Towards Pattern-Based Generation of Requirements from System Models

N.A.M. Maiden, S. Manning, S. Jones and J. Greenwood

Centre for HCI Design, City University
Northampton Square, London EC1V OHB, UK
1National Air Traffic Services, Lancaster Place, London, UK

Abstract. Academic research has produced many model-based specification and analysis techniques, however most organisations continue to document requirements as textual statements. To help bridge this gap between academic research and requirements practice, this paper reports an extension to the RESCUE process in which patterns for generating requirements statements from i* system models were manually applied to i* models developed for a complex air traffic control system. The paper describes the results of this application, the benefits of the approach to the project, and ongoing research to implement these patterns in the REDEPEND modelling tool to make requirements engineers more productive.

1. Introduction

There are many model-based specification and analysis approaches reported in the literature to specify the requirements of computer-based systems (e.g. [4, 16]). In contrast, most organizations continue to represent requirements textually, both to enable requirements to be reviewed by stakeholders, and to deliver requirements documents that are legally binding on the contractor. Unfortunately, most modeling approaches have not been designed to support the derivation of requirements statements from models or to be used alongside textual requirement descriptions.

In this paper we report an approach in which we manually applied 19 simple patterns to two i* models [16] describing a complex air traffic management (ATM) system to derive over 200 textual requirements statements structured using the VOLERE shell [14]. The simplicity of the patterns and their effectiveness in the reported case study suggests opportunities to derive requirements statements automatically from i* models, with significant potential productivity gains.

The remainder of this case study paper is in 5 sections. The next section describes RESCUE and its application to produce a requirements specification for DMAN, a new air traffic management system. Section 3 reports the two i* models of the DMAN system that were generated. Section 4 describes 19 patterns that were designed to generate candidate requirements statements from i* models. Section 5 describes how the DMAN team manually applied these patterns to the two models, and quantitative...
and qualitative results. The paper ends with proposals to implement these derivation patterns within our REDEPEND tool for $i^*$ modeling, implications for deriving requirements statements from other requirements modeling techniques, and a research agenda for tighter coupling of requirements modeling and description techniques.

2. The RESCUE Process and DMAN Project

Previously, the authors worked with Eurocontrol to design and implement RESCUE [8], an innovative process to determine stakeholder requirements for systems that will provide computerised assistance to air traffic controllers. RESCUE was successfully applied to determine the requirements for CORA-2, a complex socio-technical system in which controllers work with a computerised system to resolve conflicts between aircraft on a collision path [11]. In this paper we report the application of a new version of RESCUE to model the requirements for DMAN, a socio-technical system for scheduling and managing the departure of aircraft from major European airports such as Heathrow and Charles de Gaulle. A requirements team that included engineers from UK and French air traffic service providers modeled the DMAN system and requirements.

DMAN is a complex socio-technical system involving a range of human actors including tower controllers and aircraft pilots, interacting with other computer-based systems related to both airport and air movements, and supporting aircraft movement from push back to take off. The project was led by the UK’s National Air Traffic Services (NATS) and involved participants from Centre d’Etudes de la Navigation Aerienne (CENA) and City University’s RESCUE experts. The DMAN team was composed of 2 systems engineers employed by NATS and CENA and one RESCUE team member from City. It also worked with 4 UK and 4 French air traffic controllers who were seconded to the project, other NATS and CENA engineers, and software engineering academics.

3. $i^*$ System Models Generated for DMAN

RESCUE adopts the established $i^*$ approach [16] but extends it to model complex technical and social systems, establish different types of system boundaries, and generate requirement statements. $i^*$ is an approach originally developed to model information systems composed of heterogeneous actors with different, often-competing goals that depend on each other to undertake their tasks and achieve these goals – like the socio-technical and information systems found in air traffic management (ATM).

The DMAN system is designed to operate in airport control towers such as the one depicted in Figure 1. Different air traffic controllers (ATCOs) fulfill different roles. Two or more ATCOs control aircraft take-offs and landings. Other ATCOs control movements on the ground, from start-up to taxiing, and there are also a tower departure sequence ATCO, a tower supervisor and assistants, all who work together and depend upon each other to control aircraft safely from start-up to take-off and
from landing to park. Communication and responsibilities are currently managed using physical flight strips, one per flight, which are annotated by and passed between ATCOs from push back to take off.

![ATCOs working at the Heathrow control tower](image)

**Figure 1. The Heathrow control tower, showing the departure and arrival runway controllers in the foreground, other tower controllers in the background, and a chute along which flight strips pass to manage a departing flight**

The first i* model produced is the Strategic Dependency (SD) model, which describes a network of dependency relationships among actors identified in the context model. The opportunities available to these actors can be explored by matching the depender who is the actor who “wants” and the dependee who has the “ability”. Since the dependee’s abilities can match the depender’s requests, the system-wide strategic model is developed. Figure 2 shows the SD model for the DMAN system. The SD model specified 15 actors with 46 dependencies between these 15 actors. It specifies other systems that either depend on or are depended on by DMAN (e.g. TACT and A-SMGCS), and human roles that depend on DMAN to do their work (e.g. Runway ATCO and Departure Clearance ATCO). For example, the SD model specifies that DMAN depends on TACT to achieve the goal CTOT and slot messages updated, and A-SMGCS depends on DMAN to undertake the task update taxi time estimates. Likewise, DMAN depends on the Tower Departure Sequencer ATCO to have the departure sequence manual update, and the Departure Clearance ATCO depends on DMAN to achieve the soft goal workload not increased.

RESCUE provokes the team to ask important questions about systems boundaries by re-expressing these boundaries in terms of the goal dependencies between actors on either side of a boundary. Actors with goals that the team will seek to test for compliance are, by definition, part of the new system. Such re-expression also leads to more effective requirements description by referring to named actors that will be tested for compliance (e.g. “The controller using DMAN shall have access to the departure sequence”). The DMAN team used REDEPEND to produce the i* SD model. REDEPEND is a graphical modeling tool developed as a plug-in to Microsoft Visio 2002 that enables the team to construct and analyse i* SD and SR models [10].
The second type of $i^*$ model is the Strategic Rationale (SR) model, which provides an intentional description of how each actor achieves its goals and soft goals. An element is included in the SR model only if it is considered important enough to affect the achievement of some goal. The SR model includes the SD model, so it describes which actors may be able to accomplish something by themselves, or by depending on other actors. It specifies goals, tasks, resources and soft goals linked by dependency links from the SD model, task decomposition links, means-end links, and the contributes-to-soft goal links [16].

The DMAN SR model was more complex, with a total of 103 model elements describing 7 of the 15 actors defined in the SD model. For example, the SR model shows that the human Runway ATCO actor undertakes one major task – control flight around the runway – that is decomposed into other tasks such as issue line-up clearance and issue take-off clearance. The former task can be further decomposed into sub-tasks and sub-goals which, if undertaken and achieved, contribute negatively to the achievement of an important soft goal – that workload should not be increased.
4. Requirements Generation Patterns

In DMAN these 2 \( i^* \) models were used originally by the team to analyze boundaries and validate use case descriptions [8], but not for generating requirements statements directly. Therefore we designed simple patterns – recurring syntactic and semantic structures in the \( i^* \) models – that we applied manually to the SD model to generate textual requirement statements. Our patterns are not traditional design sense – a solution to a problem in context. Rather each pattern defines one or desired properties (requirements) on the future system that must be satisfied for the SD model dependency to hold for the future system. As such, the SD model, which has been signed off as complete and correct, informs further discovery and specification of requirements statements. The patterns were originally developed as an aid for conducting synchronization checks between \( i^* \) SD models and requirements specifications in RESCUE [8]. However, the effort committed to develop the DMAN \( i^* \) SD model raised the possibility of also deriving textual requirement statements from it.

We designed the patterns in 2 stages. Firstly, we drew on the RESCUE meta-model, which links \( i^* \) model concepts to requirement statements in the logical structure of RESCUE requirements document, and adjacent system types [14], to specify statements of functional requirement that must be satisfied to maintain different types of \( i^* \) dependency specified in the model. For example, the model dependency the Runway ATCO depends on DMAN for departure sequence information necessitates requirements on the ATCO – the Runway ATCO shall have departure sequence information and the Runway ATCO shall receive departure sequence information from DMAN, and on DMAN – DMAN shall provide departure sequence information to the Runway ATCO - for the dependency to be supported in the DMAN system. These requirements are best expressed as textual statements – their final, more expressive and succinct form in the requirements document. Secondly, we worked with the two systems engineers on the DMAN team to define types of non-functional requirement on different types of \( i^* \) dependencies that always apply for ATM systems such as DMAN. These types included timeliness, accuracy, reliability and performance, To continue the earlier example, candidate requirement statements include the Runway ATCO shall have departure sequence information on time and the Runway ATCO shall have up-to-date departure sequence information. Note that the patterns are not expected to generate measurable and testable requirements statements directly. Rather, the generated statements define statements that require decomposition and refinement to be testable.

Part of the RESCUE meta-model is depicted in Figure 3. Each \( i^* \) goal and soft goal (shown on the right-hand side of the diagram) can be mapped to one or more textual requirement statements in a requirement document (shown on the left-hand side). Each requirement statement makes reference to at least one actor (e.g. Runway ATCO, DMAN) to which the statement attributes desirable properties (e.g. workload shall not be increased), and sometimes to one or more objects that these actors manipulate (e.g. the calculated departure sequence shall be updated). Actors are explicitly modeled in \( i^* \) SD and SR models. Thus we are able to map a requirement actor and an \( i^* \) actor. Likewise, requirement objects are often information or physical resources (e.g. the departure sequence, runway slot) that are modeled as resources to produce or
consume in \textit{i*} models. Thus we are able to map a requirement object to an \textit{i*} resource.

The resulting 19 RESCUE patterns are similar to previous goal pattern research. Rolland et al. [13] describe discovery-guiding rules for discovering system goals. For example, their composition-guiding rule C2 constructs interaction pairs for \textit{consuming} and \textit{producing} resources in scenarios. If there is a goal to consume a resource, there must also be a goal to produce that resource. Our work is similar in spirit to these rules but exploits the RESCUE meta-model to leverage system models to \textit{transform} system dependencies into the needed textual requirements statements rather than \textit{discover} the requirements statements for the first time.

![Figure 3. Part of the RESCUE meta-model](image)

Our reuse of non-functional requirements patterns borrows from Sindre et al. [15], who enable reuse of security requirements using the concept of a security bundles that handle security threats to a system, and Chung et al [2] who model recurring non-functional requirements structures for reuse. However, our approach is simpler and makes no general claims as systematic reuse strategies across domains.

Our patterns also bear some similarities to satisfaction arguments. Satisfaction arguments are explanatory structures that provide a rationale for how one or more system requirements, if satisfied, lead to the satisfaction of one or more stakeholder requirements [12]. Satisfaction arguments have been successfully applied to large-scale system specifications [6]. If we treat dependencies in a SD model as stakeholder requirements, i.e. goals or soft goals that an actor wants to achieve or attain, and generated requirement statements as system requirements that must be satisfied for the dependency goal and soft goal to be achieved and attained, then each pattern encapsulates a tacit satisfaction argument that adds structure to the overall requirements specification. We shall return to the role of patterns as satisfaction arguments in the discussion.

4.1 Dependency-Specific Patterns

The 19 RESCUE patterns were of 2 types. The first 16 were specific to the \textit{i*} model dependency, defined in terms of the dependency’s process elements (goal, task,
resource and soft goal) and the types of depender and dependee actors (new system, adjacent system, stakeholder). The 16 patterns are specified in Table 1. Definitions of SD model constructs are: new software system actor (NSA); adjacent system actor (ASA); stakeholder actor (STA); any actor type (ANY); goal (G); soft goal (SG); task (T); resource (R); dependency association between the depender actor DR and dependee DE actor for an outcome O (DR depends DE: O). Definitions of requirement statement constructs are: functional requirement (FR); non-functional requirement (NFR). Each pattern was given a unique ID for reference purposes.

<table>
<thead>
<tr>
<th>ID</th>
<th>SD dependency</th>
<th>Candidate requirement statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>NSA depends ASA: SG</td>
<td>NFR (NSA shall achieve SG)</td>
</tr>
<tr>
<td>P2</td>
<td>NSA depends ASA: R</td>
<td>FR (NSA shall receive R from ASA)</td>
</tr>
</tbody>
</table>
| P3 | NSA depends ASA: G | FR (NSA shall attain G)  
FR (ASA shall provide NSA with R)  
FR (NSA shall receive R from ASA)  
NFR (NSA shall attain G accurately)  
NFR (NSA shall attain up-to-date G)  
NFR (ASA shall provide NSA with up-to-date R)  
NFR (ASA shall provide NSA with R accurately) |
| P4 | NSA depends ASA: T | FR (NSA shall do T)  
FR (ASA shall provide NSA with R)  
FR (NSA shall receive R from ASA)  
NFR (ASA shall provide NSA with R on time) |
| P5 | ASA depends NSA: T | FR (ASA shall do T)  
FR (NSA shall participate in T) |
| P6 | ASA depends NSA: G | FR (ASA shall attain G)  
FR (NSA shall do T [that enables ASA to attain G])  
FR (ASA shall receive R from NSA)  
FR (ASA shall provide ASA with R)  
NFR (ASA shall provide ASA with R accurately)  
NFR (ASA shall provide ASA with up-to-date R) |
| P7 | STA depends NSA: SG | NFR (STA shall achieve SG) |
| P8 | STA depends NSA: G | FR (STA shall attain G)  
FR (NSA shall do T [that enables ASA to attain G]) |
| P9 | STA depends NSA: R | FR (STA shall receive R from NSA)  
FR (NSA shall provide R to STA) |
| P10 | STA depends NSA: T | FR (STA shall do T)  
FR (NSA shall do subtask of T) * can be multiple tasks  
FR (NSA shall provide R to STA)  
NFR (NSA shall provide R to STA on time) |
| P11 | STA depends NSA: G | FR (NSA shall achieve G)  
FR (NSA shall enable STA to provide R)  
FR (NSA shall receive R from STA)  
FR (STA shall provide NSA with R) |
| P12 | NSA depends STA: G | FR (STA shall provide NSA with R)  
FR (NSA shall enable STA to provide R)  
FR (NSA shall receive R from STA) |
| P13 | ANY depends ANY: T | NFR (ANY shall do T accurately)  
NFR (ANY shall do T up-to-date) |
| P14 | ANY depends ANY: G | NFR (ANY shall attain G on time)  
NFR (ANY shall attain G reliably) |
| P15 | ANY depends ANY: G | NFR (ANY shall attain G on time)  
NFR (ANY shall attain G reliably) |
| P16 | ANY depends ANY: G | NFR (ANY shall attain G reliably) |

Table 1. Patterns for deriving requirements from the SD model
The 16 patterns can be divided into three broad categories – P1-P6 that link the computerized system actors (NSA & ASA); P7-P12 that link stakeholders to the new system (STA & NSA), and P13-P16, which are independent of these actor types.

P1-P6 define requirement statements for 6 of the 8 types of possible dependencies between systems. No patterns were defined for the dependencies in which an adjacent system actor depends on the new system to achieve soft goals or obtain resources – specifying requirements on adjacent systems to achieve soft goals and resource needs was considered beyond the scope of the DMAN requirement specification. P1 and P2 are simple – if the new system depends on another system to achieve a soft goal or obtain a resource, then we need requirement statements that the new system shall achieve the soft goal and obtain the resource. P3 specifies the need for functional requirement statements on the new system to attain a goal G, requirement statements on the adjacent system to enable the new system to attain G, and 4 types of non-functional requirement statement on the new and adjacent system to provide resources that enable the attainment of G on time, reliably, accurately and with up-to-date resources. P4 – for the task dependency – specifies the need for requirement statements that must be satisfied for effective task interaction between the 2 systems – the new system shall do the task, the adjacent system shall provide resources for the task, and that the new system shall receive them. P5 and P6 define the need for requirement statements on the new system so that dependent adjacent systems shall undertake tasks and attain goals. P5 is simple – the adjacent system shall undertake a task and the new system shall participate in that task. Pattern 6 defines the need for requirement statements to provide and receive resources on both systems and non-functional requirement statements on this resource exchange.

P7-P12 define requirements statements for 6 of the 8 types of possible dependencies between human actors that are part of the DMAN socio-technical system and the new software system. P7, P8 and P9 define simple requirement statements for dependencies in which the stakeholder depends on the new system to achieve soft goals, attain goals and obtain resources. P10 defines requirements on the new software system that will enable the stakeholder to undertake the task defined in the dependency. Similarly, P11 and P12 for dependencies in which the new system depends on the human actor to attain system goals and obtain resources, define requirements on the stakeholder to provide the resources to the system and on the system to receive the resources.

P13-P16 are different in that they are independent of the dependant and dependant actor types. Each pattern defines different types of non-functional requirement that should be satisfied when actors depend on each other to attain goals and complete tasks, according to the 2 DMAN systems engineers with ATM domain expertise. There is some overlap in the design of the patterns, which reflects how they were developed and applied in DMAN. Originally we defined P13 and P14, where P14 imposes timeliness and reliability requirement statements on relevant goal dependencies. However, during the DMAN work, the patterns were extended according to circumstances. P15 specifies the need for a timeliness requirement statement independent of a reliability requirement, whilst P16 specifies the need for a reliability requirement statement independent of a timeliness requirement. Other non-functional requirement types important in the DMAN project, such as safety and
security, are handled differently using soft goal dependencies in the SD model to generate textual requirement statements using pattern P7.

4.2 Composite Dependency Patterns

Three additional patterns C1-C3 were specified to handle composite process elements in i* model dependencies. Whereas RESCUE mandates atomic requirement statements that cannot be further decomposed, the systems engineers had developed the i* SD model to include compound dependencies such as \textit{DMAN depends on ATC tower supervisor for current and foreseen runway status}, primarily to simplify the development and management of the complex i* models. Therefore we introduced 3 patterns C1-C3 specified in Table 2 to detect and decompose composite dependencies. Definitions of SD model constructs are: any actor type (ANY); goal (G); task (T); resource (R); dependency association between the depender actor DR and dependee DE actor for an outcome O (DR depends DE: O); composite outcome (task, resource or goal) that can be decomposed (Composite (O)); decomposed goal G: (G1, G2, Gn); decomposed task T (T1, T2, Tn); decomposed resource R (R1, R2, Rn). Again each pattern was given a unique ID, and could be applied recursively if the results of applying the pattern were not atomic model dependencies.

<table>
<thead>
<tr>
<th>No</th>
<th>SD model dependency</th>
<th>Derived SD dependencies</th>
</tr>
</thead>
</table>
| C1 | ANY depends ANY: G & Composite (G) | ANY depends ANY: G1  
ANY depends ANY: G2  
ANY depends ANY: Gn |
| C2 | ANY depends ANY: T & Composite (T) | ANY depends ANY: T1  
ANY depends ANY: T2  
ANY depends ANY: Tn |
| C3 | ANY depends ANY: R & Composite (R) | ANY depends ANY: R1  
ANY depends ANY: R2  
ANY depends ANY: Rn |

Table 2. Patterns for decomposing compound i* model dependencies into atomic dependencies

The next section describes the effort required by the DMAN project team to apply the patterns to the DMAN SD model manually, and results from this application.

5. Deriving Requirements Statements from the DMAN Models

To recap, the DMAN SD model specified 15 actors with 46 dependencies between these 15 actors. It models other systems that either depend on or are depended on by DMAN (e.g. TACT and A-SMGCS), and human roles that depend on DMAN to do their work (e.g. Runway ATCO and Departure Clearance ATCO). One systems engineer, an experienced member of the RESCUE team at City University assigned to the DMAN project, took a total of a total of 3 working days to apply the 19 patterns to all 46 dependencies modeled in the SD model.
The result was 214 new DMAN requirement statements – almost 25% of the total number of requirements statements in the final DMAN requirements specification. Given the DMAN requirements project duration – 10 months – this represents a major advance of the DMAN specification in a short period of time, notwithstanding the time spent to produce the SD model in the first place. All DMAN requirements were specified using the VOLERE shell [14]. Using the 19 patterns, the systems engineer was able to specify the requirement identifier, description, type, rationale and source attributes of each requirement, and other requirements that were dependent on it. The identifier was automatically generated using the requirements management tool, the description was derived from the dependency (actor and process element) labels, the type from the pattern being applied, and the rationale for the requirement was the originating i* SD model dependencies. An example VOLERE shell for a DMAN requirement, generated from the application of P8, is shown in Figure 4.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>FR226</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Functional</td>
</tr>
<tr>
<td>Description</td>
<td>DMAN shall calculate the optimized runway capacity</td>
</tr>
<tr>
<td>Rationale</td>
<td>Derived from SD model, which specifies the basic rationale for the modeled dependency</td>
</tr>
<tr>
<td>Source</td>
<td>Dependency [ATC tower supervisor depends on DMAN to attain goal runway capacity optimized]</td>
</tr>
<tr>
<td>Dependencies</td>
<td>FR225 (ATC tower supervisor shall optimize the runway capacity)</td>
</tr>
</tbody>
</table>

**Figure 4. A VOLERE shell for a DMAN requirement**

Table 3 shows the number of separate applications of each pattern to the model, and the total number of requirement statements generated from these applications. The 19 patterns were applied a total of 85 times to the 46 dependencies – an average of almost 2 applications per model dependency, although the number of applications across the patterns varied. The system engineer applied the patterns to each modeled dependency in turn using her requirements and domain expertise, She applied between 0 and 3 different patterns to each modeled dependency. For example, the modeled dependency DMAN depends on AMAN to attain the goal of arrival sequence and constraints updated led to the generation of 14 different requirement statements from the application of patterns P3, P13 and C1. In contrast, applying P7 to the modeled dependency Ground ATCO depends on DMAN to achieve the soft goal workload not increased generated only one requirement statement – the Ground ATCO using DMAN shall not receive an overall increase in workload. This reliance on human judgment indicates that we need to be cautious in our claims that can be made using the quantitative data. Nonetheless, the results shown in Table 3 warrant some further analysis.

<table>
<thead>
<tr>
<th>No</th>
<th>Number of applications of each pattern to the DMAN SD model</th>
<th>Total number of requirement statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The application of patterns P1-P12 was dependent on the type of dependency and actors modeled in the DMAN SD model. The most often applied was P8 and the least often applied were P1 and P2, i.e. the most common dependency in the model was a stakeholder depending on DMAN to attain a goal whilst the dependencies where DMAN depends on an adjacent system to achieve a soft goal or obtain a resource did not exist in the model. To some degree these results are an artifact of the \( i^* \) modeling style that RESCUE encourages – modeling the goals and soft goals of human actors in the socio-technical system rather than the tasks and resources of the future DMAN software system.

The patterns C1-C3 were applied a total of 17 times, indicating that almost one-third of all dependencies in the DMAN model modeled composite goals, resources and tasks that needed decomposition in order to provide atomic requirements statements. In several cases the engineer was able to decompose one dependency into 4 separate dependencies using her domain expertise. For example, the task dependency \( \text{DMAN depends on FDPS (the flight data processing system) to update flight plan and ATC status} \) was decomposed into the following 4 dependencies:

- \( \text{DMAN depends on FDPS to update the flight plan;} \)
- \( \text{DMAN depends on FDPS to update ATC status;} \)
- \( \text{DMAN depends on FDPS to receive the flight plan;} \)
- \( \text{DMAN depends on FDPS to receive ATC status.} \)

Another result from this case study was the need for expert human judgment when selecting and writing the textual requirements statements. Results revealed that detecting the need for requirement statements can be mechanized, but developing and wording these new statements cannot be, as the following example demonstrates. P8 was applied to the modeled dependency \( \text{Departure Clearance ATCO depends on DMAN to attain the goal timely start-up achieved.} \) As well as the requirement statement \( \text{the Departure Clearance ATCO shall achieve timely start-up,} \) the engineer also specified requirement statements including \( \text{the DMAN system shall calculate the MOBT and the DMAN system shall provide the Departure Clearance ATCO with the MOBT using knowledge that the task Calculate start-up time as the MOBT must be} \)

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Count</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3</td>
<td>8</td>
<td>26</td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>P5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P6</td>
<td>8</td>
<td>23</td>
</tr>
<tr>
<td>P7</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>P8</td>
<td>13</td>
<td>25</td>
</tr>
<tr>
<td>P9</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>P10</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>P11</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>P12</td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>P13</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>P14</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>P15</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>P16</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>C1</td>
<td>11</td>
<td>54</td>
</tr>
<tr>
<td>C2</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>C3</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>

**Table 3. Results from applying patterns to the DMAN \( i^* \) SD model**

The application of patterns P1-P12 was dependent on the type of dependency and actors modeled in the DMAN SD model. The most often applied was P8 and the least often applied were P1 and P2, i.e. the most common dependency in the model was a stakeholder depending on DMAN to attain a goal whilst the dependencies where DMAN depends on an adjacent system to achieve a soft goal or obtain a resource did not exist in the model. To some degree these results are an artifact of the \( i^* \) modeling style that RESCUE encourages – modeling the goals and soft goals of human actors in the socio-technical system rather than the tasks and resources of the future DMAN software system.

The patterns C1-C3 were applied a total of 17 times, indicating that almost one-third of all dependencies in the DMAN model modeled composite goals, resources and tasks that needed decomposition in order to provide atomic requirements statements. In several cases the engineer was able to decompose one dependency into 4 separate dependencies using her domain expertise. For example, the task dependency \( \text{DMAN depends on FDPS (the flight data processing system) to update flight plan and ATC status} \) was decomposed into the following 4 dependencies:

- DMAN depends on FDPS to update the flight plan;
- DMAN depends on FDPS to update ATC status;
- DMAN depends on FDPS to receive the flight plan;
- DMAN depends on FDPS to receive ATC status.

Another result from this case study was the need for expert human judgment when selecting and writing the textual requirements statements. Results revealed that detecting the need for requirement statements can be mechanized, but developing and wording these new statements cannot be, as the following example demonstrates. P8 was applied to the modeled dependency Departure Clearance ATCO depends on DMAN to attain the goal timely start-up achieved. As well as the requirement statement the Departure Clearance ATCO shall achieve timely start-up, the engineer also specified requirement statements including the DMAN system shall calculate the MOBT and the DMAN system shall provide the Departure Clearance ATCO with the MOBT using knowledge that the task Calculate start-up time as the MOBT must be
undertaken by the DMAN system. Contrast this with what pattern P8 states – that DMAN shall undertake a task to enable the goal. That said, the engineer in possession of this domain knowledge was able to write a large number of structured requirements statements based on the SD model in a short period of time.

The effort needed to write VOLERE measurable fit criteria for the 214 requirements raised concerns about the similarity of requirement statements generated from the SD model. DMAN’s project manager reported that some of these requirements statements, although different, were insufficiently distinguishable for the intended use of the DMAN requirements specification. For example, the two requirement statements *Ground ATCO shall provide DMAN with ready status information* and *DMAN shall provide the Ground ATCO with facilities to input ready status information*, which need to be tested differently, added to the length of the specification without contributing significantly to its completeness. That said, only 7 of the 214 generated requirement statements were finally dropped from the final requirements specification, due in part to the RESCUE team providing more reusable templates for writing fit criteria for different requirement types.

At the end of the DMAN project, we were able to place these 207 requirement statements from the SD model into the DMAN requirements specification. The final DMAN requirement statements contained almost 900 such statements that came from 4 main sources – the DMAN creativity workshop (Maiden et al. 2004b), the DMAN concept definition, DMAN scenario walkthroughs using the ART-SCENE environment, and the i* SD model. As such, pattern-based generation accounted for almost 25% of the requirement statements in the specifications. Whereas most DMAN creativity workshop, concept and scenario walkthrough requirements tended to be lower-level functional and requirements on individual use cases and use case actions, 157 of the 187 system-wide requirement statements in the final specification were generated from the SD model. This finding suggests that requirements statements derived from i* system models provide important system-level requirements that complement and do not duplicate requirements from other RESCUE processes. In contrast, however, further work was still needed to structure these statements into decomposition and refinement hierarchies and to make them effectively traceable within the DMAN specification.

6. Implementing Requirement Generation Patterns in REDEPEND

DMAN systems engineers developed the i* system models using the REDEPEND graphical modeling and analysis tool plug-in to Visio. They successfully used REDEPEND to produce the SD and SR models, although the size of the SR models made it difficult to fit the complete model on one Visio sheet. The version of REDEPEND used in the DMAN project is depicted in Figure 7.

The DMAN case study suggests that REDEPEND can be a cost-effective tool for producing i* models and exploring candidate system boundaries. However, results also suggest that implementing the requirements generation patterns within REDEPEND can be a potential productivity aid to engineers. To bring this about we
will revise the patterns based on the results reported in this paper. We will then add new capabilities to REDEPEND for drafting first-cut requirement statements in VOLERE requirement shells from a selected SD model. These capabilities will: (i) graph-walk the selected model to identify each actor dependency; (ii) test each dependency for composite process elements – goals, tasks and resources – using the C-patterns to generate atomic dependencies; (iii) match the P-patterns to each atomic dependency to generate first-cut requirements statements with types, rationale, source and dependencies directly into tailored MS Word documents containing the candidate statements. By automatically generating these candidate requirement statements, we aim to exploit evidence that people are better at identifying errors of commission rather than omission [1], that is they are better at recognising incorrect rather than missing requirements statements. We have already exploited this general trend in human cognition for recall to be weaker than recognition when designing the ART-SCENE scenario walkthrough tool [11]. Generated requirement statements will be delivered to systems engineers so that they can be modified or rejected easily using macros that we implement in the REDEPEND Word documents. We plan to evaluate these capabilities on a new roll out of the RESCUE process, to deliver a stakeholder requirements specification for the NATS Flexible Use of Airspace (FUA) concept, due for completion in the second quarter of 2005.

Figure 5. The REDEPEND tool showing the DMAN i* SD model

7. Discussion and Future Work

This paper reports the results from a requirements engineering case study. A project team used RESCUE to develop the requirements specification for DMAN, a socio-technical system that will manage aircraft departures at major European airports, and produced i* SD and SR models using the REDEPEND tool to explore system boundaries and dependencies. The team also developed 19 patterns that were
applied to the SD model in 3 working days to generate 214 requirement statements, of which 207 were included in the final requirement specification. This success on a large-scale requirements engineering project reveals the potential productivity benefits of pattern-based requirements generation from system models.

Although the work was successful there are clearly some areas for future improvement. Some of the 19 patterns were developed in response to the changing needs of the DMAN project team, so the set of patterns needs to be rationalized (e.g. to remove duplicates in P13-P16) and extended to the complete set of possible dependency-actor type combinations and to generate all types of non-functional requirement statement.

The complete pattern set will then be elaborated using the concept of a satisfaction argument so that each pattern also includes generic rationale and trade-off statements [5]. For example P10 generates 4 requirement statements from the modeled dependency Stakeholder depends on new system to undertake a task. The pattern elaborated with satisfaction argument statements will also describe why new system tasks and the timely provision of resources contribute to the stakeholder completing the dependent task. Of course, systems engineers will still need to specialize these argumentation structures to the application goals, tasks and resources, but we believe that the provision of generic argument statements will lead to better argued requirements, again based people’s tendency to identify errors of commission rather than omission [1].

These results also suggest longer-term research directions that we will explore. Currently RESCUE offers simple sentence templates to generate requirement statements. We are keen to investigate reversing existing natural language parsing techniques in requirements engineering, for example the use of case grammars (Rolland et al. 1998), to generate more complete and tailored statements. Indeed, if system models are to be used effectively alongside textual requirements, seamless integration of modeling techniques [3] and two-way translation techniques based on parsing research and comprehensive requirements meta-models will be needed to generate requirements in the representation that best affords different discovery, acquisition and analysis tasks.

The success of the requirements generation process has implications for the design of the wider RESCUE process. The DMAN team applied RESCUEv4 to develop SR models for 7 human actors in the SD model. The number of interconnected SR model elements made the model time-consuming for the DMAN team to produce, whilst the benefits to DMAN’s requirements specification from the model were not as great as from the SD model. Indeed, an analysis of the SR model and requirement specification revealed many overlaps. As a result, in the design of RESCUEv5, we have decided to remove system-wide SR modeling and only use it for soft goal trade-off analyses on critical issues. Instead most SR model concepts will be expressed textually as requirement statements, thus exploiting the greater expressiveness of textual statements when the benefits for system modeling techniques are not justified.

We are currently starting to operationalise and implement a revised version of the 19 patterns in a new version of the REDEPEND tool, then evaluate its effectiveness in the NATS FUA project later in 2004. We look forward to reporting this work in the near future.
Acknowledgements

The authors wish to thank all members of the DMAN project team.

References