Feature-based Modelling of a Complex, Online-Reconfigurable Decision Support Service

Martin Karusseit\textsuperscript{a,1} Tiziana Margaria\textsuperscript{b,2}

\textsuperscript{a} Universität Dortmund, Germany
\textsuperscript{b} Universität Göttingen, Germany

Abstract

In this paper, we show how the concepts of components, features and services are used today in the Online Conference System (OCS) in order to marry the modelling of functionally complex, online reconfigurable internet services at the application level with the needs of a model-driven development amenable to analyze and verify the models. Characteristic of the approach is the coarse-grained approach to modelling and design of features and services, which guarantees the scalability to capture large complex systems. The interplay of the different features and components is realized via a coordination-based approach, which is an easily understandable modelling paradigm of system-wide business processes, and thus adequate for the needs of industrial application developers.

Keywords: Object-oriented and component-based development, feature-based systems, coordination-based approaches

1 Features as Modelling Entities

The concrete application scenario considered in this paper is an example of developing complex, collaborative, online reconfigurable internet services. Such services combine heterogeneous architectures with black/grey-box implementation, which is one of the typical difficulties of large, incrementally developed systems, in particular concerning the continuous redesign and modifications

\textsuperscript{1} Email: martin.karusseit@cs.uni-dortmund.de
\textsuperscript{2} Email: margaria@cs.uni-goettingen.de
arising during the lifetime of the systems [21]. In order to provide an understandable and manageable high-level model of the system, we design the whole application by defining and enforcing entities of complex behavior called features, which are superposed on a base system and coordinated within the system under development. The challenge is precisely how to handle this superposition and coordination in an understandable, well partitioned, and manageable way.

In our applications the user-level flow of control is of central importance: this level is realized via coordination graphs, called in our environment Service Logic Graphs (SLGs). They establish a specific modelling level which allows a direct modelling of these control aspects on their own, at the feature level, without being overwhelmed by more detailed implementation concerns like, e.g., data structures, architectures, deadlocks, and load balancing. These concerns are hidden from the application logic designer and are taken care of at a different level, during the object-oriented, component-based development of the single functionalities (which may well have a global impact on the system), capturing individual user-level requirements. Particularly well-suited for our approach are therefore applications where the user-level flow of control frequently needs adaptation or updating, as it is the case when addressing user-specific workflows or situation-specific processes. Besides the design of a number of internet services, typically role-based, client-server applications like the Online Conference Service [10,12] that we are here using as a running illustrative example, we have also successfully addressed the modelling and test of (web-enabled) CTI applications, as presented in [4].

Our understanding of the feature concept can be well explained along the similarities and differences wrt. the definitions of feature and of feature-oriented description given in the literature. We learned to appreciate the concept and the use of features in the context of Intelligent Networks [6,7,20], but our notion of features is more general than e.g. what defined in [3], in order to also capture a more general class of services like online, financial, monitoring, reporting, and intelligence services:

**Definition 1.1** [Feature]

(i) A *feature* is a piece of (optional) functionality built on top of a base system.

(ii) It is *monotonic*, in the sense that each feature *extends* the base system by an *increment* of functionality.

(iii) The description of each feature *may* consider or require other features, additionally to the base system.

(iv) It is defined from an *external* point of view, i.e., by the viewpoint of users
and/or providers of services.

(v) Its granularity is determined by marketing or provisioning purposes.

Differently from the IN setting, where the base system was the switch, offering POTS functionality, and the features were comparatively small extensions of that behaviour, we have (e.g. in CSCW-oriented internet services like the OCS) a lean base service, that deals with session, user, and role-rights management, and a rich collection of features.

In the traditional telephony setting, features are understood as modifiers of the base service [3], which are basically executed sequentially, each of them departing and returning to the base service (the so called ”daisy” or sequential model of execution [16]). In web-based applications, the proportion between base system and features is more extreme: web services have a minimal skeleton service, and are almost completely constituted by features.

The literature is rich of approaches for the modelling and verification of such feature-based systems: for example, advanced compositional model checking techniques have been proposed in [9,2], which deal efficiently with the specification of properties of feature-based systems. Their goal is to be able to partition both the functionality and also the requirements, and to achieve automatic property composition at checking time.

In order to account for complex evolutions of services, we allow a multi-level organization of features, whereby more specialistic features are built upon the availability of other, more basic, functionalities.

In order to keep this structure manageable and the behaviours easily understandable, we restrict us to monotonic features, which are guaranteed to add behaviour. Restricting behaviour, which is also done via features in other contexts (e.g. in a feature-based design in [5]) and similarly in aspect-oriented design [8]), is done in an orthogonal way in our setting, via constraints at the requirements level. Redefinition of behaviour via features, which is considered e.g. in [5], with a clear influence of object-oriented design practices, is not allowed in our setting. Attempts to define and analyze interactions in presence of redefining features have clearly shown that it is very hard to deal with such a feature model, and that it is preferable to avoid it.

Additionally, we distinguish between features as implementations and properties of feature behaviours. Both together yield the feature-oriented description of services enforced in our work.

**Definition 1.2** [Feature-oriented Description]

(i) A feature-oriented service description of a complex service specifies the behaviours of a base system and a set of optional features.
(ii) The behaviour of each feature and of the base system are given by means of Service Logic Graphs (SLGs) [7].

(iii) The realization of each SLG bases on a library of reusable components called Service Independent Building-Blocks (SIBs).

(iv) The feature-oriented service description includes also a set of abstract requirements that ensure that the intended purposes are met.

(v) Interactions between features are regulated explicitly and are usually expressed via constraints.

(vi) Any feature composition is allowed that does not violate the constraints.

In contrast to [3], we distinguish the description of the feature’s behaviour from that of a feature’s legal use. Restrictions of behaviours are in fact expressed at a different level, i.e. at the requirements level (via temporal logic constraints), and are part of an aspect-oriented description of properties that we want to be able to check automatically, using formal verification methods.

In the following Sections, we first introduce our concrete example: the Online Conference Service (OCS) (Sect. 2), subsequently we describe in detail the adopted feature-oriented description technique and illustrate the interplay of the described concepts in Sect. 3 using a specific portion of the OCS. Finally Sect. 4 contains our conclusions.

2 Application: The Online Conference Service (OCS)

The OCS (Online Conference Service) (see [10,11] for a description of the service and of its method of development) proactively helps Authors, Program Committee Chairs, Program Committee Members, and Reviewers to cooperate efficiently during their collaborative handling of the composition of a conference program. It is customizable and flexibly reconfigurable online at any time for each role, for each conference, and for each user. The OCS has been successfully used for over 35 computer science conferences, and many of the ETAPS Conferences. In 2004 and 2005 it served them all, with 6 instances of the service running in parallel.

The service’s capabilities are grouped in features, which are assigned to specific roles. In the OCS, a single user may cover many roles (e.g., PC Members may submit papers and thus be simultaneously Authors), and can switch between them at any time during a working session. A fine granular roles and rights management system takes care of the adequate administration of the context, role and user-specific permissions and restrictions. The roles cooperate during the lifetime of a PC’s operations and use the OCS capabilities, which are provisioned at the feature level. Through the cooperation of its fea-
Fig. 1. Role-based Feature Management in the OCS

Features, the OCS provides timely, transparent, and secure handling of the papers and of the related submission, review, report and decision management tasks.

2.1 Feature Description

Features are assigned to the roles, and can be fine-granularly tuned for conference-specific policies. E.g., some conferences practice blind reviewing, meaning that certain fields of the article submission form are not published to the reviewers and secret between Author and PC Chair. In this paper we focus on the principal features and on the discussion of their implications for feature-based service development. The following features are illustrative of the size and granularity adopted in the OCS, while the full collection is shown in Fig. 2.

Article Management: Over 30% of the service activity consistently concerns this feature. The central page corresponding to this feature is the Article overview page (Fig. 1(bottom)), which also contains links to activities like report submission or paper delegation that go beyond just providing access to the article and article managements pages.

Delegation Management: here the PC Chair delegates papers to appropriate PC Members and it supports PC members in their dialogue with their subreviewers. It manages the PC Members and Reviewers tasklists. The delegation process is iterative as PC members/subreviewers might refuse a
task, e.g., due to conflicts of interest and/or lack of expertise.

**Role Management**: it allows the PC Chair to define, modify, reconfigure, cancel roles at any time during the OCS operation. These capabilities are very powerful, and they are responsible for our need of checking the rights at runtime. Fig. 1(top) shows the fine granular feature-based definition of the article management for the role PC Chair. These capabilities also exceed a typical RBAC role-based access implementation [15]: this way, there is no fixed role definition, in particular there is no role hierarchy: it is almost never the case that a role includes and/or monotonically extends the capabilities of "underlying" roles. On the contrary, different roles are prevalently orthogonal to each other in the sense that they bring different access rights.

**Setup Management**: it enables the responsible administrator and/or the PC Chair to configure the service before it goes public. It also allows online reconfigurations (e.g. setting global deadlines, assigning further responsibilities, establishing newsgroups) while the service is running.

As shown in Fig. 1, the features interact: by configuring differently the role PC Member (Feature Role Management, Fig. 1(top)) a PC Chair can at any moment grant and revoke coarse and fine granular access rights to the whole Article Management feature (or to portions of it) to single users or to user groups. We address the challenge to guarantee that these dynamically defined possible service behaviours obey all our requirements.

### 2.2 Property Description

Security and confidentiality precautions have been taken to ensure proper handling of privacy and intellectual property sensitive information. In particular,

- the service can be accessed only by registered users,
- users can freely register only for the role Author,
- the roles Reviewer, PC Member, PC Chair are sensitive, and conferred to users by the administrator only,
- users in sensitive roles are granted well-defined access rights to paper information,
- users in sensitive roles agree to treat all data they access within the service as confidential.

We need to be able to check these service-wide properties in a service architecture organized in a hierarchical feature structure. The following sections explain how the needs of the development of internet services like the OCS are taken care of by our application development environment.
3 Designing the OCS as a Feature Based System

As characteristic for the application development by means of our Agent Building Center (ABC) framework [11,18], the application definition layer is structured by means of a hierarchical use of features, which are realized in terms of components and (possibly nested) macros, each with its own SLG. Concretely, the ABC supports at the component level

• a basic granularity of components in term of SIBs which offer atomic functionalities and are organized in application-specific collections. These building blocks are identified on a functional basis, understandable to application experts, and usually encompass a number of ‘classical’ programming units (be they procedures, classes, modules, or functions).

• and a structuring mechanism via macros, which allow developers to build higher-order components that can be consistently reused as if they were basic components. Consistency is in fact a central concern in terms of analyzability and diagnosis via model checking - as explained in Sect. 3.4.

Application development consists then of the behaviour-oriented combination of SIBs and macros at a coarse-granular level.

The design of the OCS, a complex application whose SLG has currently approximately 2500 nodes and 3500 edges, reflects the typical feature-based organization of the application logic in the ABC [11]. As shown in Fig. 2, the global, application-level SLG is quite simple:

• it contains at the top level the logic for the service initialization (init-service) and the base service, which is a skeleton service that provides generic internet login and session management services, and the public functionalities (those accessible without being a registered user), and

• it coordinates the calls to and the interferences between the single features.

As we have seen in the feature description, features influence each other, thus one of the aims of service validation via model checking and testing is exactly the discovery and control of the so called feature interactions.

3.1 Feature-based Design

As shown in Fig. 2, each feature is implemented as a macro, thus it has an own Service Logic Graph that defines all the services and the behaviours possible under that feature. Fig. 3 shows, e.g. the SLG that implements the Article Management top-level feature. Top-level features typically provide a number of services to the users. In the case of the OCS, the depicted version offers in addition to the Article, Delegation, and Setup Management features already
briefly introduced in Sect. 2 also services for the management of Roles, Users, and Staff, as well as e.g. a feature for performing the PC Members’ Bidding for papers. This structure becomes immediately evident through the SLG, and it is also explicitly made publicly available over the GUI, as we can see in the navigation bar on the left side of the screen shots of Fig. 1.

Fig. A.1 (in Appendix) shows an excerpt of the features and subfeatures of the OCS. We see that several subfeatures occur in more than one feature, thus can be accessed under a variety of conditions. Altogether, the OCS has over 100 features. New releases of the OCS usually do not touch the basis service but involve the addition or major redesign of top-level features.

3.2 Hierarchy of Features

According to the needs of the application, features can be structured in finer-granular (sub-)features, which are themselves also implemented by means of SLGs. Similar to the structure at the application level, the SLG of the Article Management feature, shown in Fig. 3,
contains itself a workflow, here quite simple since it provides only navigation capabilities, and

coordinates the calls to and the interferences among a number of finer granular features, which can be themselves substructured according to the same mechanisms.
In our example, the Article Management feature deals both with the management of articles, as evident from subfeatures like SubmitArticle, ModifyArticle, SubmitFinalArticleVersion, and with article-related tasks that reside in other features, like Reportlist or DelegateArticle, which are part of the features Role and Delegation respectively.

To illustrate a complete top-down SLG-based refinement structure, we examine the SubmitArticle subfeature, reported in Fig. 5, which is technically again implemented as a macro. We reach in this SLG the refinement level where the actual business logic is described: embedded in the context of several checks and of error-handling logic,
(i) the ShowSubmitArticle SIB prepares and displays the webpage for the submission action,

(ii) ShowConfirmArticle allows the user to confirm the submission after checking the correctness of the metadata (like title, article, authors, abstract),

(iii) then the actual upload in the database and in the CVS versioning system is performed, and finally

(iv) the ShowSubmitArticleAcknowledgement SIB notifies the submitter of the successful execution.

The SLG also makes use of three macros, CVS_Checkin, mail_notification, and CreateNewsgroup (see Fig. 4). These macros embed reusable pieces of business logic which are relevant to the application designers, but not to the users. Accordingly, they do not deserve the status of a feature.

In the ABC, features are enabled and published to the end-users on their finer granularity, according to a complex, personalized role-right-context management. As an example, only users with a PC Chair role are able to submit articles in the name of another user. The design of the sub-structure of features is driven exactly by the needs of distinguishing behaviours according to different contexts. Sub-features in fact usually arise by refinement of features as a consequence of the refinement of the configuration features and of the role-rights management system. This way we enable a very precise fine-tuning of the access to sensitive information and to protected actions.

### 3.3 Organizing the User/Role Management

Once an internet service is online, it is continuously navigated in parallel by a cohort of agents that execute its global service logic on behalf of a user, within the limits imposed by the roles/rights of the user they are associated with.

The SLG of an application defines the space of potential behaviours that agents can assume, and each agent’s behaviour is defined implicitly as the currently valid projection onto this potential, filtered via

(i) the roles-and-rights management system, which defines dynamic, reconfigurable projections on the behaviours defined in the SLG, and

(ii) the current global status of the application, including the data space, the configuration, and certain event- and time-dependent permissions.

This has consequences on the design of the user and role management, and on its interaction with the feature-based model of the service functionality. From the point of view of the user and role management, features are seen as a collection of functionalities of the service which can be switched on and off.
for single roles and for single users. The service functionalities have unique names, whose naming scheme is quite simple:

\[ F-{\text{FeatureCategory}}-{\text{SubfeatureID}}.{\text{Filter}} \]

- The FeatureCategory is the name of a feature at the modelling level, implemented as an own SLG in the service,
- The SubfeatureID specifies a subfeature of the feature at the modelling level, that is implemented either as an own SLG in the service, or as a functionality of a SIB.
- The Filter suffix is optional and allows steering the fine granular right management: it restricts the access at runtime to the capabilities of the business objects underlying the features.

The user and role management are themselves implemented by means of features: Roles and Users, as seen in Fig. 1 are typically accessible to the Administrator and the PC Chair.

From the User/Role management’s point of view, the Article Management feature is itself managed in the FeatureCategory ART. The right to submit an article in the OCS is called permission F-ART-03: the single permissions of a FeatureCategory are numbered, thus uniquely named. In case of access to the subfeature SubmitArticle (see Fig. 1(top)), it is first checked whether the calling agent (implemented as a process) is granted the permission F-ART-03. Only then the access is allowed.

Some subfeatures, like the permission to read an article (F-ART-05), have finer granular variants which are administered through filters. The permission F-ART-05 says that the subservice that provides access to the content of a submission can be executed, but it does not specify on which articles. This is managed through filters, which distinguish the access only to the own articles (F-ART-05.own), only to the ones the user should review (F-ART-05.delegated) or to all the articles (F-ART-05.all).

This User/Role management mechanism exploits these fine granular permissions to create at need personalized views, limiting e.g. for a user the scope of access to certain resources (documents or functionalities). A role is defined via a set of permissions, and it is reconfigurable online at any time by users which have the corresponding rights on the feature Roles. This concerns the modification of the current roles, but also the definition of new roles (e.g. to deal with exceptional cases. An example of exception elegantly dealt with this way was the definition of a Substitute PC Chair role, where a PC member acted as PC Chair for articles submitted by the PC Chair to the conference he was chairing, which should obviously be treated completely independently. This way we grant a very high flexibility of the service usage.
3.4 Model Checking-Based High-Level Validation

The correctness and consistency of the application design enjoys fully automatic support: throughout the behavior-oriented development process, the ABC offers access to mechanisms for the verification of libraries of constraints by means of model checking. The model checker individually checks hundreds of typically very small and application- and purpose-specific constraints over the flow graph structure. This allows concise and comprehensible diagnostic information in case of a constraint violation since the feedback is provided on the SLG, i.e. at the application level rather than on the code.

The ABC contains an iterative model checker based on the techniques of [17], recently extended to a game based model checker [14]: it is optimized for dealing with the large numbers of constraints which are characteristic for our approach, in order to allow verification in real time. Concretely, the algorithm verifies whether a given model (a flattened SLG, where the hierarchy information in form of macros has been expanded) satisfies properties expressed in a user friendly, natural language-like macro language [13]. Internally, the logic is mapped to the modal mu-calculus with parameterized atomic propositions and modalities.

Example 1. The general OCS policies already mentioned in Sect. 3 as well as conference-specific policies inherently define a loose specification of the service at the service logic level, which can be directly formulated as properties of the OCS in our model checking logic. For example, the access control policy is a primary source of constraints like “A user can modify the defined roles only after having successfully registered as Administrator”, expressed as

\[-(\text{modify-roles}) \text{ unless } \text{user-login} \ [\text{Role=Admin}]\]

as a global constraint on the SLG of the whole application. This example illustrates the slightly indirect way of expressing the intended constraint. It says, “A user cannot modify the defined roles unless (s)he has successfully registered as Administrator”. Additionally the example shows a parameterized atomic proposition: \[\text{user-login} \ [\text{Role=Admin}]\] is parameterized in the possible roles a user might have, and \[\text{Role=Admin}\] does not only require a \text{user-login} to appear, but also that the role matches, in this case administrator.

All the properties mentioned earlier in Sect. 2 are requirements expressible in this logic, and they are instances of the classes of safety and consistency requirements identified in [1] to be characteristic of Computer Supported Collaborative Work platforms. Being able to automatically verify such properties via model checking is a clear advantage of the ABC, and it is essential in applications like the OCS where the role-dependency is much more dynamic.
than in standard RBAC applications.

A previous version of the OCS, which was not organized in features, had been already checked wrt. temporal logic properties like the one above \[19\] This global approach became impractical due to the growing size of the web service, to the increased importance of the Setup feature, which allows almost complete reconfigurability at any time, and to the transition to distributed development and maintenance, which are distributed feature-wise within a team of people. At this point, it became central to be able to partition also the verification feature-wise. This allows us e.g. to keep the properties readable, since we do not need to add large numbers of conjuncts just to isolate specific portions of the global graph, very often coincident with the features.

Meanwhile we use a slightly enhanced variant of CTL, where we have both forward and backward modalities. This is common e.g. in program analysis, and turns out to be useful also in our application. Examples of such operator pairs are $AF_F(\phi)$ and $AF_B(\phi)$, the well known always finally forward and backward CTL operators. We use often also until operators, useful to describe ”layered” regions of properties: $ASU_F(\phi, \psi)$ (resp. $AWU_F(\phi, \psi)$) mean $\phi$ strong forward-until $\psi$ (resp. $\phi$ weak forward-until or unless $\psi$). Thereby, the primed SIB names, like ’ShowFrameSetFiles, are the atomic propositions of the logic. Given the large alphabet of SIB and branch names it is convenient to use edge modalities with sets, as e.g. in $[\sim \{ok\}]\phi$, meaning that $\phi$ is true in each successor state reachable via an edge not labelled ok.

Apart from a number of simpler constraints that just enforce some forward or backward sequence of SIBs (useful e.g. in conjunction with macros, to enforce a certain well-formedness of reusal), most properties express reachability or a certain loose ordering of functionalities.

**Example 2.** In the ForgottenPwd feature, e.g., we would like that once the page with the form for answering the private question has been shown (done by the SIB ShowFrameSetFiles), the user-entered data should always be checked for correctness and completeness SIB CheckReqParam \(^3\). This is expressed as

$'ShowFrameSetFiles => [\{ok\}]AF_F('CheckReqParam')$

**Example 3.** Once this parameter check fails, the user should return to the page with the input form. The SIB CheckReqParam is in this case exited along the branch missing or exists_empty:

$'CheckReqParam => [\{missing,exists_empty\}]AF_F('ShowFrameSetFiles')$

**Example 4.** The password question should only be shown once a valid e-mail address has been input. The constraint

$'ShowPwdQuestion =>$
(AF_.B('CheckEmailAddr') ∧ AWU_.B(¬ CheckEmailAddr, ![successful]!T))

meaning that every occurrence of ShowPwdQuestion is preceded by a CheckEmailAddr) and that that CheckEmailAddr) has been exited along a successful branch. Here we rely on the uniqueness of the successful edge within the feature. In the general case we would need additional constraints like

\[ AG_\mathcal{F}(\langle \{\text{successful}\} > T = \Rightarrow \text{CheckEmailAddr}) \]

to delimit the scope more precisely.

**Example 5.** The notification page that an e-mail with the new password has been sent should not be shown before it was really sent out without an explicit acknowledgement by the user:

\[ \text{'Service2CallContext} = \Rightarrow ASU_\mathcal{F}(\text{'ShowPwdAck}, \text{'SendMimeMessage}) \]

Here we see that, as soon as the service logic becomes a bit more complex, the intuitiveness of the constraints is also quickly impaired: in order to check properties of the service logic, we need to refer to technical SIBs like Service2CallContext. We also see that sometimes the ”minimality” of constraints is not obvious: here we use until instead of next because in the graph there are self-loops.

An example of non satisfied constraints concerned the treatment of back browse branches in some areas of the OCS like the Report management feature, where several successive modifications of forms are possible in sequence. In order to check the existence and correctness of these (quite large) loops, we have decided to model the navigation structure of these OCS portions at the SLG level. However, due to the reusal of previously available subfeatures, some of the navigation options were still implemented at the GUI level, thus we were able to detect e.g. a number of missing back branches in the SLG. This was not a functional error, but an inconsistency in the modelling style.

4 Conclusions

We are not aware of any feature-based design approach similar in its intent to our goals, in particular concerning the simplicity at the modelling level. The closest approaches we know of typically require far more knowledge at the application level (at least programming expertise) and/or lack systematic support by means of formal methods, and therefore are inadequate for the scenarios and users we address.

The impact of our approach on the efficiency of the design and documentation has been proven dramatic in industrial application scenarios: our industrial partners reported a performance gain in time-to-market of the ap-
lications of a factor between 3 and 5. The reason for the reported gain was in particular the early error detection, due to the tightened involvement of the application expert into the development cycle. More generally, we see the current approach as an instance of Model Driven Application Development, where heterogeneous models allow the individual, but interdependent modelling of complementary aspects. And indeed, features constitute a specific category of such aspects, adequate for the structuring of complex applications according to complementary views and to support elegant and powerful approaches to proving correctness and compatibility of complex behaviours.

References


### A Hierarchical Feature Structure

<table>
<thead>
<tr>
<th>Main Features</th>
<th>Sub-Features</th>
<th>Article</th>
<th>Bidding</th>
<th>Delegation</th>
<th>Report</th>
<th>Tasklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article</td>
<td>SubstituteArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ModifyArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SubmitFinalArticleVersion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SubmitArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadDownloadArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DelegateArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reportlist</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SubmitReport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SubmitFinalReport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ModifyFinalReport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AcceptRejectReviewTask</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RemoveArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SendMailToAuthor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ShowAbstracts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bidding (List)</td>
<td>Bid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DisplayPdf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadDownloadArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadOnlyBidingMatrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EditBiddingMatrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DisplayPdf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DelegateArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bidding (Matrix)</td>
<td>DelegateArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DisPlayPdf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadReport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RemoveDelegation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadDownloadArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ExtendDecisionTaskDeadline</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delegation</td>
<td>DelegateArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DisPlayPdf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadReport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RemoveDelegation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadDownloadArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Email</td>
<td>MgmtEmailTextMacros</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MgmtEmailTemplates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ComposeAndSendMail</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report</td>
<td>ModifyFinalReport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ModifyReport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadReport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RemoveReport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadDownloadArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Role</td>
<td>DefineNewRole</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ModifyRole</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RemoveRole</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tasklist</td>
<td>SubmitReport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DelegateArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ReadDownloadArticle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SubmitReport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AcceptRejectReviewTask</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. A.1. Hierarchical Feature Structure and Feature Reusal