Modelling Management Components for Reuse using UML

Dave Lewis, Chris Malbon, Alina DaCruz

Department of Computer Science, University College London, U.K.
{dlewis, cmalbon, alina}@cs.ucl.ac.uk

Abstract. The competitive pressures of the telecoms sector are leading to a major push towards the automation and integration of many operational support processes. This creates pressures to develop more flexible and thus durable operational support systems within reduced system development time-scales. To respond effectively to these pressures, developers need to maximise software reuse and make effective use of modern software engineering tools and techniques. This paper presents an approach to modelling reusable components for telecoms management using the Unified Modelling Language. This approach aims to facilitate the selection of components and their rapid augmentation to meet management system requirements. The approach uses the concept of component façades, which are generated using CASE tools and published on the Web.

Introduction

The increase of competition in the telecoms market is a major motivating factor that is leading to the automation of operation support processes that manage networks and services in order to reduce costs, improve customer care and facilitate the rapid introduction of new services [1]. However the rapidly changing nature of the telecoms industry means that the requirements on management systems, especially at the service management level, are very volatile. The resulting requirement for flexibility and robustness in management systems, together with the need for rapid service development motivate the need for reuse techniques in service management system development. Reuse may take several different forms. It may involve the reuse of interface specifications, or reuse of an existing implementation either with or without modification (know as white-box and black-box reuse respectively). Traditional management standards such as the TMN recommendations or the OSI System Management Functions (SMF) have focused on specification reuse. Linking this with software reuse is left to the implementers of specific management platforms, e.g. HP OpenView, or to ad hoc solutions by management systems developers.

Currently, there is a range of different technologies regarded as suitable for different aspects of telecoms management [2], e.g. CMIP, SNMP, CORBA, HTTP, SQL etc, and a set of technology gateway solutions is evolving to aid the integration of these technologies, e.g. JIDM CORBA to CMIP gateways [3]. This however poses

severe problems to system analysts tasked with designing integrated management systems that span a range of technologies and are constructed from components obtained from different sources. Regardless of whether they are using specification or implementation reuse, analysts will be presenting with reusable solutions in a wide variety of documentation styles. These range from OSI SMF specification and accompanying GDMO definitions [4], through TM Forum Ensemble specifications [5] to TINA specifications using ODP viewpoints and the ODL interface definition language [6]. There is also wide range of proprietary commercial management component offerings. To be able to freely select, possibly modify, and then integrate solutions from these different sources, the management system analyst must have an in depth understanding of a wide range of notational and specification styles. This paper argues that for a truly open market to evolve in management components a common approach is needed for presenting management solutions to possible re-users. Such an approach must support both the representation of open interface standards and white-box reusable components. However to provide a clear migration path from the development of in-house components by management system developers to the purchasing of commercial off the shelf component, the approach must also be suitable for documenting bespoke component development. The approach is based on the widespread acceptance of the Unified Modelling Language (UML). This third generation object-oriented modelling notation has already been applied to management systems in the ACTS projects Prospect [7] and REFORM [8], with further case studies being carried out by EURESCOM in the P.610 project [9]. Working groups are also examining its adoption within the TeleManagement Forum and TINA-C. UML, however, only defines a modelling notation, not a methodological process. For this and for concepts on component modelling Ivar Jacobson's Object Oriented Software Engineering methodology has been used, and in particular the concept of component façades, the application of which to telecoms management we believe to be novel.

The next section of this paper describes how UML and OOSE have been used to define a suitable approach for modelling reusable management components for the ACTS project FlowThru. The paper then presents some experiences in applying this approach to the documentation of an existing subscription management component using the Paradigm Plus CASE tool and its resultant publication on the WWW. The final section outlines our conclusions and issues for further investigation.

Modelling for Component Reuse

A typical, well-structured system development process will start with the capture of system requirements, expressed as categorised requirement statements. This will usually be followed by a process of analysis where the requirements are more formally specified, for instance using object oriented modelling techniques. The resulting analysis model will then be fed into the design process where implementation and non-functional requirements such as operating speed and throughput will be accommodated to arrive at a design model that can be given to implementers to generate the software ready to be tested. In practice effective

software development requires a high degree of iteration back through these stages. Parts of the system that are perceived as risky or complex ideally are prioritised so that any unexpected issues discovered during their development can be fed back into rest of the system as early as possible. As changes can therefore occur concurrently at stages of modelling, it is essential that traceable links between the different models be maintained so that the wider impact of changes can be assessed and accommodated. For instance unexpected changes revealed in a subsystems design model will need to be traced back to its analysis model and requirements in order to assess the impact of the change on the subsystem, both internally and externally. At the analysis level, prior to the introduction of implementation details, interdependencies can be more easily spotted and addressed in order to achieve a more robust design. Thus design level activities can be more readily allocated to separate groups of designers without each group needing to have an in depth knowledge of other parts of the design model. These concerns are reflected Jacobson's Object Oriented Software Engineering (OOSE) methodology [10], which uses use cases at the requirements capture stage to drive the development of a system. Such a usage scenario-based approach to system development makes it applicable to the user-centric field of telecoms management, and has been successfully applied in management development projects P.610 and PREPARE [11]. The OOSE methodology addresses the robustness modelling of a system at the analysis stage by mapping use cases to three types of analysis objects. These types are:

- Entity Objects: Long-lived objects that represent the information content of the system. Typically they may be involved in several use cases instances.
- Boundary objects: Handle the communication between the system and its surroundings.
- Control Objects: Perform use case specific behaviour that is not related to boundary or entity objects.

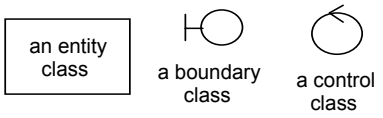

Fig. 1. Analysis Model stereotypes

The UML representation of these stereotype classes is suggested in [10] and was recently proposed to the OMG as an UML extension in [12]. A slightly different representation available with the Paradigm Plus CASE tool is shown in figure 1.

In OOSE the analysis model is refined into a design model, taking into account design level issues, such as scalability, load balancing, platform dependencies, database schemas etc. This mapping must be maintained so as to allow the evolution of objects from the analysis model to the design model to be traced. Interaction diagrams are often required at this stage to detail the behaviour and dynamic aspects of the relationships between the design objects. These may be more detailed than

interaction diagrams in the analysis model, which show how analysis objects collaborate to execute a use case.

Reusable components are typically presented to system developers as sets of libraries, i.e. as a set of software modules and the accompanying definition of the component's application programming interface. In terms of the above development process the component is therefore presented in terms of its design model and the software. This may cause problems in the development process, since changes required to accommodate the reuse of components are only likely to become apparent during the design modelling process, therefore possibly clashing with aspects of the system's analysis model. Often the component may be part of a framework. The framework may be general, e.g. CORBA Services, or aimed at supporting systems that solve problems in a particular problem domain, e.g. the TINA Service Architecture. Either way, the framework will provide some high level architectural and technological guidance on how components can be integrated together and how they can support the development of the system. Such frameworks are often considered at the analysis stage in order that the analysis model is structured in a way that accommodates the inclusion of the framework's components into the design model. This situation is depicted in figure 2a. However, frameworks typically only give general guidance on the use of components, and the suitability or otherwise of individual components in satisfying requirements still needs to be considered in the design modelling activity.

For telecommunication management systems development, such a typical component reuse situation is difficult to apply because there is no commonly accepted framework that supports a suitably wide range of components. The development guidelines for component reuse presented here are motivated by the absence of such a framework. Our approach was therefore to establish guidelines on how components can be specified in a more self-contained manner and in one that is easily understood by those performing the analysis of the system. In this way decisions about reuse can be made based on the suitability of individual components rather than a wider assessment of the suitability of an entire framework. The approach also aims to support decision-making based on architectural and functional aspects of a component rather than its technology. The technology is treated as an orthogonal issue, with heterogeneity handled primarily through the employment of suitable gateways.

The approach is derived from that described in [13] in which it is proposed that components are not presented just as units of design and software, but should be packaged together with the analysis model of the individual component, rather than being strongly integrated into a specific component framework. In addition, if the modelling techniques used for the analysis model of the component are similar to that used for the modelling of the system in which they might be included, then it is much easier to assess whether the component is suitable for use in the system. In this case the system's analysis model can directly use the analysis model abstractions from the components it reuses, easing the task of requirements analysis, and ensuring at an early stage compatibility between components and the system requirements.

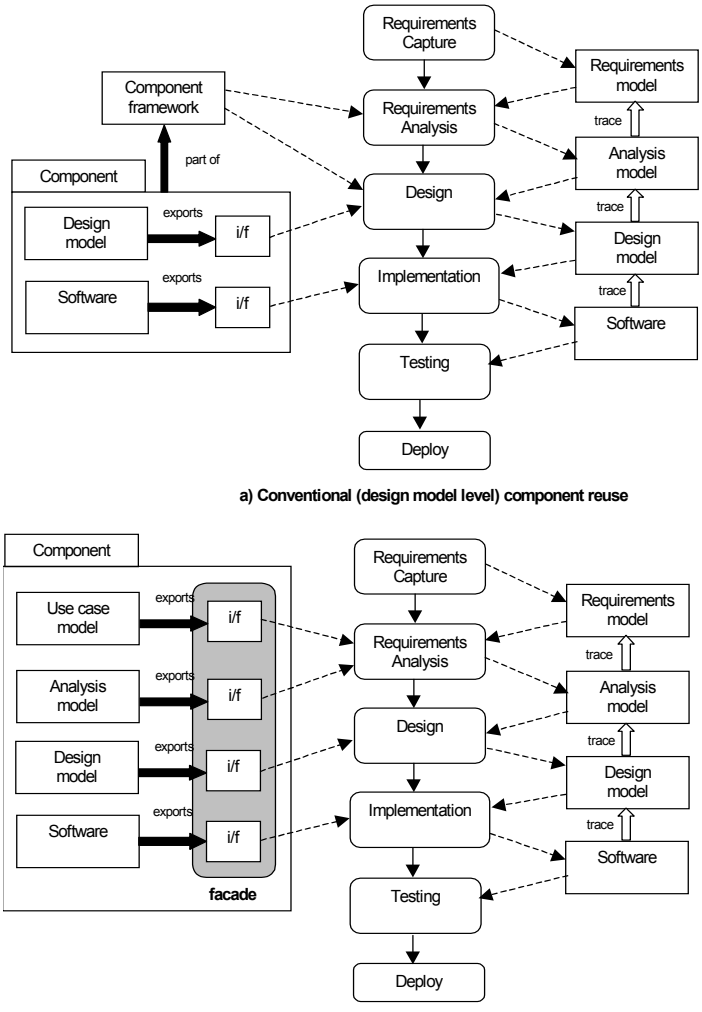

Fig. 2. Different Approaches to Component Reuse

The presentation of a component for reuse is known as a façade. A façade presents the re-user of a component only with the information needed for reuse, while at the same time hiding from the re-user unnecessary design and implementation details about the component. In the analysis model-based reuse approach the façade consists not just of reusable code and the design model details needed for reuse, but also the analysis model relevant to the design model part of the façade. This analysis model-based reuse approach is depicted in figure 2b.

A component may present several different façades, possibly aimed at different types of re-users, e.g. black-box or white-box re-users. A component may have various releases of a façade to reflect the evolution of the component. The usefulness of the façade is strengthened if there is clear traceability between the different models within the façade so that re-users can easily determine which aspects of the component are useful to them by tracing down from the analysis model level.

The construction of a façade from the internal models that resulted from the component's development process will be greatly eased if the same type of modelling approach was used for this process as in the façade. Strong traceability between the components internal development models will ease the selection of parts of the model for inclusion in the façade based on the selection of use cases for the façade. However, it is not a requirement for the component to have been originally developed and documented using an OOSE-like process, and part of the benefit of façades should be that it can hide the internal development model if necessary.

Experience with Approach

The approach outlined above is part of a wider management system development framework that has been defined in the ACTS project FlowThru [14]. This development framework specifies a development methodology for constructing systems that satisfy business process requirements from reusable components [15]. This methodology has been applied within FlowThru to the analysis of three separate Trial Business Systems demonstrating different business process areas based on the TM Forum's Telecoms Operations Map [16]. The components from which these systems are being constructed are from different ACTS projects, and as they were originally developed using a variety of notational systems. To validate the project's development framework their specifications had to be recast as façades.

The experiences presented in this section are from the development of a façade for a subscription management component that was used successfully in different applications within the Prospect project [17]. This component was based on the subscription management specification in the TINA Service Architecture [18]. This specification was structured in terms of ODP viewpoints, principally the information and computational viewpoints. This specification did not provide sufficient motivation for many of its design decisions, thus making augmenting the specification an uncertain task. The component was presented only as a design model with little in accompanying requirement statements and no explicit linking between the two. This is fairly typical of TINA specifications, and stems from a lack of a clear mapping between the enterprise, information and computational viewpoints used in the ODP approach. In generating the façade for this component, it was therefore decided to develop the requirements and analysis model more or less from scratch and then attempt to map it onto the existing design specification.

The aims in generating this façade can therefore be summarised as:

- To provide a mechanism through which a potential re-users can gain a modular view of the functions the component provides (the use case model), from which

they can quickly trace via the analysis model to the specific component interfaces that they might be interested, e.g. IDL interfaces and operations.

- To provide a mechanism for checking the impact on the component's analysis model and design model of any changes in component requirements, expressed as use case modifications, i.e. in supporting white-box reuse
- To provide a mechanism for checking the impact of any design level changes required of the component, by tracing the design modification back to effected analysis objects and then to other impacted design objects.

The general approach was to generate a consistent set of use cases, analysis model diagrams and design model diagrams. Use cases were written in a word processor that was able to generate HTML output. The analysis and design models were generated on the Paradigm Plus CASE tool as UML diagrams with additional relationships added for links between specific UML classes in different diagrams. Paradigm Plus was able to generate HTML of specific diagrams and for each class that would generate a summary of its operations and its aggregations with and associations to other classes. Paradigm Plus uses an object-oriented database for storing its UML models and contains a simple scripting language for querying that database. These features were exploited in order to augment the HTML generated by Paradigm Plus with the additional links needed to be able to trace between the analysis and design models. Though Paradigm Plus itself allowed for hyper-textual style links between different diagrams, this was not regarded as a good solution for actually publishing the component, whether this is done in the public domain or within a software development organisation. This was partly due to the cost involved in having Paradigm Plus available to anyone who might want to browse the model, it also allowed for the possibility of a similarly structured façade being generated by other tools, i.e. the approach is not tied to an particular CASE product. The decision to use HTML was also motivated by the assumption that such a CASE tool would be typically used for modelling all aspects of a component, up to and including code generation. Therefore a separate HTML view provided a secure way of publishing only the information needed in the façade for the benefit of re-users, while filtering out other modelling information that the reuse didn't require and which may erode the components developer's implementation investment if in the public domain.

Use Case Modelling

The use case model is the entry point for potential re-users of the component in using the façade to make component selection decisions. The use cases describe the component in terms of its interaction with a set of external actors.

We used use cases to provide three important façade functions:

- To record the system requirements
- As the main input to enable us to create the analysis model
- To guide final system verification once the component had been delivered

We began by examining the actors who interact with the component across its system boundary. This yielded a number of interactions from the point of view of the users of the system. These interactions served as starting point for use cases, where the users where modelled as actors. If a process can be described from the point of

view of more than one actor a new use case was written. Although this resulted in many more use cases, we felt the resulting clarity is well worth the trade-off. It enabled designers to concentrate on recording each process from a unique viewpoint and avoided writing unnecessary and confusing exclusion clauses into a use case to cope with a particular actor's requirements.

The first pass analysis often indicated other, preparatory use cases. For example, in our subscription component a major use case addresses subscribing a customer to a service. This in turn yielded two more use cases: creating a service to which a customer can subscribe, and creating the customer's account in the first place.

Once a number of use cases had been identified the use case descriptions were documented. We found it useful to do this within a structured layout using a label to uniquely identify the use case, numbered pre- and post- conditions, and the description of the use case itself. Each description was originally written using a "conversational" style, but we quickly moved to what [19] calls an "expanded" style, whereby the typical sequence of events between an actor and the component were recorded. Each sequence was then numbered. Structuring our use cases in this way, with their pre- and post- conditions and event sequences indexed, was important as it enabled us to build up a set of cross-referenced tables. By hyper-linking the document we can guide the re-user more easily through it, enabling them to gain a quicker understanding of the component and it's functionality. Pre-conditions could be referenced back to earlier post-conditions; and use case descriptions referenced back to earlier use cases such that only their extensions needed recording. This enabled us, textually, to map the use case concepts of "uses" and "extends".

Two important use case types, often overlooked, were also included; system initialisation and shutdown. These, respectively, manage the process of making the component ready to commence operation and, when requested, cleanly closing it down. As this often involves interactions other actors and system actors, the sequence of events with which this is done must be captured.

It is important to remember that as each use case description is refined there is a temptation to modify the description so that it incorporates more and more of the identified analysis objects. Doing so can result in moving the description out of the requirement domain and into the analysis domain. Pitching the descriptions at a level that those uninvolved in the component's development can understand prevents the descriptions becoming too abstract or from being used merely as annotation to the accompanying analysis model diagrams.

Analysis Modelling

Using noun-phrase selection, an initial pass of each use case description yielded the major entity objects and their relationships. The use cases also identified a boundary object for each interaction between the system and each external actor. Control objects are required whenever there is a need to create, destroy, update or otherwise manage an entity object or closely linked group of such objects.

Our example producing entity objects such as Service, Customer Account, Service Level Agreement (SLA), Subscription and Subscription Contract, a set of boundary

objects, and four major control objects: Service Manager, Customer Manager, Subscription Manager and SUG Manager.

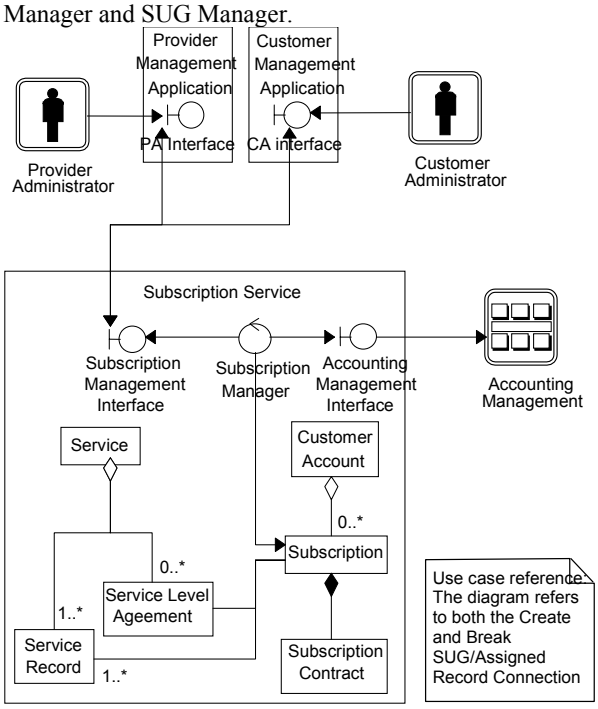

Fig. 3. Analysis Object Diagram for Subscribe a customer to a service use case

The objects were then modelled as analysis class stereotypes in UML use case diagrams since was the only diagram type in which Paradigm Plus rendered the analysis objects stereotypes. As use diagrams are more commonly used to show how use cases relate to each other using the "extends" and "uses" relationships and to actors across the system boundary we called these static analysis model diagrams, analysis object diagrams (see Figure 3 for an example). One analysis object diagram was created for each use case, though on a second pass it was found closely related use cases could be supported by a single analysis object diagram. This level of mapping aided in the clear navigation from use cases, via analysis object diagrams to design objects.

Creating the analysis object diagrams threw up issues that were not expressed in the use case descriptions, and thus needed further consideration. For example, in the subscription component the precise relationship between a customer subscription and a SLA was unclear. Could a subscription have multiple SLAs, or should we restrict each subscription contract to one? Once these issues are resolved they are fed back into the use case descriptions to ensure there is no inconsistencies between the use case descriptions and analysis models. In order to produce a cross-referenced, top-level view that acts as a reference for all analysis model diagrams, a consolidated

analysis object diagram is produced showing all interface objects, control objects, and entity objects that they manage.

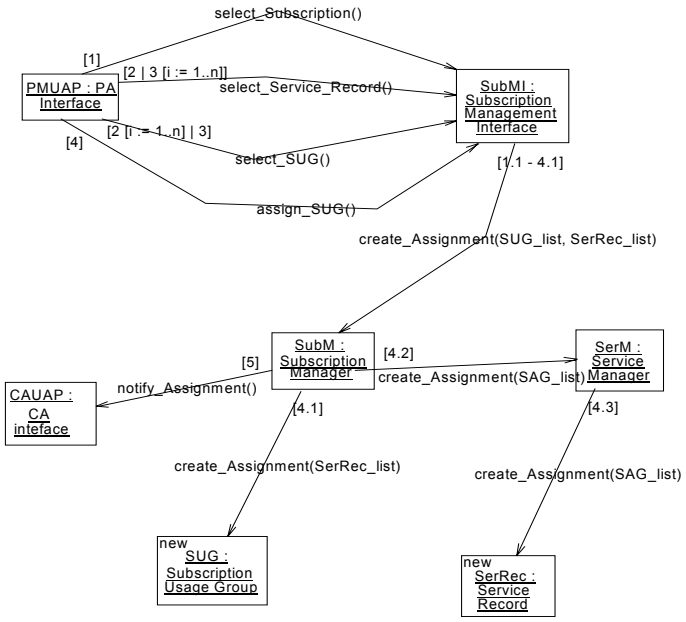

Fig. 4. Collaboration Diagram - Create Service Usage Group/Assigned Service Record Connections

Much of the analysis model can be express in this way. However, there are occasions when analysis object diagrams alone are insufficient, in particular, when two managed objects needed to establish a link, and when the status of an object must change. Here, we use collaboration and sequence diagrams in addition to the analysis object diagrams to show dynamic requirements (Figure 4 gives an example of the former). Both of these diagrams were used in a similar manner, and show the event sequence needed to fulfil a use case. Here, we were interested in the messages sent between objects, rather than their relationships with each other. It is these operations, lifted from the use case description and outside of the normal create, delete and query interfaces provided by a control object, that gave the additional information required to express analysis object behaviour and the set of operations it required. Collaboration diagrams are preferred over sequence diagrams when we are less concerned with message synchronicity, however, sequence diagrams can be clearer if many interactions are being modelled.

Design Modelling

For our subscription management component the design and implementation already existed, and therefore, we were involved in a process re-documenting the design. In being able to create a link between the analysis and design models we hoped to:

- Improve understanding by allowing a re-user to see from where an object had been derived, and ease the production of a navigable façade in HTML,
- Improve robustness as the impact on change to the component could be traced through the varying models of the façade,
- Shed light on the analysis to design mapping by commenting on the links we created.

We chose to create a link between an analysis and design object using the UML Depends-On relationship and augment this with the label "linkage" to distinguish it from other Depends-On relationships that might exist for that object. The analysis to design mapping produced several one-to-many relationships, as abstract concepts in the analysis model are decomposed into an increasing number of design objects. A smaller group of many-to-one relationships also exist. This occurred when distinct entity objects were identified from the use case descriptions but their characteristics were similar enough to justify a common, possibly abstract, design object.

There was a clear mapping between control objects in the analysis model and computational objects already defined in the design. Boundary objects (e.g. Subscription Management) mapped to individual initialisation, query, and management interfaces in the design model, which in turn mapped to IDL interfaces in the implementation.

The analysis process failed to identify some objects and relationships present in the existing design. For example, the analysis model contained no Subscription Portfolio object, a collective holder for a number of customer subscriptions. Nor did it model the complex association between a Service Record and a Subscription Usage Group. Both represent the designer's skill in being able to harness past experience to recognise suitable design patterns and take into account non-functional requirements such as performance issues, caching requirements, etc.

Publishing the Façade

We wished to translate our façade into HTML so that the component could be advertised to the widest possible audience. We used Paradigm Plus to convert each diagram into HTML page, but were unable to use the tool to create hyper-links between objects that represented the Depends-On links. For instance, the SLA object in the analysis model mapped onto three objects in the design model. Our aim was for a re-user presented with a diagram containing a SLA object should be able to click on the analysis representation which would then present them with the list of design objects to which they map. Clicking on any object in this list would take the re-user to its design object page and from there to any relevant IDL definitions. We therefore wrote a Paradigm Plus script to search for the Depend-On relationships between analysis and design objects to generate a linkage table that we had saved to file. Then, by processing this file we created an HTML table containing all links between analysis and design objects. We then modified the Paradigm Plus generated HTML pages to include links to this table. At the time of writing the façade for the Subscription Management component was available on the WWW at [20].

Conclusions and Further Work

This paper proposes an approach to presenting reusable components through façades that bundle high level functional statements, an analysis model and a design model together for every component. Such a structure is expected to ease the selection and integration of the component by re-users. In addition its compatibility with a well-tried development approach (OOSE) together with traceable links between the models aids white-box reuse, where the component must be modified for the task at hand.

The experiences presented in this paper relate to the generation of the façade using UML and its publication via the WWW. It was found that the division between the analysis model and the design model provided a good basis for drawing the line between exposing the internal details of a component needed for its features and capabilities to be understood, while hiding all detailed design issues except those relating to the interfaces via which it is re-used.

Publishing the model in HTML with a structure suitable for re-users to make best use of the façade was found to be relatively simple with Paradigm Plus. This tools did not however allow modification of the HTML generation function to suit our needs, so additional scripts had to be written to extract information from the tool's object repository in order to augment the generated HTML with the additional traceability links. A better solution would be to have more control over the built in HTML generation function.

This work opens many avenues of investigation in working towards an open market in easily re-usable management components. Directions that will come under investigation in the FlowThru project are:

- Jacobson describes the inclusion of variability points into a façade, this may provide a good common mechanism for identifying the variability mechanisms that are available with many component technologies such the emerging CORBA Component [21] standard and Enterprise Java Beans [22].
- Some assessment of how easy it is for re-users to comprehend component façades and assess the impact of required white-box modifications.
- The UML representation of components and their interfaces needs to be better supported in other modelling views such as interaction diagrams.
- To integrate management components to the analysis of a business problem, a better understanding is required of the mapping between business process re-engineering techniques, for instance UML activity diagrams [23], and the features offered by façades.

  The use of an underlying object repository in the case tool used to generate the façade presents the possibility of also generating complexity metrics for the component. Metrics useful to the re-user of a component needs to be assessed.
- Promoting tool support for the façade construct requires a more formal structural definition. A UML definition of the façade structure is under development, with an XML version planned in order to support façade interchange and processing by CASE tools.

Acknowledgements

The work presented in the paper was conducted with partial funding of the European Commission under the FlowThru project (AC335). The views expressed here are not necessarily those of the FlowThru consortium.

References

1. E. Adams, K. Willetts, The Lean Communications Provider: Surviving the Shakeout though Service Management Excellence, McGraw-Hill, 1996.
2. NMF Technology Map, Draft NMF GB909, July 1998.
3. Inter-Domain Management Specifications: Specification Translation, X/Open Preliminary Specification, X/Open, Reading, Draft of April 17, 1995.
4. TMN Interface Specification Methodology, ITU_T Draft Revised Recommendation M.3020, 1994.
5. the Ensemble Concepts and Format, NMF-025, issue 1.0, Network Management Forum, Morristown, NJ, 1992.
6. Salleros, J., TINA-C Service Design Guidelines, TINA Report, TP_JS_001_0.1_95, TINA Consortium, March 1995.
7. Lewis, D., A Development Framework for Open Management Systems, To be published in a special issues of the Icon Journal.
8. Kande, M., Mazaher, S., Prnjat, O., Sack, L., Wittig, M., Applying UML to Design an Inter-Domain Service Management Application, Proceeding of UML'98, Mulhouse, France, June 1998.
9. Nesbitt, F., Counihan, T., Hickie, J., The EURESCOM P.610 Project: Providing a Framework, Architecture and Methodology for Multimedia Service Management, Proceeding of 5th International conference on Intelligence in Service and Networks, Antwerp, Belgium, Springer-Verlag, 1998.
10. Jacobson, I., Christerson M., Jonsson P., and Overgaard G.,, Objected-oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992.
11. Modelling and Implementing TMN-based Multi-domain Management, PREPARE consortium, editor J. Hall, Springer-Verlag, 1996.
12. UML Extension for Objectory Process for Software Engineering, version 1.1, September 1997.
13. Jacobson I, Griss M., and Jonsson P., Software Reuse: Architecture, Process and Organisation for Business Success, ACM Press Addison Wesley Longman, 1997.
14. Lewis, D., Wade, V., Bracht, R., The Development of integrated Inter and Intra Domain Management Services, to be published in the proceedings of IFIP/IEEE Integrated Management'99 conference, Chapman-Hall, May 1999.
15. Initial guidelines for System Design and component Reuse, FlowThru deliverable D2, Oct. 1998
16. NMF Telecoms Operations Map, NMF GB910, Stable Draft 0.2b, April 1998
17. Lewis, D., Tiropanis, T., McEwan, A., Redmon, C., Wade, V., Bracht, R., Inter-Domain Integration of Services and Service Management, Intelligence in Services and Networks: Technology for Cooperative Competition, Proceedings of IS&N'99, Cernobbio, Italy, Springer-Verlag,May 1997.
18. Service Architecture, ed. L. Kristiansen, TINA-C, version 5.0, 1997.
19. Larman, C, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, Prentice Hall PTR, 1998.

20. http://www.cs.ucl.ac.uk/research/flowthru/models/fulfilment/subscription
21. CORBA Components: Joint Initial Submission. OMG TC Document orbos/97-11-24, Draft, November 1997.
22. Thomas, A., Enterprise JavaBeans: Server Component Model for Java, White paper, http://java.sun.com, December 1997.
23. Allweyer, T., Loos, P., Process Orientation in UML through Integration of Event-Driven Process Chains, Proceedings of UML'98, Mulhouse, France, June 1998.