WORK PRODUCT-DRIVEN SOFTWARE DEVELOPMENT METHODOLOGY IMPROVEMENT

Paul Bogg, Graham Low
School of Information Systems, Technology and Management, University of New South Wale, Sydney, NSW 2052, Australia

Brian Henderson-Sellers
School of Software, University of Technology of Sydney, PO Box 123 Broadway, Sydney, NSW 2007, Australia

Ghassan Beydoun
School of Information Systems and Technology, University of Wollongong, Wollongong, NSW 2522, Australia

Keywords: Software Development Methodologies, Method Engineering, Software Process Improvement, MOBMAS.

Abstract: A work product is a tangible artifact used during a software development project; for example, a requirements specifications or class model diagram. Towards a general approach for evaluating and potentially improving the quality of methodologies, this paper proposes utilizing a work product-based approach to method construction known as the "work product pool" approach to situational method engineering to accomplish this quality improvement. Starting from the final software application and identifying work product pre-requisites by working backwards through the methodology process, work product inter-dependencies are revealed. Using method fragments from a specific methodology (here, MOBMAS), we use this backward chaining approach to effectively recreate that methodology. Evaluation of the artificially recreated methodology allows the identification of missing and/or extraneous method elements and where process steps could be improved.

1 INTRODUCTION

Methodologies for software development involve key elements of user roles, tasks they perform and work products they produce and consume. In the development stage of a methodology, either for general, widespread use (e.g. as published in text books) or for a specific organization, the traditional approach has been to focus initially on the work that is done (often given labels such as tasks, activities or processes). Only secondarily is consideration given to the work products that are either produced or consumed. [A *work product* is a tangible artefact used during a software development project, for instance, requirements specifications, class model diagrams, and use case specifications.] In other words, the focus is on a methodology as a series of connected transformation engines where each of these transformation engines has one or more input work products and one or more output work products. The network of work units and work products will ultimately lead to a work product that is the final application but it does not guarantee that there will be no cul de sacs i.e. by identifying first a work unit in the early stages of the methodology's lifecycle, then taking the output (a work product) of this work unit and making it the input of a second work unit – and so on – we derive a network of work units and work products that may culminate in a work product that is of no value whatsoever. In other words, this is a Taylorian approach by which methodology is equivalent to a series of work units that may include several non-essential elements.

As an alternative, arguing that the most important element of software development is the final application (a work product), we can invert the argument and ensure that our final work product is the target software application by working backwards from this clearly necessary final work

Bogg P., Low G., Henderson-Sellers B. and Beydoun G. (2010).
WORK PRODUCT-DRIVEN SOFTWARE DEVELOPMENT METHODOLOGY IMPROVEMENT.
In *Proceedings of the 5th International Conference on Software and Data Technologies*, pages 5-13
DOI: 10.5220/0002898200050013

product to ask what work products are prerequisites to it and then, and only then, what are the work units (tasks acting as transformation engines, for example) that will link this prerequisite work product (input) to the output work product. Having established this work product and work unit, we continue chaining backwards until the initial requirements statements are identified. This is the so-called "work product pool" (WPP) approach, recently argued (Gonzalez-Perez and Henderson-Sellers, 2008) to provide a higher quality methodology – in the sense of containing no redundant elements.

The WPP approach supports methodology creation from method fragments (Harmsen et al., 1994) within the auspices of the situational method engineering (SME) paradigm (e.g. Ralyté et al., 2007). SME is the engineering discipline aimed at creating a methodology from the contents of a previously constructed methodbase (Saeki et al., 1993) specifically attuned to the project (or set of projects) at hand. Furthermore, SME has a flexibility that can be capitalized upon in the context of software process improvement (Henderson-Sellers et al., 2007). The analysis presented here of our case study methodology can be regarded in the same context of software process improvement (SPI) although, generally, methods for SPI are process-oriented rather than work product focussed and have a wider scope for potential improvements than those discussed here i.e. our approach makes a minor contribution to SPI.

Although the WPP approach can be argued to be generally applied in all kinds of software development (and perhaps, arguably, to a wider range of human-intensive methods than just software), in this paper we make a choice – and investigate its application in a case study of a single agent-oriented software engineering (AOSE) methodology: MOBMAS (Methodology for Ontology-Based Multi-Agent Systems Development: Tran et al., 2006; Tran and Low, 2008). MOBMAS has been constructed with the traditional mindset of retaining a focus on work units as transformation engines. Here, we investigate, firstly, whether the WPP approach can be successfully used in re-constructing MOBMAS using backward chaining rather than the forward chaining implicit in the methodology descriptions in the MOBMAS literature and, secondly, whether the WPP approach can give added insights and perhaps improvements to MOBMAS. For example, a backward chained, reconstructed MOBMAS may have additional work products or we may be able to identify unnecessary work products as postulated by Tran and Low (2008) but which the WPP approach identifies as having little or no value in creating the final software application work product.

In the next section, we outline the overall background to software development methodology creation followed by an evaluation of how a work-product-driven approach may be applied to the issue of methodology improvement and possible improvement. Section 4 then applies the general arguments of Section 3 to the chosen MOBMAS methodology. Final discussions and conclusions are to be found in Section 5.

2 WORK PRODUCT DRIVEN METHODOLOGY EVALUATION AND SPI

When members of a software development team use a methodology, they need to be assured of its quality, particularly in terms of completeness and, conversely, in ensuring that they do not undertake any tasks that provide no real value. Here, we apply the WPP approach, as outlined above, as an evaluative tool for existing software development methodologies. In this section we describe the way in which quality assessment could be undertaken and, in Section 4, we illustrate this with a case study of one specific methodology from the literature – MOBMAS (Tran and Low, 2008) as being one that is familiar to us.

The decision to undertake this methodology evaluation could be either as a simple quality check for a newly constructed methodology or because there is some indication that the methodology in use either lacks elements or contains extraneous elements. Whatever the reason for deciding to undertake a methodology assessment, the result of the assessment will have the same format: recommendations for "improvements", which may loosely be called SPI.

We propose a four step assessment process:

1. Work Product elicitation – identifying work product types, and relationships (inter-dependencies) between work product types: both then stored as method fragments in a methodology-specific repository or methodbase (Saeki, *et al.*, 1993).
2. Using the backward chaining approach of the WPP approach, and using only method fragments (identified in step 1) from the methodology-under-investigation, recreate the target methodology. This we will call the "WPP-

reconstituted methodology" or WPPM.

3. Analyze the WPPM for missing elements or for work product types that are outputs only (using the rule that all work products output during a development must necessarily also be inputs to some other process element – excepting the final deliverable software of course).
4. Draw up a document making suggestions for revising the methodology for presentation to decision-makers such as in-house methodologist, CTO, project manager.

These steps are detailed in the following.

2.1 Step 1: Work Product Type Elicitation and Specification

Work product type elicitation begins by considering (a model of) a software application. Then, by working backwards through the methodology process, work product pre-requisites are identified. The elicitation of work products enables identification of the process steps required to produce them and is guided by definitions for four key methodology components:

- *Work Product type* – specification of an artefact of interest, to be of use for a methodology instance
- *Work Product Group* – a group of closely interrelated work product types
- *Task type* – specification of a small-grained job performed within a methodology instance, including production of work products
- *Process steps* – a collection of task types that produce a work product type

The elicitation of work product types, work product groups, task types and process steps is the basis upon which evaluations (in Step 3) are conducted on the methodology as a whole. For each *Work Product type*, we then identify the following:

- *Work Product type* dependencies between pairs of work product types. There are two types:
  - Production dependencies – where this work product type is (partially) derived from another work product type.
  - Verification dependencies – where a work product type specifies retrospective verification with another work product type. Verification might mandate further revision to work product types to bring about consistency. Consistency is important for ensuring the integrity of the development process (i.e. requirements are addressed with analysis, and ultimately in design). Verification may require that work products are mutually revised to ensure consistency.

Consequently we can identify:

- *Task types* that produce the work product types
- *Work Product Groups* – where work product types are closely related by similarity

Work product types are identified from the published literature on the target methodology. Each work product type is defined, and: production and verification dependencies identified.

Once these model fragments have been identified in isolation and stored in a methodology-specific methodbase, in Step 2 we now use them to reconstruct the methodology following the WPP approach for method construction.

2.2 Step 2: Develop WPPM

In Step 2, the WPPM is constructed from the work product types, connected using task types. The WPPM may make use of *work product groups* to provide a higher-level overview, which is particularly beneficial when making global, project wide comparisons to a methodology.

Conducting the reconstruction process may identify methodology problems when:

- Work products are redundant;
- Work product dependencies are missing or unnecessary;
- Work products are introduced at an inappropriate point in the process

These problems may mandate methodology revision if they cause inefficiencies for project contexts. Where one or more of these problems arise, further analysis can determine how they should be resolved. Suggestions on how to address the problems are discussed in Step 3 and formalized in Step 4.

2.3 Step 3: Analyze the WPPM

The aim of this step is to detail possible areas for improvement, elaborating on those areas identified in Step 2 as potentially problematical and/or identify new concerns. Two types of analysis are:

i) Identifying whether work products can be developed in parallel
ii) Identifying types of skill sets necessary for enacting the methodology

Where traditional software development processes are enacted in a linear order, recommendations for methodology improvement may be made to improve coordination of software

development. For i), identifying that two work product groups can be developed in parallel might be beneficial for streamlining software development. For ii), identifying skill sets might be beneficial for understanding in what parts of the development process particular skills are needed. Management recommendations are proposed amendments to the methodology that may improve the efficiency of methodology enactments (Step 4).

2.4 Step 4: Document and Formalize the Recommended Improvements to the Methodology

Where problems have been identified in Steps 2 and 3, methodology revisions might be needed. It is recommended that further analysis is made to understand the significance of problems identified. For significant problems, the following heuristics may be used to formalize problems:

Where a Work Product type has no dependencies of any kind, it is removed

- Where a Work Product type has missing dependencies, then add a task type connection
- Where a Work Product type has unnecessary dependencies, remove a task type connection
- Where two similar Work Product types have the same dependencies to other work products, attempt to identify whether the two Work Product typescanb e consolidated to one.
- Where Work Product types have been introduced at inappropriate points, use the dependencies to work out the best place to introduce them in the methodology process

There are notable exceptions to these heuristics. For instance, initial work products will not have dependencies and may not need to be removed. Ultimately, revision to the methodology should only be undertaken when it is certain that the revision will lead to an improvement to the methodology as a whole.

3 REVISING MOBMAS

The application of this WPP-driven revision of a methodology is further illustrated by one specific case study – the analysis of MOBMAS.

In MOBMAS, the MAS development starts with a domain ontology, used initially to identify goals and roles of the system to index an appropriate set of problem solving capabilities from an appropriate existing library of capabilities. Individual ontologies corresponding to the knowledge requirements of each capability are then extracted from the initial common ontology to provide knowledge representation and allow reasoning by individual agents. Those ontologies form the basis for an iterative process to develop a common communication ontology between all agents and verify the knowledge requirements of chosen capabilities. Individual, localised ontologies may require incremental refinement during the iterative process. Appropriate ontology mappings are needed between local ontologies and the communication ontology. The development of an MAS using MOBMAS has five activities (Figure 1). Each focuses on one of the following key area of MAS development: Analysis, Organization Design, Agent Internal Design, Agent Interaction Design and Architecture Design.

Analysis Activity: This aims to form a conception of the target MAS from the domain ontology and the system requirements, giving a first-cut identification of the roles and tasks that compose the MAS. It consists of developing the following five models: System Task Model, Organizational Context Model, Role model, Ontology Model as well as identification of Ontology-Management Role. The Role Model is developed in a highly iterative manner with the System Task Model, given the association between roles, role tasks and system tasks. The Ontology Model is used to refine and validate those models (and vice versa). This activity also specifies the ontological mappings between the MAS Application Ontologies.

MAS Organization Design: This refines the organizational structure of the target MAS and identifies a set of agent classes composing the system. If the MAS is a heterogeneous system that contains non-agent resources, these are also identified and their applications are conceptualized. It consists of the following four steps: Specify the MAS Organizational Structure, Develop the Agent Class Model; Develop the Resource Model; and Refine the Ontology Model of the previous activity. The developer also specifies the mappings between Resource Application Ontologies and relevant MAS Application Ontologies, to enable the integration of these resources into the MAS application and to support the interoperability between heterogeneous resources.

Agent Internal Design: For each agent class, this activity specifies belief conceptualization, agent goals, events, plan templates and reactive rules. It consists of the following five steps: Specify Agent Class' Belief Conceptualization identifying which

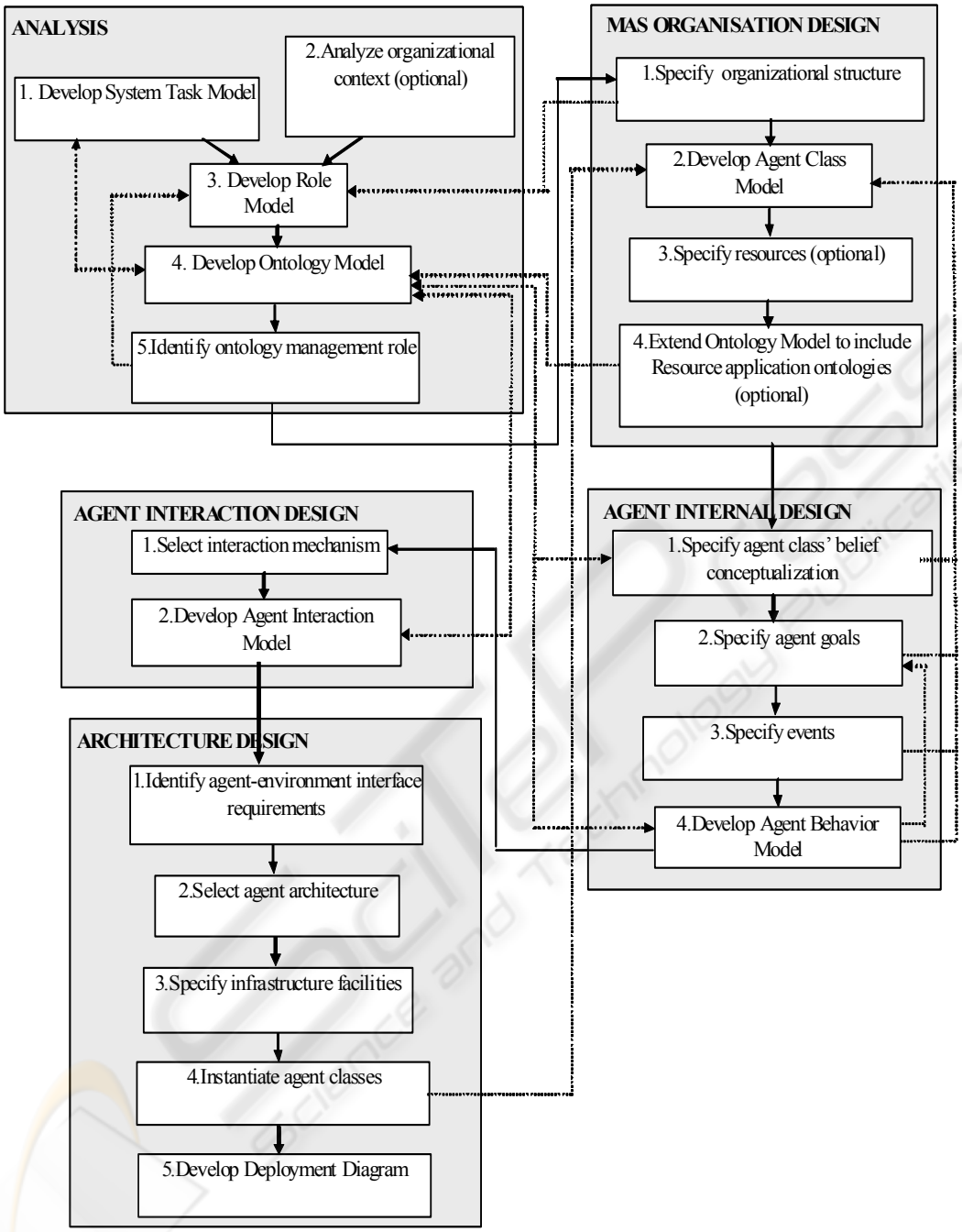

Figure 1: MOBMAS development process**:** Solid arrows represent the flow of steps within and across activities; dotted arrows indicate potential iterative cycles of steps. Models produced or refined by each step are shown in square brackets.

part(s) of the Ontology Model are needed by an agent class to conceptualize its run-time beliefs; Specify Agent Goals identifying the states of the world that an agent class aims to achieve or satisfy using the Role Model; Specify Events in the environment that agents need to respond to at run-time; Develop Agent Behaviour Model specifying how each agent class behaves to achieve or satisfy each agent goal as planning behaviour or reactive behaviour; and Update the Agent Class Diagram with the details identified in the previous three steps. The Agent Behaviour Model is checked for consistency against the Ontology Model and vice versa.

Agent Interaction Design: This models the interactions between agent instances, by selecting a suitable interaction mechanism for the target MAS and modelling the interactions. It has two steps: Decide which interaction mechanism is best suited to the target MAS (direct or indirect); and then Define how agents interact depending on the selected interaction mechanism. The resultant Agent Interaction Model is represented by a set of Interaction Protocol Diagrams. The developer validates the Agent Interaction Model against the Ontology Model. The Agent Class Model is also checked to ensure that all communicating agent classes share the same application ontologies that govern their interactions. Lastly, the Agent Relationship Diagram is updated to show descriptive information about each interaction pathway between agent classes.

Architecture Design: This activity deals with various design issues relating to agent architecture and MAS architecture. It has the following five steps: Identify Agent-Environment interface requirements; Select Agent Architecture for the most appropriate architecture(s) for agents in the MAS; Specify MAS Infrastructure facilities identifying system components that are needed to provide system-specific services; Instantiate agent classes; and Develop MAS deployment plan.

The application of Step 1 is the elicitation of work products from the MOBMAS methodology, beginning from the final work product, *MAS Deployment Diagram*. Each work product type that is identified (as a method fragment) is shown in Table 1, together with an allocated ID number. These IDs are then used in the fourth and fifth columns of this matrix table to identify the work product types with which the work product type in column 2 has a dependency (either a production dependency or a verification dependency).

To simplify the analysis in this proof-of-concept example, we group the various models in MOBMAS as shown in column 1 of Table 1.

Step 2 focuses on the construction of a WPPM for MOBMAS. To limit our discussion here, rather than individual work products, we propose to continue to use *work product groups*.

Starting with the deliverable software application, the immediately precedent models are those of the Architecture Group. In order to be able to create these models, it is necessary to have in hand the models of the Agent Interaction Model, Agent Class Model and Agent Behaviour Model groups. We then link these with available relationship types (dependencies) — Figure 2.

This reconstruction continues iteratively until the full methodology is created (Figure 3) with work product types connected to task types.

Analysis of this diagram (Step 3) reveals there are four work product types (System Task Model, Organisation Context Chart, Agent Tuple-Centre Diagram and Agent Architecture Diagram) that do not have dependencies on other work products. For these, further analysis is required to determine whether they contribute to the overall methodology. The first three each have work product types that are dependent on them while the fourth, *Agent Architecture Diagram,* documents the selected architecture to be used. MOBMAS recommends the adoption of an existing architecture where possible. This closer analysis suggests no changes are required.

A number of work products may also have unnecessary work product dependencies:

- Interaction Protocol Diagram
- Agent-Tuple-Space Interaction Diagram

If both of these work product types are derived from *Agent Plan Template*, *Reflexive Rule Specification* and *Agent Plan Diagram*, then they should already be consistent with work products used previously; these were already verified against the same work products as suggested for the Interaction Protocol Diagram and Agent-Tuple Space Interaction Diagram work products. By identifying unnecessary verification dependencies that may lead to development process overhead, methodology revision is suggested to perhaps remove some of the verification steps.

By analysing the WPPM, it is possible to suggest two sets of groups that can be developed in parallel without hampering the methodology:

- System Task Model Group and Organisational Context Model Group
- Role Model Group and Ontology Model/Resource Model Group

For the first, neither work product groups have dependencies. Since both are also initial work products, then they can be developed in parallel. In the case of the second, there are no direct task links between the two groupings. A further check on production dependencies also confirms that no relationship between the two groupings exists; consequently they could be developed in parallel.

In assessing whether any other work product groups could be developed in parallel, the only other possibility is the Agent Interaction Model Group and Agent Behaviour Model Group. Although neither are exclusively dependent on one another, they both require mutual verification (and possible revision)

Table 1: Work Product Types and their Dependencies.

Model Group	Work Product Types elicted (ID)	ID	Production Dependencies (by ID)	Verification & Refinement Dependencies (by ID)
Architecture Model	MAS Deployment Diagram	1	6, 7, 8, 9, 10, 11, 12, 13	-
	Class Instantiation	2	13	-
	Infrastructure facility specification	3	-	6, 7, 8, 9, 10, 11
	Agent architecture diagram	4	-	-
	Environment Interface	5	6, 7, 8	-
Agent Interaction	Interaction Protocol Diagram	6	9, 10, 11	12, 13, 14, 15, 16, 17, 18, 19
	Agent-Tuple-Space Interaction Diagram	7	9, 10, 11	12, 13, 14, 15, 16, 17, 18, 19
	Agent TupleCentre Diagrams (Optional)	8	-	-
Agent Behaviour	Agent Plan Template	9	11, 14, 15, 16, 17, 18, 19	12, 13, 15, 16, 17, 18, 19
	Reflexive Rule Specification	10	15, 16, 17, 18	12, 13, 14, 15. 16, 17, 18, 19
Agent Class	Agent Plan Diagram	11	19	-
	Agent Class Diagram	12	14, 15, 16, 17, 18, 19	-
	Agent Relationship Diagram	13	12, 19	-
	Agent Goal Diagram (optional)	14	19	-
Ontology	Application Domain Ontology	15	19, 21	-
	Application Task Ontology	16	19, 21	-
Resource	Resource Ontology	17	18	-
	Resource Diagram	18	19, 21	-
Role	Role Diagram	19	20, 21	15, 16
Organisation	Organisation Context Chart	20	-	
System Task	System task model	21	-	15, 16

of the Agent Class Model Group and Ontology Model Group. This mutual verification may make it difficult to develop any further work products in parallel.

In Step 4, these recommendations are formalized, ensuring that they reach the appropriate decision maker(s): in this case the authors of MOBMAS (one author of which kindly agreed to co-author this paper). A formal recommendation might thus have the following form and content.

- The System Task model and the Organisational Context diagram can be developed in parallel. These work products do not depend on any other work products. Furthermore, the Role Diagram and the Domain, Task and Resource Ontologies are also able to be developed in parallel. It should be noted that parallel development requires that both System Task model and Organisational Context Diagrams have both been completed.
- Omit the verification dependencies for the Interaction Protocol Diagram and Agent-Tuple-Space Interaction Diagram work products.

4 DISCUSSION AND CONCLUSIONS

A work product pool approach (Gonzalez-Perez & Henderson-Sellers, 2008) was adapted for use in devising a method to identify weaknesses in methodologies, and subsequently recommend revision(s). The rationale is that, since the final work product is the most important aspect of the methodology (as opposed to the process steps that produce the final work product and all necessary intermediate work products), a work product focus to methodology revision is well suited.

Methodology revision proposed in this paper is

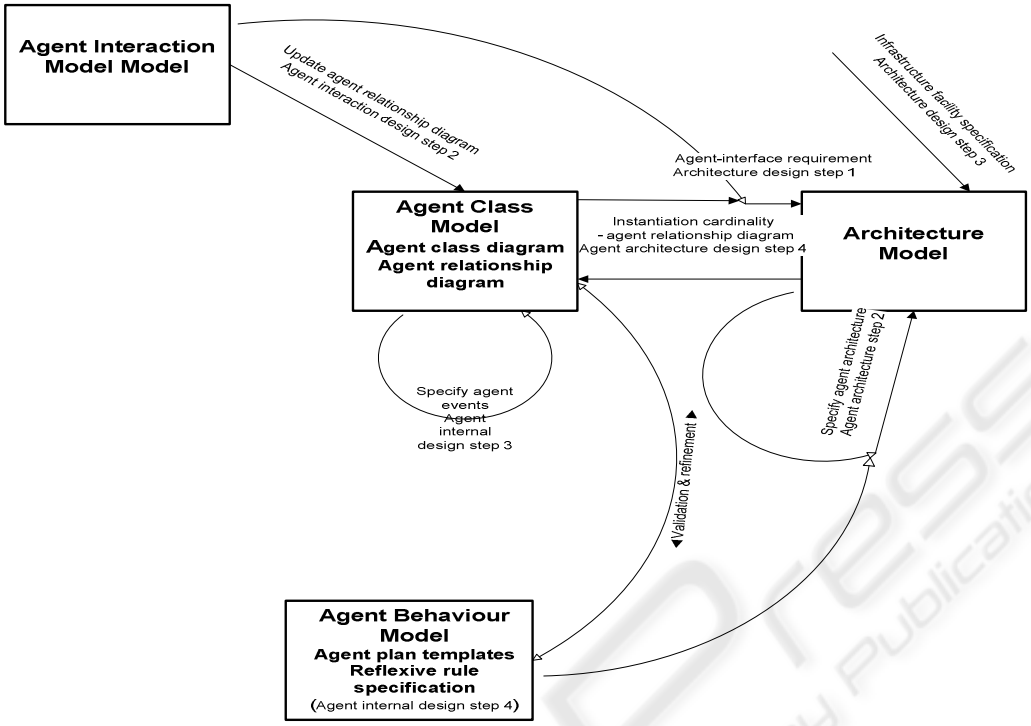

Figure 2: Initial version of the WPPM for the MOBMAS methodology.

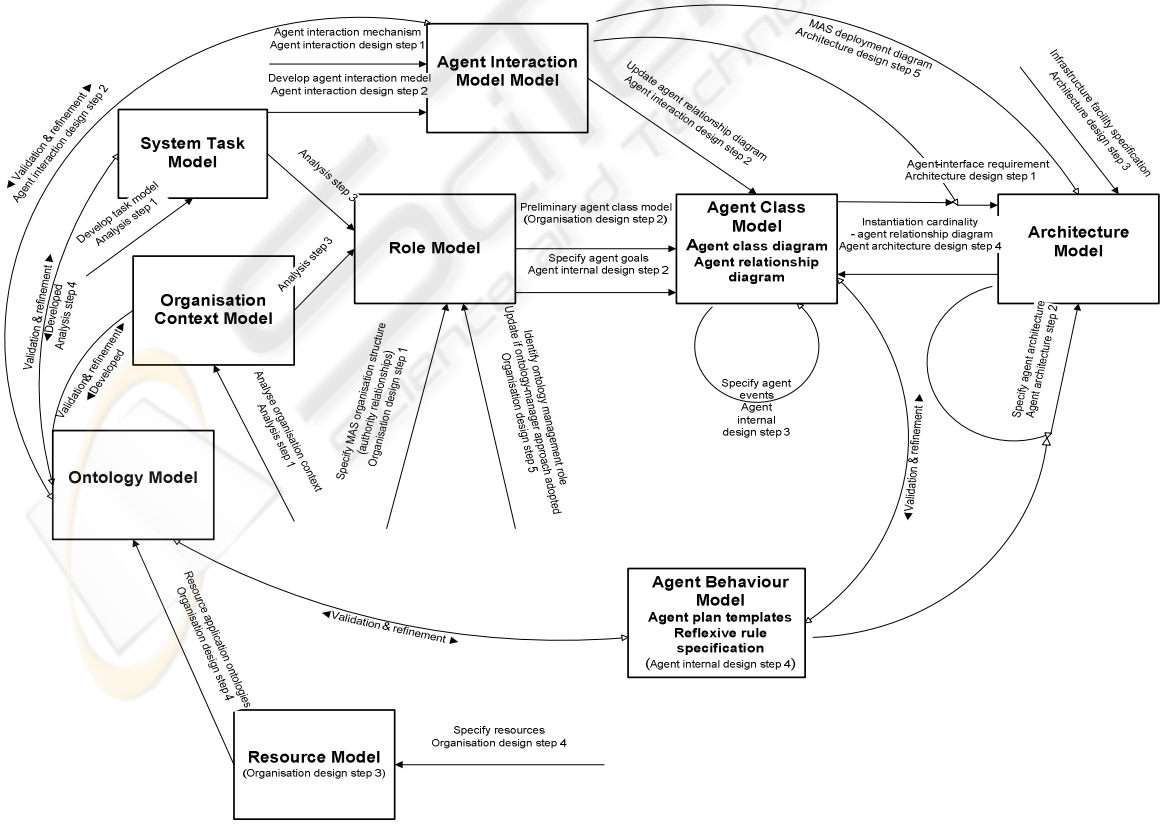

Figure 3: Group-level WPPM for the MOBMAS methodology.

fragments from the target methodology, recreating the target methodology from the fragments using the work product pool approach, identifying weaknesses in the methodology (both between work products, and globally across the whole methodology), and making recommendations for methodology revision aimed at minimising weaknesses. Work product dependencies are used as the basis upon which weaknesses are identified where work products are inadequate or missing, or process steps could be improved upon. Work product group models are used as the basis upon which management level recommendations are made to improve the enactment of a methodology instance.

The proposed methodology revision approach can also be placed within the context of method engineering and software process improvement (SPI). It is a distinct form of method engineering that focuses on work products rather than method/process components in order to engineer a better suited method. It is distinct from SPI because refinements are not project-specific – these are generic refinements. It is also distinct from SPI because SPI addresses general qualities of a methodology, such as efficiency or reusability i.e. the contribution made to SPI is of a relatively minor nature.

An important factor presented here is that the revision of the methodology was based on improving process steps for achieving work products. Further work includes addressing revising methodologies based on improving the work products themselves. However, in order to do this, an objective means of identifying the quality of work products is necessary.

REFERENCES

Gonzalez-Perez, C & Henderson-Sellers, B 2008, 'A Work Product Pool Approach to Methodology Specification and Enactment,' *Journal of Systems and Software*, vol. 31, no. 8, pp. 1288-1305.

Harmsen, F, Brinkkemper, S & Oei, JLH 1994, 'Situational Method Engineering for Informational System Projects Approaches,' in *Methods and Associated Tools for the Information Systems Life Cycle, Proceedings of the IFIP WG8.1 Working Conference on Methods and Associated Tools for the Information Systems Life Cycle*, eds AA Verrijn-Stuart & TW Olle, Elservier Science, New York, pp.169-194.

Henderson-Sellers, B, Serour, M K, Gonzalez-Perez, C & Qumer, A 2007, 'Improving Agile Software Development by the Application of Method Engineering Practices,' in *Proceedings of the 25th Conference on IASTED International Multi-Conference: Software Engineering*, ed W Hasselbring, ACTA Press, Calgary, Canada, pp. 55-60.

Ralyté, J, Brinkkemper, S & Henderson-Sellers, B (eds) 2007, *Situational Method Engineering: Fundamentals and Experiences, Proceedings of the IFIP WG 8.1 Working Conference*, Springer, New York.

Saeki, M, Iguchi, K, Kuo, W Y & Shinohara, M 1993, 'A Meta-Model for Representing Software Specification & Design Methods,' in *Information System Development Process, Proceedings of the IFIP WG8.1 Working Conference on Information Development Process*, eds N Prakash, C Rolland & B Pernici, North-Holland, pp. 149-166.

Tran, Q N N & Low, G C 2008, 'MOBMAS: A Methodology for Ontology-Based Multi-Agent Systems Development,' *Information Software and Technology*, vol. 50, no.708, pp. 697-722.

Tran, Q N N, Low, G C & Beydoun, G 2006, 'A Methodological Framework for Ontology Centric Oriented Software Engineering,' *International Journal of Computer Systems Science and Engineering*, vol. 21, no. 2, pp. 117-132.