Automated Context-Sensitive Dialog Synthesis for Enterprise Workflows Using Templatized Model Transformations

Amogh Kavimandan[†][*], Reinhard Klemm[‡], and Aniruddha Gokhale[†]
[†]Dept. of EECS, Vanderbilt University, Nashville, TN
[‡]Collaborative Applications Research, Avaya Labs, Basking Ridge, NJ

Abstract

In modern enterprises, workflows are essential to automating business processes. During their execution, workflows need to interact with users using a mechanism called dialogs *to deliver information and collect input that is required for further decision-making in the workflows. Information delivery and input collection by enterprise workflows is often a time-sensitive matter. Thus, dialogs have to be communicated to users in a timely fashion, which necessitates sending dialogs to user communication endpoints that permit the recipients to quickly, effectively, and conveniently view the information and provide the requested feedback. The proliferation of communication devices and clients among enterprise users implies that the middleware that creates dialogs has to provide mechanisms to tailor the content and the rendering of dialogs to a large number of endpoints. This customization poses several challenges to developing and maintaining a manageable, extensible, and flexible middleware mechanism for synthesizing dialogs from specific decision points in enterprise workflows.*

In this paper, we first describe the challenges associated with context-sensitive dialog synthesis. We discuss how we have applied templatized *model transformation techniques to automatically synthesize dialogs in enterprise workflows. We show how our templatized transformation approach supports the evolution of communication endpoints and system requirements with a minimum of downtime and invasive design changes. We demonstrate our approach in the context of a representative enterprise case study.*

1 Introduction

As part of their normal or exceptional operation, modern enterprise workflows not only set up communications (calls, conferences, chats, etc.) between decision makers in the enterprise but also need to deliver information to users and, in return, collect important input from users in a timely fashion. Such input is typically based on the information that a workflow delivered to a user and serves as the basis for further decision-making in the workflow. As enterprises strive to increase productivity and efficiency by automating their business processes through workflows, there is a growing need to accelerate this type of interaction between workflows and enterprise users. In this context, we call a mechanism to present information to a user and collect subsequent feedback from the user a *dialog* between workflow and user.

Increasingly, context-aware communications middleware is used to provide communication support to workflows including the synthesis, delivery, and rendering of dialogs. The need for accelerating the interaction between workflows and users results in a requirement to embed sophisticated context-sensitive dialog synthesis, delivery, and rendering mechanisms in the middleware to reach enterprise users in a ubiquitous fashion. Due to ever-increasing user mobility and progress in communications technology, enterprise user communication environments have changed from a limited set of fixed, largely stationary devices and clients to a wide array of personal and shared, stationary and mobile communications endpoints of differing capabilities and supporting different kinds of media. The panoply of endpoints in use in modern enterprises poses a set of complex challenges to the context-sensitive support for dialogs in communications middleware. With a potentially large volume of dialogs between workflows and certain users, the receipt, perusal of, and response to dialogs has to be as convenient and efficient for the user so as to maintain a high level of user productivity.

The true difficulty with our goal of customizing dialogs for a multitude of endpoints and based on user context, in the interest of accelerating workflow/user interactions and maintaining user productivity, is that most dialogs are created at workflow runtime. Thus, dialog customization has to be done dynamically as well and suggests the development of a large number of customization software modules (akin to device drivers). These modules would need to be constantly adapted to the ever-changing landscape of end-

[*]Contact author email:amoghk@dre.vanderbilt.edu

points.

Despite the need for such a customization, the set of dialogs tend to share strong commonalities with each other and, depending on the endpoint on which they have to be rendered, have certain distinct characteristics. Thus, there is a significant opportunity to synthesize *families of dialogs* by employing customizable and reusable software patterns and artifacts, as opposed to building them from scratch. Product-line architectures (PLAs) and its characteristic scope, commonality, and variability (SCV) [5] engineering process present a promising approach to development of families of dialogs.

This paper first describes how we have conducted SCV analysis for a family of dialogs. We then show how templatized model transformations can be used to synthesize customized dialogs. In our approach, the commonalities among the dialog variants are captured as a common set of transformation rules. Higher order parametrized rules capture the variability lending themselves well to the notion of templatized transformations.

The remainder of the paper is organized as follows: Section 2 discusses an enterprise case study that motivated our work on dialogs; Section 3 presents the challenges in context-sensitive dialog synthesis in detail; Section 4 discusses design details of our solution and lists various steps involved in the development of reusable model transformations, and how we have applied it to our case study; Section 5 compares our work with the existing literature; Section 6 provides concluding remarks and lists our plans for future work.

2 A Case Study Motivating Context-Sensitive Dialogs

An example of context-aware communications middleware that supports enterprise workflows and ubiquitous, automated dialogs between workflows and users is *Hermes* [8], developed at Avaya Labs Research. An illustrating use case scenario for Hermes, drawn from an extensive case study with several insurance companies, is a business process workflow that deals with claims in a car insurance company. The workflow gets triggered when a policy holder calls in an insurance claim for damages to his/her car. Suppose this claim raises a difficult question and it is unclear how to apply the insurance company's rules to this claim. In such a case, the workflow attempts to set up a conference call between various employees of the insurance company, including a legal expert, an appraiser, and the appraiser's supervisor, to resolve the question.

As part of the process of setting up the conference call, the workflow has to first reach out to potential participants and present (1) a conference call topic (open claim), (2) documentation or a link to documentation pertaining to the case, possibly containing audio, video, and image elements in addition to text, (3) an invitation to a conference call, and (4) a range of user response options to establish the user's ability, availability, and willingness to participate in the conference call.

These four items constitute a simple dialog in Hermes. For example, the dialog may first provide the information *"There is an open claim from policy holder 243779 that cannot proceed due to a mismatch between the appraised damage and a corporate limit on lifetime coverage for a vehicle. For more information, please consult case number 243779-041"*. Next, the dialog may pose the question *"Can you be available to participate in a voice conference about this claim at 2 PM EDT today?"* Eventually, the dialog gives the user a range of response options including "Yes", "1 hour earlier", "1 hour later", "Only if you cannot find somebody else", and "No".

A communications-enabled workflow platform like Hermes could, of course, simply send a notification of a pending dialog to the recipient via an email that contains a link to an enterprise portal with the actual dialog. The dialog could then be an HTML form or similar. However, this procedure may lead to many scenarios where the recipient may not receive or respond to the dialog in a timely fashion, thus violating our stated goal of accelerating the interaction between workflows and users. The following are some of these scenarios:

- The recipient is in a location such as a car, an off-site meeting, or a conference room with sporadic, limited, or no email access.
- Due to a focus on other activities, the recipient is not checking incoming email frequently enough.
- The recipient can receive email on a mobile device which does not have access to the enterprise portal.
- The mobile device of the recipient cannot render Web pages or makes it very inconvenient to navigate the enterprise portal and dialogs.
- The recipient is driving or for other reasons is not in a position to read a dialog or use a manual input device (mouse, keyboard, keypad) to respond to it.

To remedy *some* of the problems that the above approach to conveying dialogs to a user incurs, the Hermes middleware attempts to send (1) a dialog topic (Item 1 above), (2) a URL for the actual dialog in a Web-based portal (i.e., a link to Items 2-4 above) to an endpoint that the user is likely to be present on at this time. Hermes makes a determination of the target endpoint based on the user's context information which includes information about the user's presence and activities on various monitored endpoints such as telephones, instant messaging (IM) and email clients, Web browsers, etc. However, the user must access the URL via a Web browser to find the dialog in question. As with the email approach described in the previous paragraph, forcing the user to log into the Web portal, finding the dialog there, reading it, and responding to it via a mouse or keyboard may not be possible in a timely fashion or, at the least, may negatively impact user convenience and productivity.

Our goal for Hermes is therefore to send the entire dialog to a specific endpoint and to customize its rendering to this endpoint and a given communication modality (voice, IM, email, Web, SMS, etc.). For example, assuming that the user is present and active on a Web browser on his/her office computer, a dialog may be presented as an HTML popup in that browser [12]. This endpoint is also suitable for presenting the supporting documentation (Item 2 above) about the insurance claim. The response options in the dialog can be rendered as HTML buttons. In addition, a more sophisticated response option may be included, such as a text box that allows a freeform specification of a different time, day, and modality by the user.

On the other hand, suppose the employee is known to be present on his/her mobile phone that has no (known) data connectivity and only a standard phone numeric keypad. Due to its limited hardware capabilities, the content of the dialog to be sent to the mobile phone ought to be significantly different from the HTML popup described earlier. Moreover, the mobile phone user may not be in a position to read the dialog on the mobile device or respond via the keypad because he/she may be driving a car at this point in time. Thus, the dialog may best be rendered as a *call* to the mobile phone with a VoiceXML (VXML) script that first renders Item 1 as voice, skips Item 2 except for a brief summary of the documentation, and renders Item 3. Finally, for collecting the user's response, the script reads the available response choices and asks for either a voice response (*"Yes"* etc.) and/or a key input (*"Press 1 for Yes"* etc.).

Clearly, the set of customized dialogs derived from a defined sequence of Items 1-4 share strong commonalities with each other and, depending on the endpoint on which they have to be rendered, have specific distinct characteristics. Next, we use the case study described in this section to present a detailed discussion of the challenges involved in dynamically adapting the content and rendering of dialogs based on user context ("context-sensitive dialog synthesis") in enterprise communications middleware such as Hermes. We demonstrate how we have used model transformations for the automatic synthesis of dialogs from specific decision points in enterprise workflows.

3 Design Challenges in Context-Sensitive Dialog Synthesis

Using the case study of Section 2 we now discuss the design challenges in automatically synthesizing dialogs based on the user context.

1. Programmatic, customizable mappings for dialog creation. The dialog in our insurance example asked only one simple question to ascertain the ability, availability, and willingness of an employee to participate in a conference call. However, dialogs in general may be much more complicated and may involve a sequence of sub-interactions. Currently in Hermes, the content of a dialog is a simple text template that a workflow designer manually creates for a specific decision point in a workflow and that can be parameterized at runtime with user names, URLs, case and policy numbers, etc. The entirely manual creation and maintenance, especially of complicated dialogs for a large number of workflow decisions points, requires significant efforts.

Instead, we would like to come up with programmatic mappings from workflow decision points and user context to dialogs on specific endpoint types. User context not only determines which endpoint to send the dialog to and to render on but also ideally results in adjusting the dialog content to the specifics of a user. If, for example, one of the recipients of the insurance dialog in our above example is hearing- or vision-impaired or is most fluent in a language other than the insurance company's official business language, the dialog would ideally accommodate these user-specific parameters. Different sets of employees also have different skill sets. For example, the appraiser in our case study may receive case details as part of the dialog that are meaningful only to somebody with expertise in appraising car damages.

Thus, programmatic mappings must allow customization of the output dialogs based on parameters outside the workflow decision point and user context, such as the enterprise for which the workflow is designed, the vertical market in which the enterprise is operating, or technical constraints of the communications middleware.

2. Dialog formatting and rendering. Even though content formatting and rendering of a dialog are inextricably intertwined, we list the determination of how to format and render a dialog separately from the content selection because the emphasis is different in both challenges. The challenge in formatting a dialog for and rendering it on a target communication endpoint is the large number of static and dynamic characteristics of endpoints, resulting in vastly different types of formatting and rendering options for basically the same dialog content.

The static characteristics include the modality (voice, IM, email, Web, SMS, plain text, etc.), processing power, screen size and resolution, type of input devices attached to the endpoint, audio/video capabilities, etc. The dynamic characteristics include current data connectivity and battery power. An explanation of how such characteristics are determined in Hermes is provided in [12]. For example, the same abstract dialog may have to be rendered as a VXML script over a voice connection, via an HTML form on a mobile device, or as a sequence of SMS or IM messages, each one of which would require the user to reply with an SMS/IM message.

3. Response option definition. To be really useful, many dialogs require fairly differentiated or complex feedback from the user, based on response options given in the dialog. Our insurance example listed response options such as "1 hour earlier", "1 hour later", "Only if you cannot find somebody else" in addition to "Yes" and "No". Clearly, these options unreasonably limit the recipient's expressiveness in terms of the most desirable time and communication modality of and willingness to participate in the conference call. The situation is exacerbated in more complex dialogs.

There is a trade-off between the number of response options in a dialog on the one hand and user convenience and productivity on the other hand. Too few response options may frustrate the user because the response that the user might like to give is not part of the dialog. Too many response options may frustrate the user because it takes too long and too much effort to peruse and understand the given response options, and to select the most appropriate one.

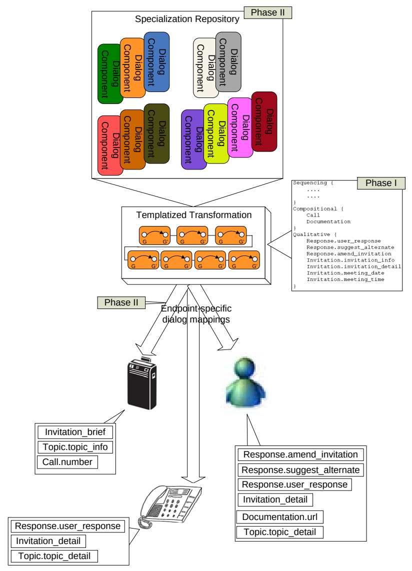
4. Linking to additional documentation. Our stated goal in Section 2 was to render the entire dialog in a target communication endpoint, including potentially multimodal supporting documentation (Item 2 in Section 2), in the interest of a timely delivery of information to the user, collection of a response, and increased user productivity and convenience. However, even lengthy or rich text documentation, let alone audio/video or other multimodal documentation pertaining to the dialog, is often infeasible or too costly to render on a given endpoint. In such cases, the programmatic mapping to dialogs would have to produce a solution that leaves the supporting documentation out of dialogs but allows dialog recipients to access it as easily and quickly as possible. A fallback solution is always to email links to the documentation to the recipient but more sophisticated options may be possible as well. For example, the recipient of a dialog on a mobile device may have the option of sending an SMS with a fax number to an enterprise server that would then send out supporting text documentation to that fax number.

5. Extending customizable mappings to new endpoints. The steady evolution of communication media and endpoints, in particular mobile devices and more powerful enterprise-class hard- and softphones, increases the complexity of managing dialogs. For example, suppose the insurance company in our example had upgraded their office phone system to IP telephones with large touch-screens. A dialog sent to such a phone may now best be rendered not as a phone call but as a rich text popup on the display with user response options rendered as touch buttons. Thus the programmatic, customizable mappings from workflow decision points to dialogs must be flexible enough to accommodate new endpoints with relatively minor changes to the mappings and negligible workflow execution downtime.

Section 4.1.1 describes how our approach helps modularize these mappings and separate their variabilities, and Section 4.2 discusses how developers can incorporate new endpoints by (partially) using existing mappings.

4 Templatized Model Transformation for Dialog Customization

This section describes how we have used *Model transformations Templatization and Specialization (MTS)* [10], which is our templatized transformation framework, for dialog customization. At the heart of our dialog synthesis approach are two phases shown in Figure 1 and described below:



```
Sequencing {
    ....
    ....
}
Compositional {
    Call
    Documentation
}
Qualitative {
    Response.user_response
    Response.suggest_alternate
    Response.amend_invitation
    Invitation.invitation_info
    Invitation.invitation_detail
    Invitation.meeting_date
    Invitation.meeting_time
}
```

Figure 1: MTS: Model Transformation Templatization and Specialization

- **Phase I: SCV analysis.** In this offline phase, developers analyze their transformations and identify their commonalities and variabilities across workflow structure patterns, various attribute values, and mapping rules. The result of this analysis phase is fed into the transformation in terms of a simple *constraint notation specification* discussed in detail in Section 4.1.1. This phase is similar to coding templatized functions in C++ that captures the pattern of the code to be generated later by the compiler.
- **Phase II: Transformation specialization.** In this phase, the developers use higher order transformations defined in MTS to generate a variability metamodel (VMM) from their templatized model transformations. VMM is useful in creating a *specialization repository* of a particular product-line and is created in terms of VMM models. The specialization repository

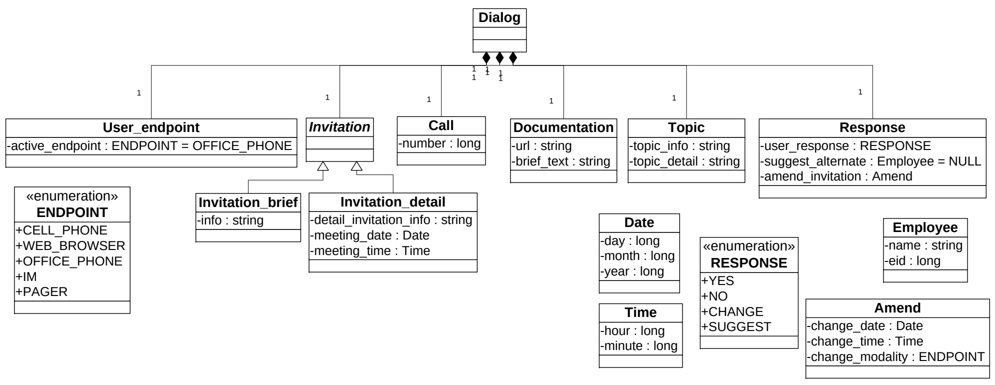


Figure 2: Generic dialog structure for supporting enterprise communication

contains a VMM model for each communication endpoint. A combination of templatized transformation and a VMM model (corresponding to that endpoint) is used for generating the communication dialog for a specific endpoint. This phase is similar to template instantiation in C++ when the compiler automatically generates the code specific to the actual type of parameters passed to a template function.

We have used the Generic Modeling Environment (GME) [15] as the modeling environment in MTS. GME provides a general-purpose editing engine, a separate view-controller GUI, and a configurable persistence engine. GME is meta-programmable, and thus the same environment used to define modeling languages is also used to build models, which are instances of the metamodels.

For defining transformation rules we have used the Graph Rewriting And Transformation (GReAT) [9] language. GReAT is developed using GME and can be used to define model-to-model transformation rules using its visual modeling language. It also provides the GReAT Execution Engine (GR-Engine) for execution of these transformation rules for generating target models.

Model transformations in GReAT require source and target domain-specific modeling languages (and their corresponding metamodels). A transformation developer uses the GReAT visual transformation language to define various translation rules in terms of patterns of input and output modeling objects. Finally, developers execute GReAT's transformation engine called GR-engine that translates an input model using the specified rules into an output model. The remainder of this section describes the details of our approach.

4.1 Applying MTS for Context-Sensitive Dialog Synthesis

In this section, we first explain the details of a generic dialog generated by the workflow which acts as the input for our templatized transformation. We then discuss the constraint notation in MTS, and how it can be used for separating transformation variabilities for our customizable mappings. Finally, we show how VMM models in the specialization repository are incorporated into the middleware to yield context-sensitive dialogs for individual endpoints.

4.1.1 Phase I: SCV Analysis

Figure 2 shows a simplified UML notation of a generic dialog in an enterprise workflow in the Hermes middleware. The following are some of the properties/attributes in this communication dialog: (1) `User_endpoint` indicates the most active endpoint the employee has used, (2) `Call` specifies a number to call to retrieve a dialog, (3) `Documentation` contains further details about documentation pertaining to the claim in question, (4) `Topic` specifies details about the claim itself, (5) `Invitation` contains details about a conference call for discussing the claim, and (6) `Response` allows an employee to reply to the invitation in the current dialog.

Additionally, the `Response` element allows an employee to suggest an alternate employee that can be contacted about the claim in question. The latter can be done by setting `user_response` to FALSE and populating the `suggest_alternate` attribute with an appropriate value. Similarly, a user can send a request for a modified invitation to the workflow that initiated this dialog by using the `amend_invitation` enumerated data type.

As a first step in the automated synthesis of dialogs, we must first determine the commonalities and variabilities across the elements (instances). In our model transformations approach the commonalities constitute the templa-

Table 1: Dialog profiles for representative communication endpoints

Communication endpoint	Modality	Dialog properties/Attributes: Commonalities	Dialog properties/Attributes: Variabilities
Cell phone/Office phone	VXML	invitation_info, topic_info, User_endpoint	invitation_detail, meeting_date, meeting_time, user_response, topic_detail, claim_id, customer_name, customer_id, claim_date
Pager	text	invitation_info, topic_info, User_endpoint	call
Web browser	SMIL	invitation_info, topic_info, User_endpoint	user_response, suggest_alternate, Documentation, invitation_detail, meeting_date, meeting_time, topic_detail, claim_id, customer_name, customer_id, claim_date, amend_invitation

tized transformation rules while the variabilities constitute the specializations. This is achieved via the SCV analysis.

SCV analysis for Dialog profiles (i.e., properties necessary to create a dialog for an endpoint) are shown in Table 1 for three representative communication endpoints. The Modality field in the Table is a static characteristic of an endpoint and indicates the communication type used to deliver the dialog. Notice that a given endpoint may support more than one modality. The modality clearly affects the format and rendering of a dialog. For example, VXML and SMIL are W3C standard XML formats for designing interactive voice- and multimedia-based dialogs, respectively. As shown in Table 1, cell phones and office phones use VXML while Web browsers use the SMIL modality. Note that the modality will affect the format and rendering of a dialog.

In Table 1, all profiles contain at a minimum the `invitation_info`, `topic_info` and `User_endpoint` attributes. However, only the cell phone, browser, and office phone endpoint profiles contain all of the attributes in the `Topic` and `Invitation` elements of a dialog. Only the Web browser endpoint allows the user to respond with an alternate employee that can be contacted for the claim in question, or request a change in the invitation, and can optionally present documentation about the claim. Similarly, the `Call` element is present only for a pager. Note that the `User_endpoint` attribute is common for all endpoint profiles and is used in selecting the appropriate VMM in the specialization repository. We will explain the details on this model selection in Section 4.1.2.

The results of the SCV analysis must then be mapped to templatized transformation rules (for the commonalities) and constraint specifications (for the variabilities). For example, during our synthesis of a dialog family the common model elements in Table 1 get mapped directly from a generic dialog in Figure 2. The variabilities from our SCV analysis results, on the other hand, must be incorporated into the transformation so that they can be subsequently used by MTS. In our insurance case study, the variabilities can be categorized as follows:

a. **Compositional variabilities,** where model elements in each family instance/member are different and variability stems from these instances getting composed using distinct model elements. For example, compositional variability exists between pager and Web browser endpoints. Recall from Table 1 that the dialog profile of a pager endpoint consists of `Call`, `Invitation`, `Topic`, and `User_endpoint` elements. On the other hand, the profile of a Web browser endpoint is composed of `Invitation`, `Topic`, `Response`, `Documentation`, and `User_endpoint` elements.

b. **Qualitative variabilities,** where two family instances may share a common model element but not the absolute values of attributes of that element. The term quality here refers to what a system model describes as a whole, which is a collective aggregate of values of all of its attributes. Thus, even though the `Response` element is present in both the cell phone and the Web browser, the `suggest_alternate` and `amend_invitation` attributes are not applicable and are omitted for the cell phone (because of limited capabilities of the endpoint). For a Web browser endpoint, however, these attributes can be used in its dialog and are available. A similar variability exists for the attributes of the `Invitation` element for the office phone and pager endpoints.

In our MTS approach, the constraint notation specification is inserted as a special comment in the transformation rules that is transparent to the GR-engine and thus does not interfere with the engine's execution and translation logic. The constraint specification captures the variabilities in a model transformation as simple implication relations between the source and target model elements. In our insurance case study both the source and target model elements belong to the Dialog modeling language and hence the same input dialog specification is specialized and transformed into a dialog for individual endpoints. As such, the variabilities are captured in terms of generic dialog model elements. Below we show excerpts from a complete constraint specification for capturing the variabilities that we discussed in Items (a) and (b) above:

```
Sequencing {
    ....
    ....
}
Compositional {
    Call
    Documentation
}
Qualitative {
    Response.user_response
    Response.suggest_alternate
    Response.amend_invitation
    Invitation.invitation_info
    Invitation.invitation_detail
    Invitation.meeting_date
    Invitation.meeting_time
}
```

The specification is quite self explanatory – the `Qualitative` block captures all the attributes while the `Compositional` block captures all the model elements that vary between family members. The `Sequencing` block is used later in the specialization in Phase II and will be explained in Section 4.1.2.

4.1.2 Phase II: Transformation Specialization

We now describe the higher order transformation in GReAT that we have developed that auto-generates the variability metamodel (VMM) from the constraint specification in templatized transformation. The current version of this transformation contains ∼83 rules and operates on GME and GReAT metamodels, the templatized transformation itself, and source and target modeling languages (of a templatized transformation). Algorithm 1 gives some of the translation rules in this higher order transformation.

The auxiliary function $initializeVMM(V)$ on Line 5 creates VMM V and initializes its attributes required in order to define a new language in GME. As shown in the Algorithm, for all the compositional mappings in the transformation, the source and target patterns are read in Lines 13 and 18. Next the types of each modeling object, for both source and target patterns is found by parsing the respective modeling languages as shown in Lines 15 and 20. The type information is used to create appropriate modeling objects corresponding to the specified source and target patterns in Lines 16 and 21. A similar approach is taken in generating modeling objects in VMM for qualitative variabilities in constraint specification as shown in Lines 25–42. Additionally, attributes of the corresponding modeling objects are also created as shown in Lines 31–33 and Lines 38–40.

Thus, when higher order transformation represented by Algorithm 1 is applied to the transformation for our insurance case study from Section 4.1.1, a VMM is generated automatically for the dialog family. Figure 3 shows a screenshot of the generated VMM in GME. In this Figure, `InputPattern` denotes the source language pattern (e.g., it is generated in Algorithm 1 in Lines 13–16) while `OutputPattern` denotes the target language pattern (e.g., it is generated in Algorithm 1 in Lines 18–21). As stated earlier, since the same input dialog specification is refined as it is transformed in our case study, the `InputPattern` model does not contain any elements.

Variabilities are separately modeled and contained in the `Compositional` and `Qualitative` elements. The specialization repository can now be easily synthesized by

Algorithm 1: Higher order transformation algorithm for generating family-specific VMM from constraint specification

```
Input: source modeling language S, target modeling language T, set of
       templatized transformation rules R
Output: variability metamodel V
1  begin
2      transformation rule r; constraint notation block cnb; set of constraint
       notation blocks CNB;
3      compositional variability cm;set of compositional variabilities CM;
       qualitative variability qm;set of qualitative variabilities QM; pattern p;
       modeling object ob; attribute at; modeling object type type;attribute type
       atttype;
4      integer c;
5      $initializeVMM(V)$;
6      foreach $r \in R$ do
7          if $r.cnb() \neq \emptyset$ then
8              $CNB \leftarrow r.cnb()$;
9          foreach $cnb \in CNB$ do
10             if $cnb.compositionalMappings() \neq \emptyset$ then
11                 $CM \leftarrow cnb.compositionalMappings()$;
12                 foreach $cm \in CM$ do
13                     $p \leftarrow cm.SRC()$;
14                     foreach $ob \in p$ do
15                         $parseLanguage(S, ob, type)$;
16                         $createSRCObject(V, ob, type)$;
17                     end
18                     $p \leftarrow cm.TRGT()$;
19                     foreach $ob \in p$ do
20                         $parseLanguage(T, ob, type)$;
21                         $createTRGTObject(V, ob, type)$;
22                     end
23                     $composeVariabilityAssociation(V)$
24                 end
25             if $cnb.qualitativeMappings() \neq \emptyset$ then
26                 $QM \leftarrow cnb.qualitativeMappings()$;
27                 foreach $qm \in QM$ do
28                     $p \leftarrow qm.SRC()$;
29                     foreach $ob \in p$ do
30                         $parseLanguage(S, ob, type)$;
                           $createSRCObject(V, ob, type)$;
31                         foreach $at \in ob$ do
32                             $parseObject(ob, at, atttype)$;
                               $createSRCAttribute(V, ob, at, atttype)$;
33                         end
34                     end
35                     $p \leftarrow qm.TRGT()$;
36                     foreach $ob \in p$ do
37                         $parseLanguage(T, ob, type)$;
                           $createTRGTObject(V, ob, type)$;
38                         foreach $at \in ob$ do
39                             $parseObject(ob, at, atttype)$;
                               $createTRGTAttribute(V, ob, at, atttype)$;
40                         end
41                     end
42                     $composeVariabilityAssociation(V)$
43                 end
44         end
45     end
46 end
```

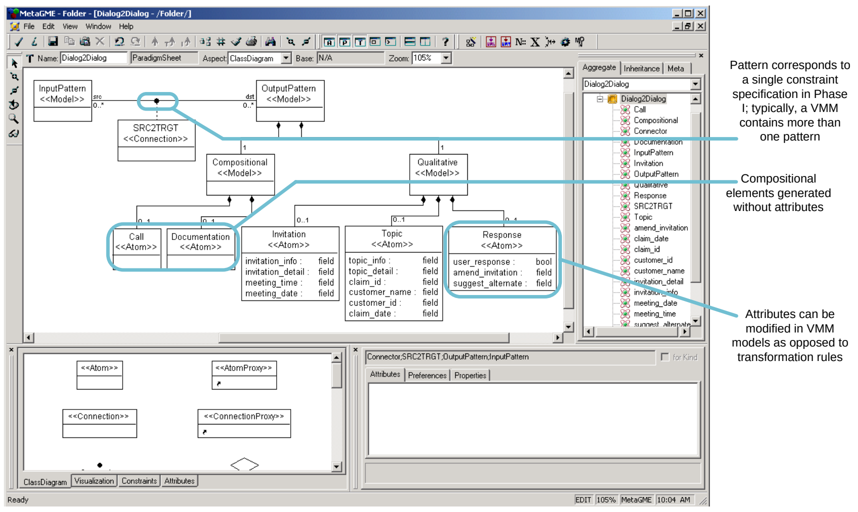

MetaGME - Folder - [Dialog2Dialog - /Folder/]
InputPattern <<Model>>
OutputPattern <<Model>>
SRC2TRGT <<Connection>>
Compositional <<Model>>
Qualitative <<Model>>
Call <<Atom>>
Documentation <<Atom>>
Invitation <<Atom>>
Topic <<Atom>>
Response <<Atom>>
Pattern corresponds to a single constraint specification in Phase I; typically, a VMM contains more than one pattern
Compositional elements generated without attributes
Attributes can be modified in VMM models as opposed to transformation rules

Figure 3: Auto-generated variability metamodel using SCV analysis results from Phase I

developers in terms of VMM models, where each model captures an individual dialog profile (for example, as shown in Table 1). Finally, VMM models are used in conjunction with our original (templatized) transformation to create context-sensitive dialogs as shown in Figure 1.

4.2 Discussion

The mappings from workflow decision points to dialogs are highly use case-specific and largely depend on the characteristics of individual dialog family members. The rendering of communication dialogs can be affected for individual endpoints by modifying the specialization repository instance at modeling level (i.e., the VMM model for that endpoint). In particular, we showed how static characteristics of an endpoint (in our example its modality), can be be used to format dialogs.

Our approach can easily be extended to include *dynamic* endpoint information, such as current data connectivity levels (bandwidth) or remaining battery power, simply by updating the specialization repository. For example, Table 2 shows two dialog profiles for handheld devices. The `handheld_1` profile can be used for devices known to have sufficient battery and bandwidth. On the other hand, `handheld_2` does not contain detailed claim information and therefore is more applicable for devices with low batter power and bandwidth.

With MTS it is possible to control the degree of flexibility in responding to an invitation. Thus, on one hand, it can be used to define a wide range of response options set for the Web browser endpoint, and on the other, a minimal response options set (with only "Yes" and "No" allowed options) for the cell phone. Some of these response options

Table 2: Using dynamic endpoint characteristics in dialog formatting & rendering

comm. endpt.	modality	Dialog Prop./Attribs.
handheld_1	text	invitn_detail, meeting_date, invitn_info, topic_info, meeting_time, user_resp., User_endpt., topic_detail, claim_id, Doc., amend_invitn., customer_id, claim_date, customer_name, suggest_altern.
handheld_2	text	invitation_info, topic_info, user_response, User_endpoint

are dictated by the type of endpoints (e.g., options such as `suggest_alternate` and `amend_invitation` can not be used for the pager endpoints). For others however, the transformation developers need to perform a careful tradeoff analysis, between providing a rich feature set and increasing employee productivity. MTS allows rapid synthesis of dialogs (e.g., each with a separate response set) and thus can be a very effective tool in a tradeoff analysis.

Finally, using VMM models for specifying variability in dialog synthesis allows developers to reuse dialog customization mapping rules for new endpoints. In addition, VMM offers the following advantages: (1) both transformations and VMM models can evolve independently, and (2) changes in requirements for dialogs targeted at a particular endpoint does not require recompilation of the model transformation.

5 Related Work

We categorize related research into: (1) domain-space research i.e., work that has attempted to address dynamic dialog customization and synthesis, and (2) solution-space research i.e., work that specifically deals with model transformation reuse.

Context-sensitive dialog synthesis techniques: Research presented in [4, 3] discusses how user interfaces can be customized based on user context information. The authors employ a model-based development process to model user communication (in terms of interactions) with a context-aware system. Services such as context-sensitive guided tours using users' mobile devices can be developed using their prototypical approach.

A number of context-aware services and frameworks have been proposed over the years [6, 19, 16] that incorporate users' location and availability, and awareness information while establishing communications between them. For example, the connector service in [6] aims at establishing communication between two users at the most appropriate time, using the most appropriate endpoints, and takes factors such as physical location, social relations and current state of the users into account [6]. Our previous work in this area [12] discussed a Web browser-based dialog system that facilitated user communications in response to events in the enterprise workflow so as to improve and accelerate decision-making.

In contrast to the above body of work, our paper focuses on customizing dialogs such that they can be appropriately rendered on user endpoints. Our work can be used in conjunction with user interface customization approaches such as [4] when different kinds of user endpoints are allowed to use the context-aware service.

Model transformation reuse & templatization techniques: The model driven architecture (MDA) [17, 13] development process is centered around defining application platform-independent models and applying (typed, and attribute augmented) transformations to these models to obtain application platform-specific models. In the context of MDA, requirements and challenges in generating specialized transformations from generic transformations are discussed in [14]. Existing model transformation toolchains [18, 7, 2] provide some support for development of higher order transformations – PROGRES and ATL allow specification of type parameters while VIATRA allows development of meta-transformations i.e., higher order transformations that can manipulate transformation rules (and hence, model transformations).

Reflective model driven engineering (MDE) approach [1] proposes a two dimensional MDA process by expressing model transformations in a tool- or platform-independent way and mapping (i.e., transforming) this expression into actual tool- or platform-specific model transformation expressions. Although reflective MDE focuses on having durable transformation expressions that naturally facilitate technological evolution and development of tool-agnostic transformation projects, mappings still have to be evolved (i.e., modified) with change in platform-specific technologies. MTS on the other hand, is concerned with managing and evolving model transformation variability in systems developed using an MDA process. Parameterization techniques supported by MTS can be highly effective in managing variability of mappings from platform-independent to specific forms in the context of the above body of work.

An aspect-oriented approach to managing transformation variability is discussed in [21, 20] that relies on capturing variability in terms of models and code generators. In essence, using the aspect-oriented approach requires developers to learn a new modeling language for creating aspect models for their product-line. In contrast, MTS generates VMM from variability specification in the templatized transformation to automate the entire process. The population of VMM models itself, as shown in Section 4, does not involve learning an entirely new language since all its modeling objects are part of a source (or target) modeling language of the transformation.

6 Conclusion

In this paper we discussed a model transformation-based approach to customizing dialogs between enterprise workflows and users to a variety of user communication endpoints, from cell phones to Web browsers to office phones. Our two-phase approach to dialog specialization offers the following benefits: (1) It allows developers to separate variabilities in their dialog mappings in Phase I such that templatized model transformations can be developed. (2) Through use of VMM models in the specialization repository in Phase II, developers can easily create family instance-specific dialogs for individual endpoints, and extend existing mappings such that dialogs for new endpoints could be synthesized.

The following is a summary of lessons learnt from our work:

• **Mapping of workflows to context-sensitive communication is essential to rapid decision-making in enterprises.** With the increasing reliance on automated processes in enterprises, there is an immediate need to accelerate the communication between workflows and enterprise employees. Such communication enables employees to make informed business decisions with lesser "turnaround time", which ultimately leads to increased overall productivity and efficiency of the enterprise. Our MTS approach provides a simple, extensible solution to context-sensitive dialog creation. The templatized transformation together with the specialization repository are useful in automatically mapping workflow decision points onto appropriate dialogs. Since the variabilities are expressed as VMM models, addition of dialogs, corresponding to new endpoints introduced in the enterprise, can be achieved simply by creating a new VMM model.

• **Templatized transformations & specializations can be more widely applicable to development of PLAs in other domains.** The MTS approach in this work has been demonstrated specifically in the context of context-sensitive dialog synthesis. However, the MTS toolchain, its various development processes and artifacts are not domain-specific and can be re-targeted for other domains. Thus, the toolchain itself can be applied in general to any product-line development scenario, without requiring any change. An effort is underway in applying the MTS approach to configuration of heterogeneous component-based distributed applications [11, 10].

In the future we plan to apply our MTS approach to more complex customizable mappings. In our ongoing work, we are extending our approach to allow customization of connectors that map abstract communication primitives (examples of such primitives include "set up a conference call", or "call an available expert on Java") generated by the Hermes workflows to the concrete underlying communications infrastructure.

MTS is available in open source as part of the CoSMIC MDE tool suite from `www.dre.vanderbilt.edu/cosmic`.

References

[1] J. Bézivin, N. Farcet, J.-M. Jézéquel, B. Langlois, and D. Pollet. Reflective Model Driven Engineering. In *Proceeding of The 5th International Conference on Unified Modeling Language, Modeling Languages and Applications*, pages 175–189, Oct. 2003.

[2] J. Bézivin, G. Dupé, F. Jouault, G. Pitette, and J. E. Rougui. First Experiments with the ATL Model Transformation Language: Transforming XSLT into XQuery. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2003*. ACM, 2003.

[3] T. Clerckx, K. Luyten, and K. Coninx. Dynamo-aid: A Design Process and a Runtime Architecture for Dynamic Model-based User Interface Development. In *Engineering Human Computer Interaction and Interactive Systems Lecture Notes in Computer Science*, volume 3425/2005, pages 77–95. Springer Berlin/Heidelberg, July 2005.

[4] T. Clerckx, C. Vandervelpen, K. Luyten, and K. Coninx. A Prototype-Driven Development Process for Context-Aware User Interfaces. In *Task Models and Diagrams for Users Interface Design Lecture Notes in Computer Science*, volume 4385/2007, pages 339–354. Springer Berlin/Heidelberg, Aug. 2007.

[5] J. Coplien, D. Hoffman, and D. Weiss. Commonality and Variability in Software Engineering. *IEEE Software*, 15(6), November/December 1998.

[6] M. Danninger, G. Flaherty, K. Bernardin, H. K. Ekenel, T. Köhler, R. Malkin, R. Stiefelhagen, and A. Waibel. The Connector: Facilitating Context-aware Communication. In *Proceedings of the 7th International Conference on Multimodal Interfaces (ICMI 2005)*, pages 69–75, Trento, Italy, Oct. 2005. ACM.

[7] G. Csertán and G. Huszerl and I. Majzik and Z. Pap and A. Pataricza and D. Varró. VIATRA: Visual Automated Transformations for Formal Verification and Validation of UML Models. In *Proceedings of 17th IEEE International Conference on Automated Software Engineering*, pages 267–270, Edinburgh, UK, 2002. IEEE.

[8] A. John, R. Klemm, A. Mani, and D. Seligmann. Hermes: A Platform for Context-Aware Enterprise Communication. In *Proceedings of the 3rd International PerCom Workshop on Context Modeling and Reasoning (CoMoRea)*, Pisa, Italy, Mar. 2006. IEEE.

[9] G. Karsai, A. Agrawal, F. Shi, and J. Sprinkle. On the Use of Graph Transformations in the Formal Specification of Computer-Based Systems. In *Proceedings of IEEE TC-ECBS and IFIP10.1 Joint Workshop on Formal Specifications of Computer-Based Systems*, Huntsville, AL, Apr. 2003. IEEE.

[10] A. Kavimandan and A. Gokhale. A Parameterized Model Transformations Approach for Automating Middleware QoS Configurations in Distributed Real-time and Embedded Systems. In *Proceedings of ASE Workshop on Automating Service Quality, (WRASQ 2007)*, Atlanta, GA, Nov. 2007.

[11] A. Kavimandan and A. Gokhale. Automated Middleware QoS Configuration Techniques using Graph Transformations. Technical Report ISIS-07-808, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, May 2007.

[12] A. Kavimandan, R. Klemm, A. John, D. Seligmann, and A. Gokhale. A Client-Side Architecture for Supporting Pervasive Enterprise Communications. In *Proceedings of the IEEE International Conference on Pervasive Services (ICPS) 2006*, Lyon, France, June 2006. IEEE.

[13] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture(MDA^{TM}): Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, Apr 2003.

[14] J. Kovse. Generic Model-to-Model Transformations in MDA: Why and How? In *Proceeding of 1st OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, Nov. 2002.

[15] A. Ledeczi, A. Bakay, M. Maroti, P. Volgysei, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing Domain-Specific Design Environments. *IEEE Computer*, pages 44–51, November 2001.

[16] A. E. Milewski and T. M. Smith. Providing Presence Cues to Telephone Users. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 2000)*, pages 89–96, Philadelphia, PA, Dec. 2000.

[17] Object Management Group. *Model Driven Architecture (MDA)*, OMG Document ormsc/2001-07-01 edition, July 2001.

[18] A. Schürr, A. J. Winter, and A. Zündorf. Progres: Language and environment. In H. Ehrig, G. Engels, H. Kreowski, and G. Rozenberg, editors, *Handbook on Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools*, pages 487–550. World Scientific Publishing Company, 1999.

[19] J. C. Tang, N. Yankelovich, J. Begole, M. V. Kleek, F. C. Li, and J. R. Bhalodia. ConNexus to Awarenex: Extending awareness to mobile users. In *Proceedings of the International Conference on Computer Human Interaction (CHI 2001)*, pages 221–228, Seattle, WA, Apr. 2001. ACM.

[20] M. Voelter and I. Groher. Handling Variability in Model Transformations and Generators. In *Companion to the Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2007)*, Montréal, Canada, October 2007. ACM.

[21] M. Voelter and I. Groher. Product Line Implementation using Aspect-Oriented and Model-Driven Software Development. In *Proceedings of the 11th Annual Software Product Line Conference (SPLC)*, Kyoto, Japan, Sept. 2007.