. *Overview of the MPEG-4 Standard, ISO/IEC JTC1/SC29/WG11 N4668*. March 2002. http://mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm.

[10] J. M. Martinez. *MPEG-7 Overview, ISO/IEC JTC1/SC29/WG11 N4980*. July 2002. http://mpeg.telecomitalialab.com/standards/mpeg-7/mpeg-7.htm.

[11] F. Pereira and T. Ebrahimi, editors. *The MPEG-4 Book*. Prentice Hall PTR, 2002.

[12] S. McIlraith, T.C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):46–53, 2001.

[13] M. Sabou, D. Richards, and S. van Splunter. An experience report on using DAML-S. In *Proceedings of the World Wide Web Conference 2003, Workshop on E-Services and the Semantic Web*, May 2003.

[14] P. Schojer, L. Böszörmenyi, H. Hellwagner, B. Penz, and S. Podlipnig. Architecture of a Quality Based Intelligent Proxy (QBIX) for MPEG-4 Videos. In *Proceedings of the World Wide Web Conference 2003*, May 2003.

[15] A. M. Zaremski and J. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, 1997.