Compliance Flow - Managing the Compliance of Dynamic and Complex Processes

Paul W H Chung, Larry Y C Cheung and Colin H C Machin

Department of Computer Science, Loughborough University, UK

Email: p.w.h.chung@lboro.ac.uk

Abstract

To develop a reliable system or product, the current best practice for the development process is typically embodied in standards and guidelines, such as IEC61508 for safety and ISO9001 for quality assurance. Generally, the standard proposes a framework, which deals in a systematic manner with all the activities necessary to achieve the required quality. However, every application of a given standard is different because of differences in project details. One serious limitation of current workflow systems is the lack of the ability to ensure that the specification and execution of a process are compliant with the standard.

This paper presents the treatment of managing process compliance in the Compliance Flow system. Process-based reasoning is used to identify compliance errors within a user-defined process by matching it against the standard model during both process specification and process execution. Examples drawing on a version of IEC61508 are used to illustrate the mechanism of modelling and compliance checks. A case study of the development of a light-guard is discussed.

Keywords: Workflow management systems; Automated compliance checks; Model of standards

1. Introduction

In order to provide acceptable systems or services, the current best practice for the development process is typically embodied in recent standards and guidelines, such as IEC61509 (IEC, 1997) for safety and ISO9001 (ISO9001, 2000) for quality assurance. In this context, a user-defined process complies with a standard if there is a clear description of the development stages, the inputs to each stage are fully and unambiguously defined, and all the objectives and requirements stipulated in the standard are met. In safety-related engineering projects significant amounts of resource are spent on their management and in demonstrating standards compliance; much of the time of developers, managers and quality assurance teams is occupied with tracking and managing the compliance of the project. A workflow system with compliance management ability can considerably shorten the development time and reduce costs.

This paper describes the solution to compliance management in the Compliance Flow system. The proposed approach is to represent a given standard into a structured semi-formal model which acts like a knowledge database, providing information about a standard for process management. Such information

is used in a process akin to “spell-checking”, in which a number of compliance checks are performed to assess the degree of compliance of a user-defined process against a standard. Compliance checks can identify compliance errors, assist in process specification and prevent non-compliant tasks in a process from being performed.

The next section gives a brief introduction on workflow systems. Section 3 gives a general introduction to the IEC61508 international standard and identifies its key requirements. Following that, section 4 introduces the standard modeling approach. Section 5 describes the use of ontology and its motivation. The Compliance Flow process model, where the compliance checks take place, is described in section 6. Section 7 discusses the different types of checks that can be performed by Compliance Flow. A case study, which shows how Compliance Flow supports the management of an IEC61508 compliance project, is discussed in section 8, with appropriate conclusions following.

2. Workflow Management System

A workflow is an automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules (WfMC, 2000). The essential workflow characteristics are persons, tasks/activities, application tools and resources (Marshak, 1994; 1997). The (role-playing) persons perform tasks using application tools that provide access to various shared information resources. This characterisation of workflow is shown in Figure 1.

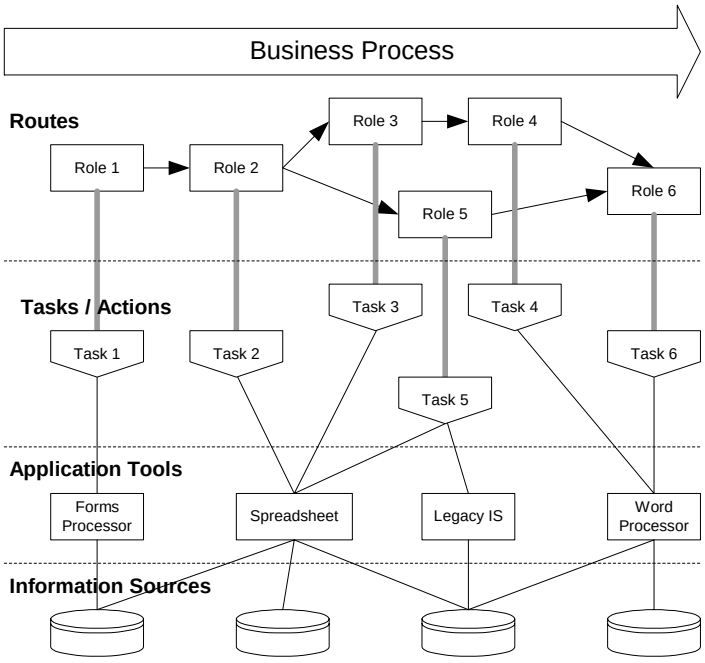

Figure 1 Characterisation of workflow (from Marshak, 1994).

A Workflow Management System (WfMS) is a system that provides computer-based support for the task of workflow management. It supports the specification (build-time functions), execution (run-time

control functions), and dynamic control of workflows involving humans and information systems (run-time interactions) (McCarthy & Sarin, 1993).

Figure 2 shows an outline of a workflow management architecture. The process analysis, modelling and definition tools facilitate the specification of the components of a workflow as a process definition. The workflow enactment service enacts a process definition by assigning tasks to humans and software systems while also maintaining the constraints between tasks. The workflow control data represents the dynamic state of the workflow system and its process instance, which is managed and accessible by the workflow management system or workflow engine. The workflow-relevant data is used by the workflow management system or workflow engine to determine the state transitions of the workflow instance. The application data is used strictly by applications supporting the process instance.

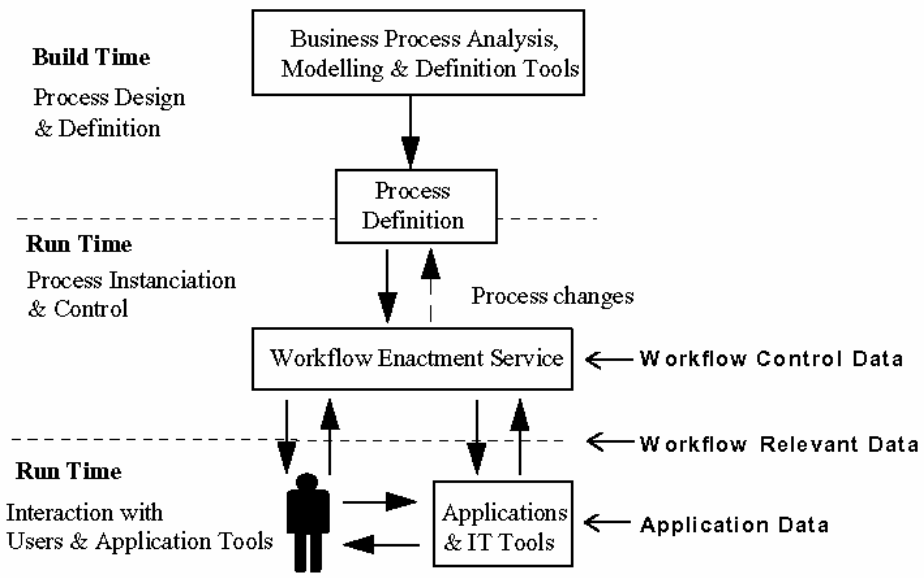

Figure 2. Workflow system architecture and data structure (from WfMC, 2000a).

An engineering project can be treated as a workflow and the project plan can be seen as its process model. A workflow enactment service provides sophisticated support to enforce the steps of a project to be performed exactly as specified in the project plan. International standards are generic and can also be seen as workflow models. Each application of a standard is different because of the difference in project details. The standards should be seen as a reference that functions as a guideline for project planning and process execution.

Current computing-based compliance management solutions provide support only at the document level (Emmerich et al, 1999). The compliance issues are treated in similar ways to inconsistency management in specifications. Key contributions in this area include (Easterbrook et al., 1994) and (Finkelstein et al., 1994). In these approaches, the deliverables required by standards are precisely modelled. The model is used in compliance checking against the user-created documents when they become available – normally at late project stage. If incompliance is detected then part of the process may need to be re-done.

However, compliance checking of workflow models against specified standards is missing in current workflow management technologies; this is because the concept of a reference model is not implemented.

3. IEC61508 International Standard

IEC61508 (IEC, 1997), hereinafter referred to as "IEC61508", is an international standard for the development of Electrical/Electronic/Programmable Electronic Systems (E/E/PEs) that are used to perform safety functions. There are two distinct groups of sub-systems considered in IEC61508: the Equipment Under Control (EUC) and the safety-related system. The former performs the required manufacturing, process, transportation, medical or other activities, while the latter provides the safety functions required to ensure that the EUC is suitably safe. The scope of IEC61508 is limited to the functional safety of the E/E/PEs and is concerned with hazard analysis on the EUC in order to identify requirements for the safety-related system. Design issues relating to the EUC are not included.

3.1 Structure of IEC61508

IEC61508 consists of three main parts, together with additional definitions and guidance. Part 1 specifies the general requirements that are applicable to all parts of IEC61508. In addition, it provides an introduction to the safety lifecycle and the overall structure for the technical requirements for safety-related E/E/PES. Part 2 presents a safety lifecycle for the hardware aspects of E/E/PEs. This part also contains the requirements for the design, test and validation of the hardware used in safety-related E/E/PEs. Part 3 specifies the requirements for software. It also identifies the measures and techniques for specific software development activities within the lifecycle.

IEC61508 has two important concepts which are fundamental to its application, namely Safety Lifecycle and Safety Integrity Level (SIL). A Safety Lifecycle is used, as the key framework, to deal in a systematic manner with all the activities necessary to achieve the required SIL.

3.2 Safety Lifecycles

The Overall Safety Lifecycle proposed by IEC61508 is shown in Figure 3, which is a simplification of reality and as such do not show all the iterations relating to specific phases or between phases. However, the iterative process is an essential and vital part of development throughout the Safety Lifecycles.

IEC61508 requires that the Overall Safety Lifecycle shall be followed (or concluded) by a verification activity, and that this verification shall be planned in a specific way. The verification techniques to be applied will depend upon the specific phase but shall include such things as document and design reviews, testing and analysis of results.

In IEC61508, safety requirements are defined as safety functions together with a safety integrity level for each function. The safety integrity is the probability of a safety-related system performing the required safety function under all stated conditions within a stated period of time. The Safety Integrity Level (SIL)

is one of four discrete levels for specifying the safety integrity requirements. Safety integrity levels are numbered from 1 to 4 with 4 having the highest level of safety integrity.

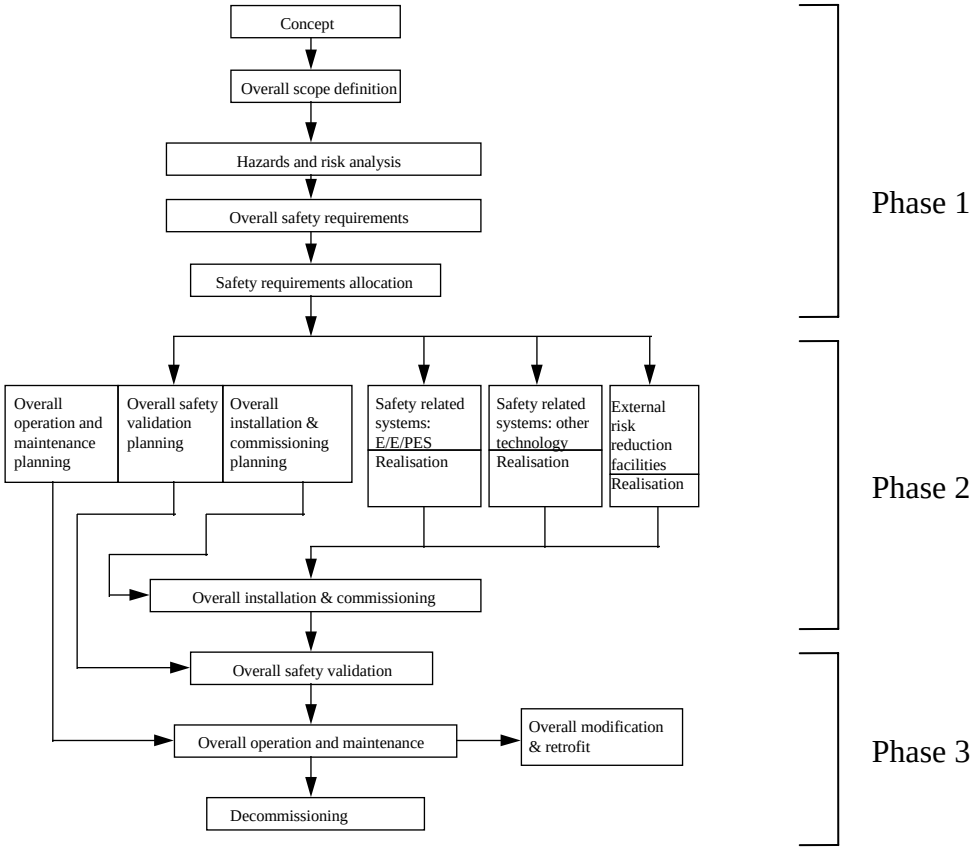

Figure 3. IEC61508 Overall Safety Lifecycle (from IEC, 1997)

3.3 Requirements for Development Safety Equipments

As with other standards relating to safety-related systems, IEC61508 defines a set of requirements for developing E/E/PES for use in safety-related systems. Implementing such requirements is the current best practice for developing high quality products. The key requirements (ERA Technology, 2000) are shown below under different headings.

3.3.1 Quality and Safety Management Requirements

Requirement 1: The development shall be performed in accordance with a defined quality and safety plan. A clearly defined quality plan and a separate safety plan are required for complex safety-related systems. These must identify all development activities, the methods and associated procedures to be followed in performing the identified activities, and the allocation of responsibilities for performing the identified activities. The purpose of the quality plan is to facilitate the overall development process management. The process of overall development can be reviewed at any point during the development and misunderstandings of identified activities among performers can be avoided.

Requirement 2: Suitable techniques and measures shall be selected. Suitable techniques, measures and supporting tools shall be selected for each of the identified activities. IEC61508 identifies possible techniques and measures that should be considered for developing E/E/PES with different safety integrity

level targets. The basic principle in making a selection is that the use of the selected techniques, measures and tools should reduce the risk to a level that is "As Low As Reasonably Practicable" (ALARP).

Requirement 3: The project organisation and allocated responsibilities shall be defined. The project organisation should be clearly defined in terms of the process of interactions between the individual members of the organisation and the allocation of responsibilities. The persons who have responsibilities for any safety lifecycle activities should be competent to discharge those responsibilities. In particular, it should be clear as to how the responsibilities for addressing the safety of the equipment will be shared amongst the members of the organisation.

Requirement 4: Configuration management and change control procedures shall be defined. Procedures for managing changes, both during the initial development of the equipment and subsequent design corrections and enhancements, should be clearly defined and understood by all staff involved in the development process.

Requirement 5: Design reviews shall be planned and carried out. The activities associated with design review should be included in the quantity plan.

Requirement 6: Documents shall be produced. The results of all development and review activities should be clearly documented and presented in an auditable form.

3.3.2 Safety Requirements

Requirement 7: Hazard analysis and risk assessment shall be performed. The hazards associated with the safety-related system application and the associated risks should first be identified in order to derive safety requirements. The resulting requirements should then be analysed in the context of the identified hazards and risks to ensure that they are correct and complete with regard to all possible eventualities.

Requirement 8: The specification of safety requirements shall be documented. The safety requirements should be clearly specified in terms of the required functionalities, using appropriate notation. The safety requirements specification should also include a definition of a required safety integrity level, which is based on the results of the hazard and risk analysis activities.

Requirement 9: There shall be clear traceability from requirements through design. The design of a safety-related system should be carried out in a manner that shows explicitly how the safety requirements are implemented.

3.3.3 Architecture Requirement

Requirement 10: Appropriate programmable electronics architecture shall be selected. The suitable E/E/PES architecture should be designed based on the required safety integrity level of the design.

3.3.4 Design Requirement

Requirement 11: Suitable design techniques shall be employed depending on the required safety integrity level. The design of the E/E/PES should be performed in accordance with the quality and safety plan in which the methods, techniques and measures to be adopted are specified.

3.3.5 Test and Analysis Requirements

Requirement 12: Test specification shall be prepared prior to testing. The tests to be performed on the design should be specified in parallel with the actual design activities, and should be clearly documented.

Requirement 13: The results of test and analysis activities shall be recorded. The results of test and analysis activities shall be documented for subsequent review and audit.

3.3.6 Independent Safety Assessment Requirement

Requirement 14: The development shall be subjected to independent safety validation and assessment. The development process, and the results of that process, should be subjected to review, performed by a suitable competent individual or organisation that is independent of the development team.

4. Standard Modelling

In Compliance Flow, a standard with which compliance by a development process is required is modelled and represented as a Model of Standards (hereinafter referred to as "the Model"). The Model acts as a knowledge-base to provide the required information that will be used in a “spell-check” process to support compliance checks. Figure 4 is a meta-model of standard modelling, demonstrating the three important aspects of a standard in terms of workflow management.

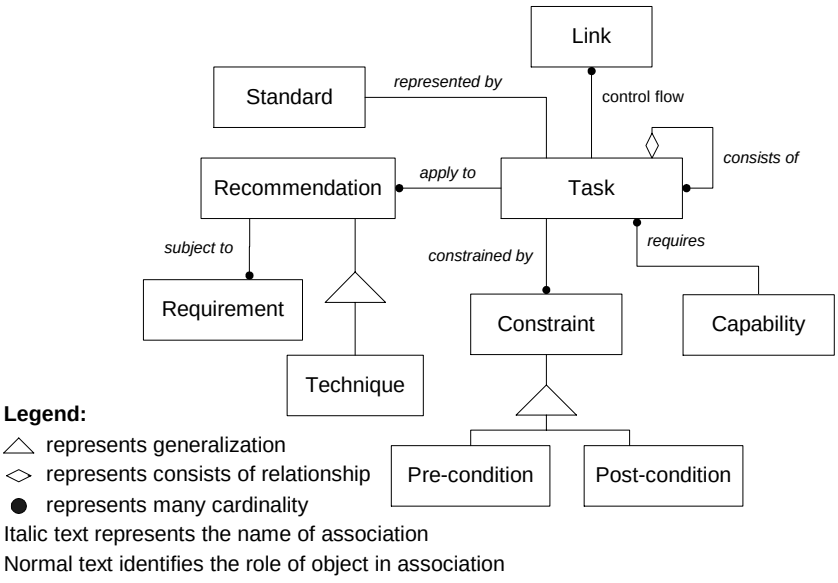

Figure 4. Meta-model of standard modelling.

4.1 Modelling of Task Framework

The task framework in the Safety Lifecycle proposed by IEC61508 is modelled into a hierarchical task network (HTN). A task is a basic unit of work, which can be hierarchically decomposed into sub-tasks if necessary until the required details are modelled, as long as the parent-child relationship between tasks is maintained. Each task is associated with two sets of conditions: pre-conditions and post-conditions. The post-conditions of a task are sometimes the pre-conditions of its subsequent tasks. Performing a task requires the fulfilment of its pre-condition, and to do so, the preceding tasks that can satisfy those conditions with their post-conditions must be completed successfully in advance. The post-conditions of a task will be achieved when the task is completed successfully. Therefore the order of the execution of tasks is constrained by their dependencies.

IEC61508 views the requirements simply as the input to a distinct stage in the lifecycle, and the design specification as the output of that stage. The requirements and specifications are equivalent to the pre-conditions and the post-conditions respectively as they have to be achieved under the recommended sequence in order to comply with IEC61508. A condition is presented in the form of checklists, and it is stated as fulfilled when the checklist is completed.

4.2 Modelling of Task Recommendation

The recommended techniques, measures, tools or methods that have to be used for performing specific tasks to achieve the specified objectives are modelled with four parameters: (1) the task for which the technique is applied, (2) the requirements for applying the technique, (3) the technique itself and (4) the level of recommendation. The value of the second parameter can be null, implying that no requirement is necessary to apply the technique.

IEC61508 introduces sets of techniques for specific development activities with different levels of applicability according to the SIL of the product to be developed. The SIL is normally achieved after the safety requirements are addressed. Therefore, the level of SIL (from 1 to 4) becomes the requirement for applying the recommended techniques. These techniques are categorised into four levels of recommendation in IEC61508 namely Highly Recommended (HR), Recommended (R), No Recommendation (-) and Not Recommended (NR).

A model can be developed that incorporates such recommendations. For example, IEC61508 suggests that the technique ‘Structure Methodology’ (parameter 3) is Highly Recommended (parameter 4) during the achievement of the objective of Clause B.30 (parameter 1) when the SIL of the product being developed is equal to 1 (parameter 2).

4.3 Modelling of Task Capability

A task capability refers to the required capabilities of an agent who is going to perform a specific task. The modelling of task capability adopts the same approach as the modelling of agent capability. A

capability consists of two parts: the technical capability and its application area. Two ontologies are used in the modelling of a task capability, namely capability ontology and application ontology. While the capability ontology defines the required techniques, skills, roles and other attributes of an expected agent, the application ontology defines their application areas. The terms in an ontology are organised into a hierarchical structure, which provides a hierarchy of capabilities. This hierarchical structure can ease the process of specifying capabilities since specifying a high-level capability implies that all its lower levels are covered.

The task capability is normally implicitly specified in most of the industry standards. A standard can be applied to wide areas where the situations are different and therefore agents with different capabilities will be involved in performing similar tasks but in different projects or in different companies. Standards normally give the guidelines for agent selection rather than details of required capability. For example, IEC61508 gives a number of competence factors that should be addressed when assessing and justifying the competence of persons to carry out their duties, including:

> *"(1) Engineering appropriate to the area; (2) Engineering appropriate to the technology; (3) Safety engineering appropriate to the technology; (4) Knowledge of the legal and safety regulatory framework; (5) The consequences in the event of failure of the safety-related systems. The greater the consequences the more rigorous shall the specification and assessment of competence be; …" (IEC, 1997)*

Thus, the task capabilities of a standard have to be defined by users according to their understanding of these factors and the particular environment where the system is running. Factors such as the nature of the product, the country where the product will be consumed and differences in organisational culture can affect the capability specification. The required capability for the same process in different projects may vary because of different situations. For example, different techniques may require different levels of SIL. The group of required capabilities together with the suitable weights for a particular situation can be collected together and saved as a scheme. It is possible for a task to be assigned multiple schemes. As a result, users can choose suitable schemes of capability during runtime.

5. The Use of Ontology

As a user-defined process has to be checked against the Model, the terms used in both sides in describing a concept of interest must be consistent, and this is achieved by the use of an ontology. This is a data model that "consists of a representational vocabulary with precise definitions of the meanings of the terms of this vocabulary plus a set of formal axioms that constrain interpretation and well-formed use of these terms" (Campbell & Shapiron, 1995). An ontology is therefore an explicit representation of a "… shared understanding of some domain of interest …" (Uschold & Gruninger, 1996). It can be shared and re-used by others in the same domain to minimise ambiguity.

The advantage of using ontologies in workflow management is discussed in Moore et al. (1999). A number of ontologies are used in describing the entities in both the Model and the user-defined processes. The terms of an ontology are organised into a hierarchy in which abstract terms are located at a higher level while lower level terms represent more concrete objects. The user is able to create, change, remove or extend the terms on an ontology to suit a particular application.

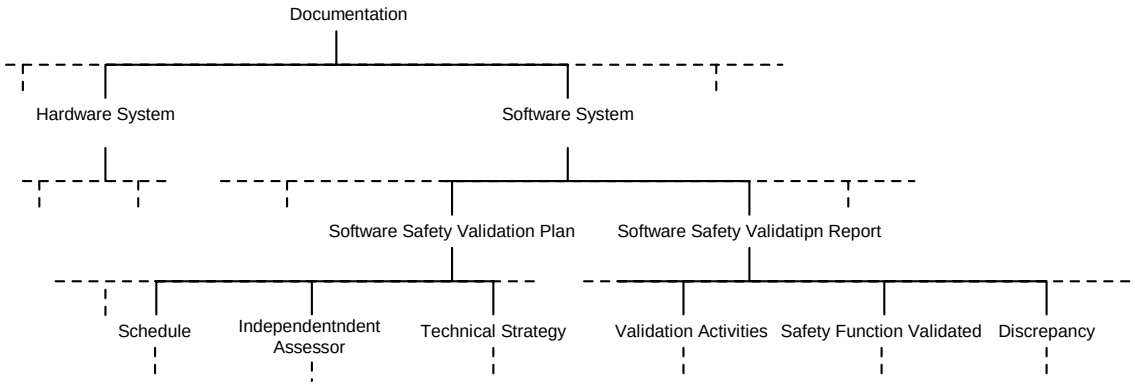

Figure 5. An example of documentation ontology.

An excerpt from the documentation of an ontology for safety-related systems is given in Figure 5. Based on the figure, a software safety validation report should include: (1) a chronological record of the validation activities, (2) the safety function being validated and (3) discrepancies between expected and actual results, and other information. Each of these sub-items can be defined as an individual sub-specification and is to be achieved through a particular task.

6. Process Model

A user-defined process (UDP) is captured through an enhanced activity-based modelling framework following an enhancement of the structures outlined in section 4.1. The meta-process model is given in Figure 6. Task decomposition can be performed to any level of detail. The tasks at the lowest level are concrete work instructions, while the tasks at higher levels are more abstract.

As discussed in section 4.1, task execution is constrained by the pre- and post-conditions with an added type of pre-condition being the *technique* that has to be used to carry out a task. The execution sequence of tasks is modelled through links. A *link* represents the connection between two tasks and the execution order. The input and output links of a task have to be maintained in task decomposition.

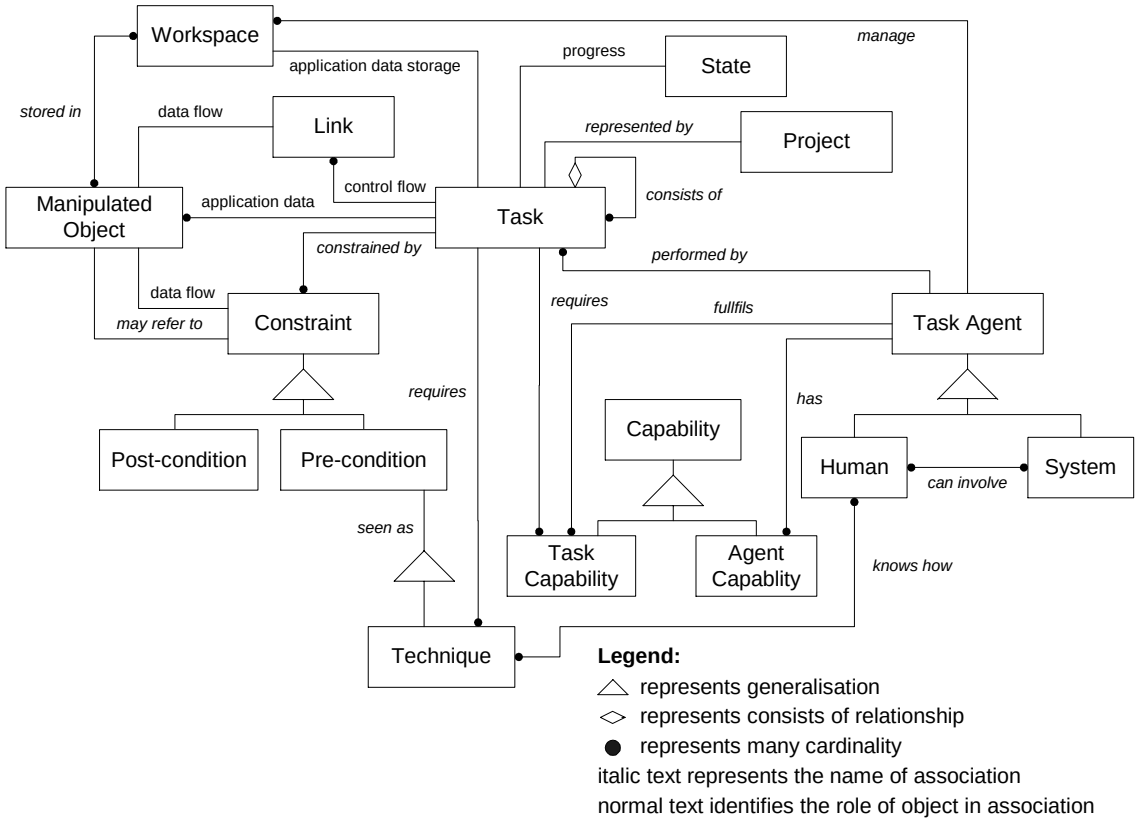

Figure 6. Meta-process model.

A task is performed by a *task agent* and the progress of the execution of the task is represented through a number of *states*. A task agent could be a *human* or a *system*; a system may need a human to operate it. A *capability* is a skill, technique, method, knowledge or any attribute that a task agent requires in order to perform a task. While the required capability of a task agent to perform a particular task is defined as the *task capability*, the *agent capability* is used to describe the capability possessed by an agent. The specifications of task capabilities and agent capabilities enable automatic identification of the most suitable agent for a task.

Every task is associated with a *workspace* that is used to store the *information objects* related to that task, including the documents corresponding to the pre- or post-conditions of the task. A workspace associated with a high-level task contains all the information objects of its child tasks. The agent assigned to that task manages the workspace. Information objects can be copied or moved from one workspace to another. The information objects in a workspace are hyperlinks that link to the concrete objects stored in a secured space, such as an FTP server. Therefore, only one physical copy of each document is stored. Cheung et al. (2002) provides further details of using this process model to manage dynamic processes.

7. Compliance Checks

Compliance checks are for checking a UDP against the Model to identify compliance errors and assist the user in specifying a process that meets the requirements of a selected standard. Compliance checks can

be grouped into two categories namely (1) Error identification and prevention and (2) Process management assistance.

Error identification and prevention:

a) Correctness Check – to ensure the sequence of tasks specified in a UDP is in accordance with a selected standard.
b) Completeness Check – to ensure all the required tasks within a standard are defined in the UDP.
c) Capability Check – to ensure the required capabilities of an agent are specified according to a selected standard.
d) Recommendation Check – to ensure the recommended techniques, measures, tools or methods for performing a particular task are fully considered.

Process management assistance:

a) Planning Assistant – to remind the user of possible pre-conditions, post-conditions, recommendations and required capability of a particular task while it is being planned.
b) Information Navigation – to transfer the required information, once it is available, to the tasks where the information may be used for their execution.
c) Cross-Referencing – to refer a user-defined task (UDT) to the relevant part of a standard or vice versa.

Process planning normally starts at an abstract level and becomes more concrete as planning progresses. Compliance checks can be applied at any stage during process planning. The checks can also be applied to a task at any level of an HTN. If the selected task in the UDP is a high-level task, then its child tasks will be checked. A high level task is standard-compliant if all of its child tasks pass the compliance checks. The levels of abstraction between the UDP and the Model may vary, with three possible situations:

1. A UDP is at the same level of abstraction as the Model.
2. A UDP is at a higher level of abstraction than the Model.
3. A UDP is at a more detailed level than the Model.

Compliance Flow will perform checks on a process or a task in response to a request from the user at build-time, but it will automatically perform them before the execution of a task at run-time. The behaviour of a compliance check during build-time and run-time may vary. In the following section, compliance checks are described based on the three situations identified above and the different behaviours during build-time and run-time are identified and discussed.

7.1 Identifying the Corresponding Specifications and Tasks

IEC61508 views specifications as outputs of tasks and an output specification from one task can be an input requirement of another task in the same process. Specifications are modelled as post conditions of tasks and they are represented using a document icon in this paper because specifications are normally described in documents. A task specification corresponds to a standard specification if they have the same concern. The two corresponding specifications will have the same name because of the use of ontology. Two tasks that produce corresponding output specifications are corresponding tasks, but the tasks can have different names. Therefore, all the specified constraints of a standard task, including pre- and post-condition, capability and information must be applied to its corresponding task in the UDP in order to comply with the standard.

A specification may partially correspond to another specification. This relationship is captured by the parent-child relationship between two specification terms in the Documentation Ontology. There are two possibilities: (1) a standard task corresponds to multiple tasks in a UDP, and (2) a UDT corresponds to multiple tasks in the Model. For the first situation, all the constraints specified with the standard task must be applied to its corresponding tasks in a UDP. Similarly, for the second situation, the constraints associated with the standard tasks must be applied to the corresponding task in the UDP. However, some cases will override this general rule and these will be discussed in particular compliance checks. A common but important function in all the compliance checks is to identify the corresponding specifications and tasks in the Model in order to retrieve the required information for checking.

In the figures that follow, the rounded rectangles with identifiers beginning with the letters ‘ST’ represent standard tasks and identifiers that begin with the letter ‘T’ represent user-defined tasks. The document icons represent the pre- and post-conditions of tasks. The dotted lines represent the corresponding relationships between standard specifications and task specifications. The arrows on the dotted lines represent the mapping directions for identifying the corresponding specifications and tasks in the examples.

If a UDP and the Model are at the same level of abstraction, the task specification and the corresponding standard specification can be matched directly as they have the same name. If the corresponding standard specification cannot be matched directly, it could be because the UDP and the Model are at different levels of abstraction. The system will first try to establish whether that UDP is more detailed than the Model. To do so, its parent terms are matched with the Model, starting from the immediate parent. As an example using Figure 7, to identify the corresponding standard specification of B2.1.1, its parent terms are matched with the Model in the order B2.1, B2 then B. In this example, B2.1 can be located.

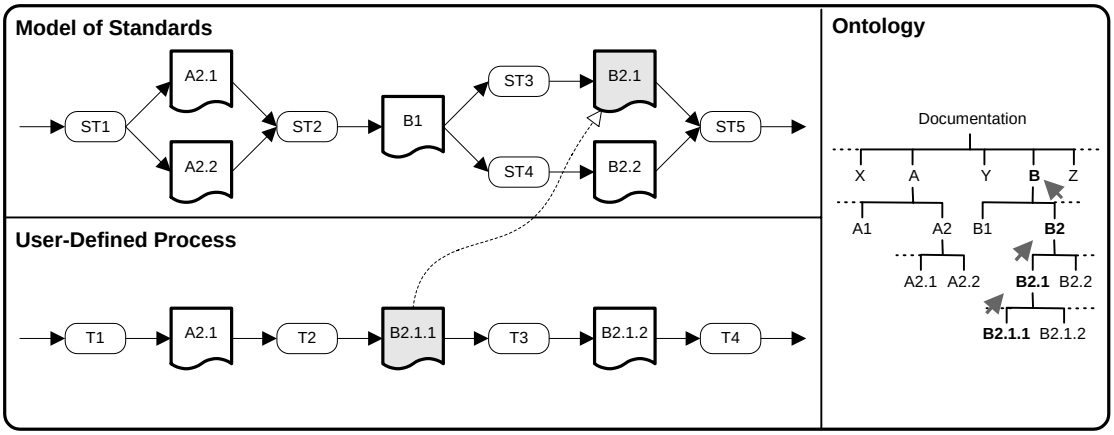

Figure 7. An example of a UDP at a more detailed level than the Model.

If no parent terms can be matched, then its child terms will be tried. As an example, in Figure 8, three of the child terms of B, i.e. B1, B2.1 and B2.2, matched the Model and are therefore considered the corresponding child specifications of B.

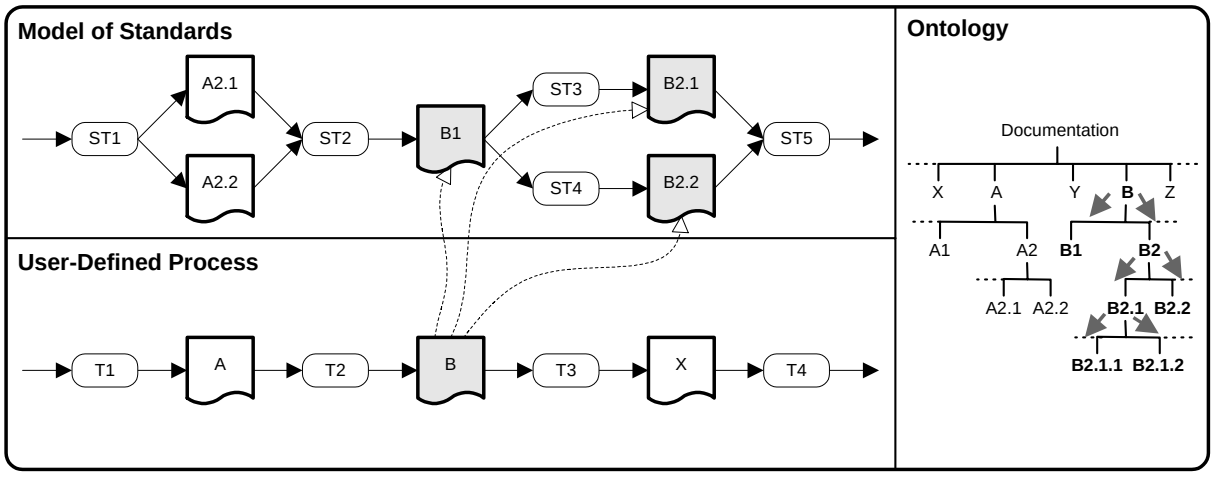

Figure 8. An example of a UDP at a higher level of abstraction than the Model.

If a specification or any of its parent and child terms cannot find a match in the Model, then it is a user-defined specification, and it is not a concern of the compliance checks. An example is the specification X in Figure 8.

7.2 Correctness Check

The Correctness Check is to verify that the placement of a task specification complies with the Model, and therefore to ensure that the execution sequence of the tasks in the UDP is correct. In order to achieve this we firstly need to identify its corresponding standard specification in the Model. The second step is to ensure its immediate previous specification is also located in front of the corresponding standard specification. If this is the case then the task specification is placed correctly. If the immediate previous specification is a user defined specification, then the next previous specification will be matched instead.

As an example, consider Figure 9, where the goal is to check the compliance of the task specification 'Software Safety Validation Report'. The first step is to identify its corresponding standard specification in the Model. Its previous specification is then matched against 'Software Safety Validation Plan' in the Model. The matching of X failed as it is a user-defined specification. The previous specification of X, 'Software Safety Validation Plan', is then tried. It is found in the Model and is placed in front of the corresponding standard specification. Therefore, it can be claimed that the specification 'Software Safety Validation Report' is placed in the right order.

If a UDP is at a higher level of abstraction than the Model, then the specifications cannot be matched directly. The identification of two corresponding specifications at different levels of abstraction is described in section 7.1.

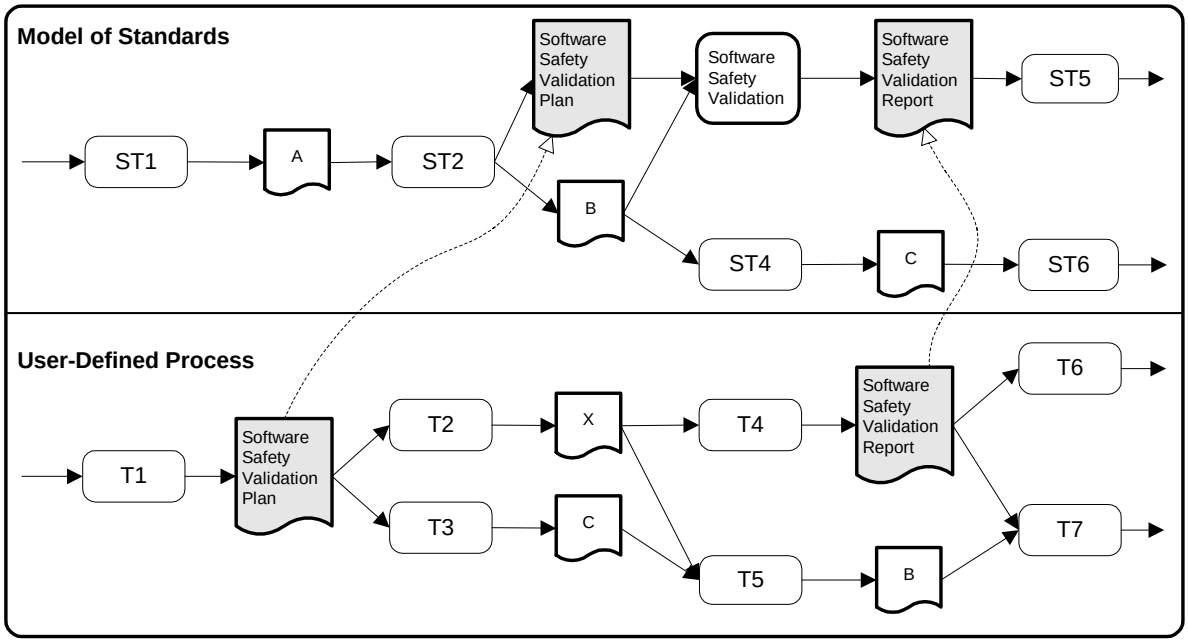

Figure 9. Correctness Check when a UDP and the Model are at the same level of abstraction.

When applying the Correctness Check to a task, all the specifications of the task will be checked. A task is compliant if all of its specifications pass the Correctness Check. Similarly, a process is compliant when all of its tasks are compliant. The steps required for carry out the Correctness Check are shown in the algorithm in Figure 10.

```
Function CorrectnessCheck(spec, Std) as Boolean
      spec: A specification in a user-defined process for which
            the correctness check is applied to.
      Std: The Model of the selected standard.
      ImSpec{…} = {x:x ∈ Standard Specification};
      ImSpec = GetImmediatePreviousStandardSpecification(spec,Std);
      If ImSpec ≠ 0 Then
            PvTSpec{…} = {x:x ∈ Task Specification};
            PvTSpec = GetPreviousTaskSpecifications(spec);
            If (ImSpec ⊆ PvTSpec) Then
                  Return True;
                  Exit Function;
            End If
            PImSpec{…} = {x:x ∈ Standard Specification};
            For each i ∈ ImSpec
                  PImSpec = GetParent(i);
                  If ((PImSpec ∩ PvTSpec) ≠ 0) Then
                        ImSpec = ImSpec - {i}
                  End If
            Next i
            CImSpec{…} = {x:x ∈ Standard Specification};
            For each i ∈ ImSpec
                  CImSpec = GetChildren(i);
                  If ((CImSpec ∩ PvTSpec) ≠ 0) Then
                        ImSpec = ImSpec - {i}
                  End If
            Next i
            If ImSpec ≠ 0 Then
                  Return False
            Else
                  Return True;
            Endif
      Else
            Return True;
      End If
End Function
```

Figure 10. Correctness Check algorithm.

7.3 Completeness Check

The Completeness Check is to ensure that the required specifications are included in a UDP and therefore ensures that the required tasks are also included. It can be applied to the whole process or a sub-process at build-time as long as the corresponding part in the Model is given. For example, to check the completeness of the UDP part that is related to the part of software development in IEC61508, the corresponding sub-process entitled ‘Safety Software Design and Development’ in the Model must be identified. Selecting an inappropriate corresponding standard sub-process will lead to either a long list of missing specifications or some of the required specifications not being checked.

In the Completeness Check, all the standard specifications in the given standard process are matched against the task specifications in the UPD. If all the specifications are found, then the Completeness Check succeeds. In Figure 9, the Completeness Check fails because specification A is not in the UDP. To pass a Completeness Check during run-time, the required specifications to perform the UDT must be ready. The required specifications can be retrieved from the corresponding task in the Model. The user

does not need to identify a corresponding part in the Model.

Figure 11 shows an example in which user-defined tasks T1 and T2 are completed and the required specifications X and B for T3 are ready. ST3 is the corresponding task of T3 as both have the same post-condition. The Model dictates that a specification C is required for T3 to produce a specification D but it is not specified as a pre-condition of T3 in the UDP. Therefore, the Completeness Check applied to T3 fails.

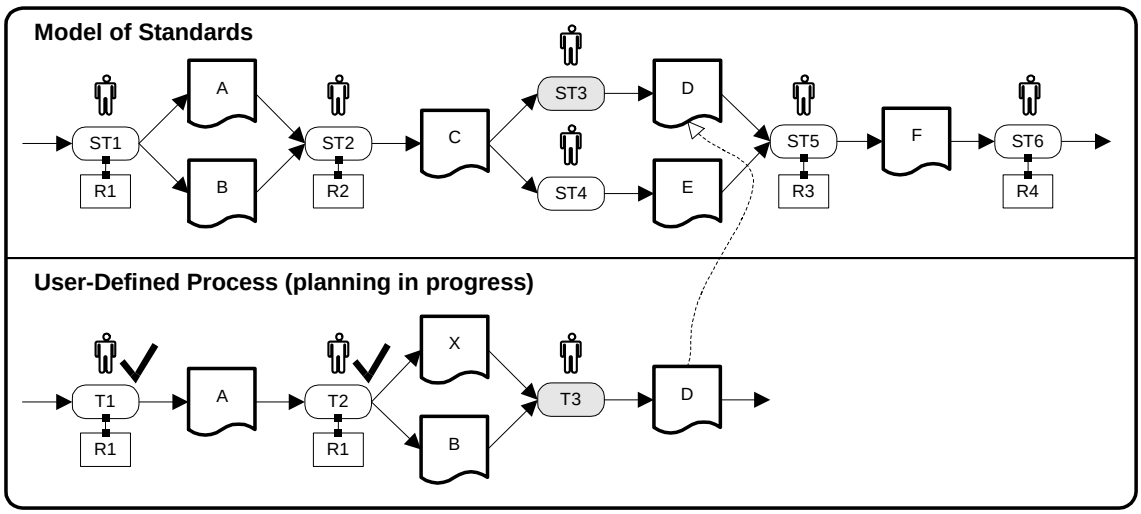

Figure 11. An example of applying a Completeness Check to a task.

The algorithms for the Completeness Check are provided in Figure 12 (process build-time) and Figure 13 (run-time).

```
Function CompletenessCheck(Pros, SPros) as Boolean
      Pros: A user-defined process consisted of a set of tasks
            for which the completeness check is applied to.
      SPros: A corresponding process in the Model of a
            selected standard.
      TaskSpec{…} = {x:x ∈ Task Specification};
      StdSpec{…} = {x:x ∈ Standard Specification};
      TaskSPec = GetSpecifications(Pros);
      StdSpec = GetSpecifications(SPros);
      For each i ∈ StdSpec
            If (i ∈ TaskSpec) Then
                  StdSpec = StdSpec - {i};
            Else
                  PTaskSpec{…} = {x:x ∈ Task Specification};
                  PTaskSpec = GetParent(i);
                  If ((PTaskSpec ∩ StdSpec) ≠ 0) Then
                        StdSpec = StdSpec - {i};
                  Else
                        CTaskSpec{…} = {x:x ∈ Task Specification};
                        CTaskSpec = GetChildren(i);
                        If ((CTaskSpec ∩ StdSpec) ≠ 0) Then
                              StdSpec = StdSpec - {i};
                        Else
                              Return False;
                        End If
                  End If
            End If
      Next i
      If (StdSpec ≠ 0) Then
            Return False;
      Else
            Return True;
      End If
End Function
```

Figure 12. Completeness Check algorithm used at process build-time.

```
Function CompletenessCheck(task, SPros) as boolean
      task: A user-defined task which user is trying to start.
      SPros: A corresponding process in the Model of a
            selected standard.
      PvTaskSpec{…} = {x:x ∈ Task Specification};
      PvTaskSPec = GetPreviousSpecifications(task);
      TaskSpec{…} = {x:x ∈ Task Specification};
      TaskSpec = GetSpecifications(task)
      CorStdSpec{…} = {x:x ∈ Standard Specification};
      For each i in TaskSpec
            CorStdSpec = CorStdSpec ∪ _
            GetCorresdpondingStdSpecifications(i,SPros);
      Next i
      StdTask = {x:x ∈ Standard Task};
      For each i in CorStdSpec
            StdTask = StdTask ∪ GetAssociatedTask(i);
      Next i
      StdSpec = {x:x ∈ Standard Specification};
      For each i in StdTask
            StdSpec = StdSpec ∪ GetPreviousSpecification(i);
      Next i
      For each i in StdSpec
            If (i ∈ PvTaskSpec) Then
                  StdSpec = StdSpec - {i};
            Else
                  ParentTaskSpec{…} = {x:x ∈ Task Specification};
                  ParentTaskSpec = GetParent(i);
                  If ((ParentTaskSpec ∩ StdSpec) ≠ 0) Then
                        StdSpec = StdSpec - {i};
                  Else
                        ChildTaskSpec{…} = {x:x ∈ Task Specification};
                        ChildTaskSpec = GetChildren(i);
                        If ((ChildTaskSpec ∩ StdSpec) ≠ 0) Then
                              StdSpec = StdSpec - {i};
                        Else
                              Return False;
                        End If
                  End If
            End If
      Next i
      If (StdSpec ≠ 0) Then
            Return False;
      Else
            Return True;
      End If
End Function
```

Figure 13. Completeness Check algorithm used at process run-time.

7.4 Capability Check

The Capability Check is to ensure that a task agent possesses the required capability specified in a standard for performing a task. A Capability Check will return a goodness of fit (GOF) value indicating how well an agent matches the required capabilities. Each GOF is assessed through a fuzzy matching algorithm developed in the Compliance Flow project. Assessing GOF requires two sets of capabilities: the required capability specified in the Model and the agent capability stored in a database. The Capability Check is an example of where the relative levels of abstraction of the UDP and the Model have an influence on the type of check required and these differences are now described.

7.4.1 UDP and the Model at the Same Level of Abstraction

If a UDT is at the same level of abstraction as its corresponding standard task then the standard capability for the corresponding standard task is the required capability for the user-defined task.

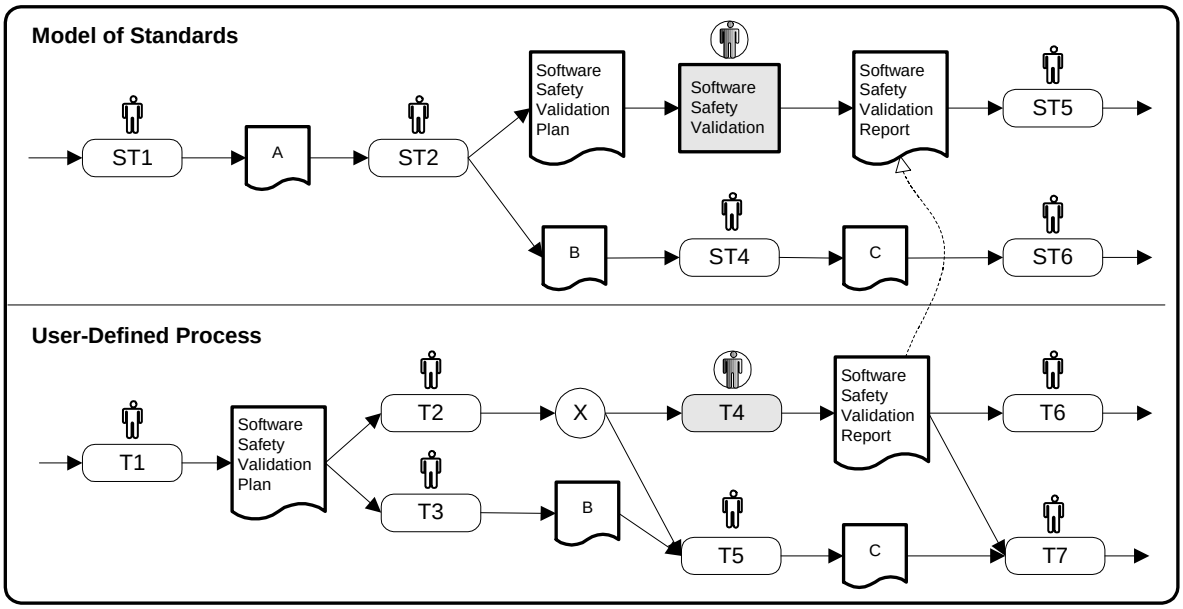

Figure 14. Capability Check when a UDP and the Model are at the same level of abstraction.

As an example using Figure 14, the Capability Check is applied to the UDT T4. The required capability can be retrieved from its corresponding standard task, 'Software Safety Validation', in the Model. The agent capability is retrieved from the database. These two sets of capability are sent to the fuzzy capability matching algorithm to assess the GOF values.

7.4.2 UDP at a Higher Level of Abstraction than the Model

If a UDT is at a higher level of abstraction than the Model, then it may correspond to multiple standard tasks. The collection of the capabilities of these standard tasks will be treated as the required capabilities for the user-defined task. In Figure 15, a Capability Check is applied to the UDT T2 which has three corresponding standard tasks, ST2, ST3 and ST4, in the Model. The specified capabilities of these tasks are the required capabilities for T2.

If a UDP is at a higher level of abstraction than the Model, then the Capability Check is advisory at build-time but is enforced at run-time. At build-time, a Capability Check can be applied to a task at any level of abstraction. A high-level task may be assigned to a member of management staff who is not going to perform the sub-tasks. The sub-tasks will be specified in detail and assigned to suitable technical staff to execute. Consequently, the manager does not need to fulfil the required technical capability specified in the Model. As process planning starts at an abstract level, the workflow engine is unable to determine

whether a task is to be executed or further decomposed. Therefore, the result from the Capability Check is only advisory. However, when a Capability Check is applied to a task to be performed at run-time, a task should not be allowed to proceed if it fails Capability Check.

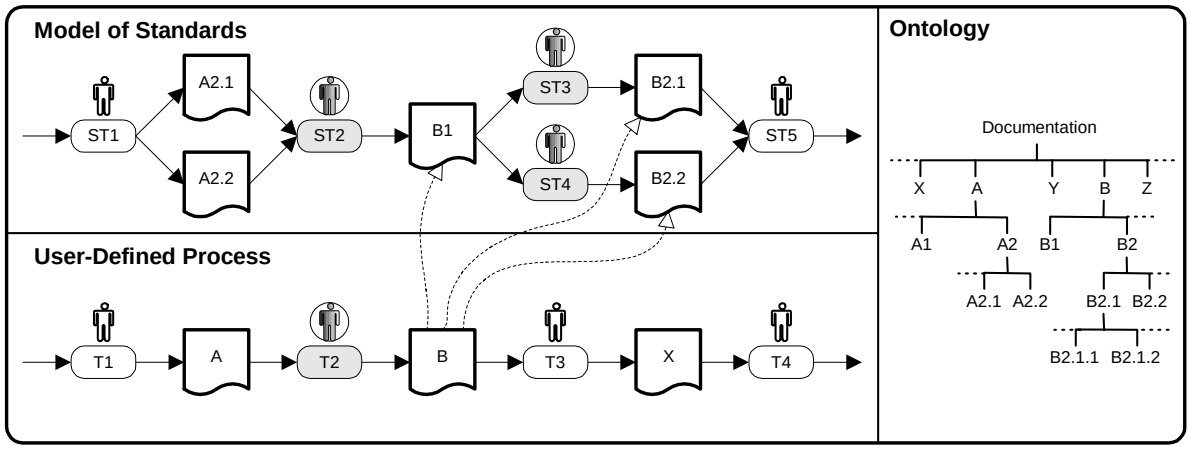

Figure 15. Capability Check when UDP at a higher level of abstraction than the Model.

7.4.3 UDP at a More Detailed Level of Abstraction than the Model

If a UDT is at a more detailed level of abstraction than the Model, then it is concerned only with part of the corresponding standard task. Therefore, the standard capability in the Model may exceed the actual requirement for the user-defined task. Because there is no way of filtering out the exceeding part, the whole standard capability will be considered as the required capability.

The algorithm for performing the Capability Check is provided in Figure 16. A Capability Check can also be applied to a process at build time. To do so, Compliance Flow will perform the Capability Check on every task in the process. The process will pass only when all its tasks passed.

```
Function CapabilityCheck(task, Std) as Integer
      Task: A task in a user-defined process for which
            the capability check is applied to.
      Std: The Model of the selected standard.
      TaskCap{…} = {x:x ∈ Task Capability};
      TaskCap = GetCapability(task);
      TaskSpec{…} = {x:x ∈ Task Specifiation};
      TaskSpec = GetSpecifications(task);
      StdSpec{…} = (x:x ∈ Standard Specification);
      For each i ∈ TaskSpec
            StdSpec = StdSpec ∪ _
            {GetCorrespondingStdSpecification(i,Std)};
      Next i
      StdTask{…} = {x:x ∈ Standard Task}
      For each i ∈ StdSpec
            StdTask = StdTask ∪ {GetAssociatedTask(i)};
      Next i
      RequiredCap{…} = {x:x ∈ Standard Capability};
      For each i ∈ StdTask
            RequiredCap = RequiredCap ∪ {GetCapability(i)};
      Next i
      GOF = AssessGOF(TaskCap, RequiredCap);
      Return GOF
End Function
```

Figure 16. Capability Check algorithm.

7.5 Recommendation Check

A Recommendation Check is employed to ensure that the techniques, measures, tools or methods that are recommended by a standard for performing particular tasks are fully considered by the user in planning a process. IEC61508 recommends techniques and measures for the development of safety-related systems for the control of failures. These recommendations are defined and related to standard tasks in the Model. On the other hand, a task recommendation is modelled as the pre-condition of a UDT in a UDP.

7.5.1 UDP and the Model at the Same Level of Abstraction

When carrying out the Recommendation Check (RC), the pre-condition of a UDT is matched against the standard recommendations retrieved from the corresponding standard task to verify whether the recommended techniques are selected. If any highly recommended (HR) technique is not used or a non-recommended technique is chosen, then an explanation is required and it will be recorded in the Tracking Server.

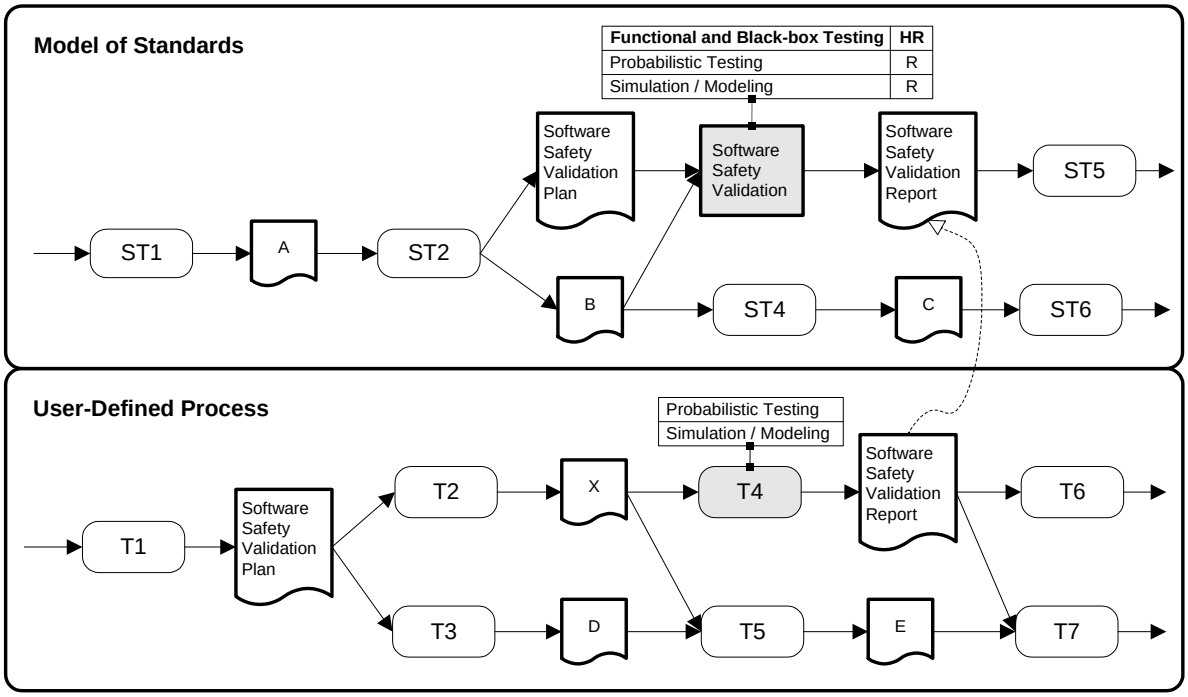

Figure 17. RC when a UDP and the Model are at the same level of abstraction.

In Figure 17, the Recommendation Check is applied to a UDT T4 which is to produce the 'Software Safety Validation Report' where two recommended (R) techniques are selected and defined as pre-conditions. Its corresponding standard task 'Software Safety Validation' has three recommended techniques, one of which is a highly recommended (HR) technique 'Functional and Black-box Testing'. The Recommendation Check fails because the highly recommended technique is not used in T4. The check will pass only when either the reason for not using the technique is provided or the technique is added to T4 as a pre-condition.

7.5.2 UDP at a Higher Level of Abstraction than the Model

If a UDT corresponds to multiple standard tasks in the Model, then all the recommendations related to the corresponding tasks in the Model must be used or considered. If the UDT is to be further decomposed, then the selected recommendations will be kept with the child tasks.

7.5.3 UDP at a More Detailed Level of Abstraction than the Model

When a standard task corresponds to multiple tasks in a UDP, all its recommendations must be considered by the corresponding user-defined tasks.

Figure 18 shows the algorithm for the Recommendation Check.

```
Function RecommendationCheck(task, Std) as Boolean
      task: A task in a user-defined process for which the
            recommendation check is applied.
      Std: The Model of a selected standard.
      TaskSpec{…} = {x:x ∈ Task Specification};
      TaskSpec = GetSpecifications(task);
      StdSpec{…} = {x:x ∈ Standard Specification};
      For each i ∈ TaskSpec
            StdSpec = StdSpec ∪ _
            {GetCorrespondingStdSpecification(i,Std)};
      Next i
      StdTask{…} = {x:x ∈ Standard Task};
      For each i ∈ StdSpec
            StdTask = StdTask ∪ {GetAssociatedTask(i)};
      Next i
      Rcom{…} = {x:x ∈ Recommendation};
      For each i in StdTask
            Rcom = Rcom ∪ {GetRecommendations(i)};
      Next i
      PreCond(…) = {x:x ∈ Task Pre-Condition};
      PreCond = GetPreConditions(task);
      If (Rcom ⊆ PreCond) Then
            Return True;
      Else
            Return False;
      End If
End Function
```

Figure 18. Recommendation Check algorithm.

7.6 Planning Assistance

Planning Assistance (PA) provides the user with the possible related information of the task being planned, including the possible pre-conditions, post-conditions, recommendations and the required capability of the task. The concept of Planning Assistance is that if any pre- or post-condition of a UDT are already specified, then its corresponding standard task in the Model can be identified and the information related to the standard task may be retrieved to assist task specification.

7.6.1 UDP and the Model at the Same Level of Abstraction

If a UDT is at the same level of abstraction as the Model, then the related information is retrieved from its corresponding standard task. In Figure 19, a user is specifying the UDT T3. Its corresponding standard task ST2 can be identified as specification C is given. Planning Assistance will then provide the information related to ST2 to the user for consideration. The possible information includes the pre-conditions, required capabilities and recommendations.

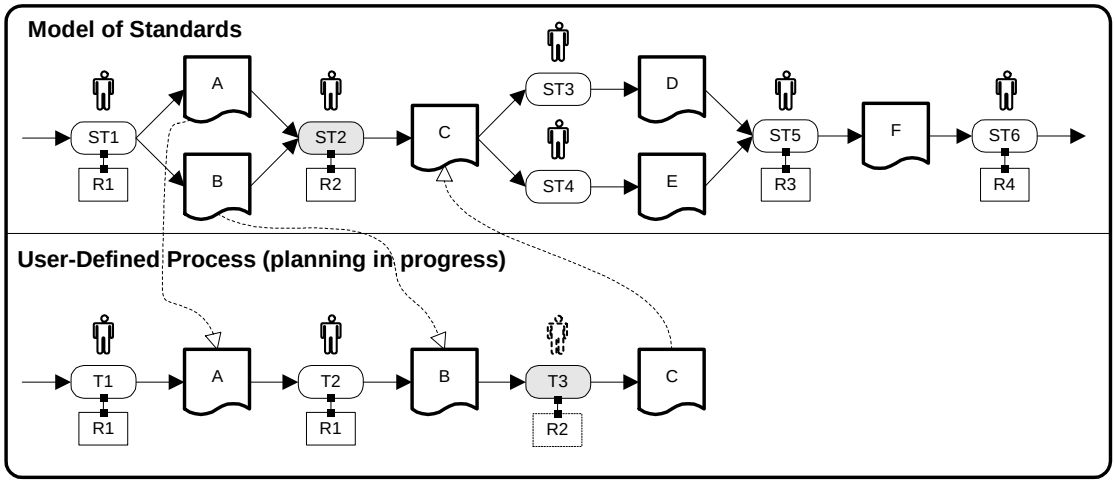

Figure 19. PA when a UDP and the Model are at the same level of abstraction.

7.6.2 UDP at a Higher Level of Abstraction than the Model

If a UDT corresponds to more than one standard task in the Model, all the information from the corresponding standard tasks will be treated as they are related to the user-defined task.

7.6.3 UDP at a More Detailed Level of Abstraction than the Model

If a standard task corresponds to multiple user-defined tasks in a UDP, then all the information from the standard task will be treated as they are required for all the corresponding user-defined tasks.

Algorithms for determining any possible capability and any possible recommendation for a UDT are shown in Figure 20 and Figure 21 respectively.

```
Function FindPossibleCapability(task, Std) as PossibleCap{…}
      Task: A task in a user-defined process for which
            the capability check is applied to.
      Std: The Model of the selected standard.
      TaskCap{…} = {x:x ∈ Task Capability};
      TaskCap = GetCapability(task);
      TaskSpec{…} = {x:x ∈ Task Specifiation};
      TaskSpec = GetSpecifications(task);
      StdSpec{…} = {x:x ∈ Standard Specification};
      For each i in TaskSpec
            StdSpec = StdSpec ∪ _
            {GetCorrespondingStdSpecification(i,Std)};
      Next i
      StdTask{…} = {x:x ∈ Standard Task}
      For each i in StdSpec
            StdTask = StdTask ∪ {AssociatedTask(i)};
      Next i
      RequiredCap{…} = {x:x ∈ Standard Capability};
      For each i in StdTask
            RequiredCap = RequiredCap ∪ {Capability(i)};
      Next i
      PossibleCap = TaskCap ⊄ RequiredCap;
End Function
```

Figure 20. The algorithm to find the possible capability for a user-defined task.

```
Function FindPossibleRecommendation(task, Std) as PossibleRcom{…}
      task: A task in a user-defined process for which the
            recommendation check is applied.
      Std: The Model of a selected standard.
      TaskSpec{…} = {x:x ∈ Task Specification};
      TaskSpec = SetSpecifications(task);
      StdSpec{…} = {x:x ∈ Standard Specification};
      For each i in TaskSpec
            StdSpec = StdSpec ∪ _
            {GetCorrespondingStdSpecification(i,Std)};
      Next i
      StdTask{…} = {x:x ∈ Standard Task};
      For each i in StdSpec
            StdTask = StdTask ∪ {GetAssociatedTask(i)};
      Next i
      Rcom{…} = {x:x ∈ Recommendation};
      For each i in StdTask
            Rcom = Rcom ∪ {GetRecommendations(i)};
      Next i
      PreCond(…) = {x:x ∈ Task Pre-Condition};
      PreCond = GetPreConditions(task);
      PossibleRcom = Rcom ⊄ PreCond;
      Return PossibleRcom;
End Function
```

Figure 21. The algorithm to find the possible recommendation for a user-defined task.

7.7 Information Navigation

The use of workspace enables information to be transferred from one task to another according to the task flow. However, the task flow does not guarantee that all the information has been considered. Consider Figure 22 where, according to the Model, the specification ‘Software Safety Validation Plan’ is required

for the task to produce a 'Software Safety Validation Report'. The UDP is compliant with the standard as both specifications exist and are placed in the correct sequence even though two user-defined tasks are in between. It is unknown whether T2 and T3 require the 'Software Safety Validation Plan' for their execution. However, it is definite that the plan is required for T4 as it is the actual task to produce 'Software Safety Validation Report' but it is not defined as a pre-condition because of a user mistake. Therefore the report will exist only in the workspaces of T1 and T2 when it is ready.

Information Navigation allows the transfer of information, according to the Model, during process run-time to the related tasks where such information may be required for their execution. For a standard specification that is used as an input to a task to produce another standard specification (hereinafter referred to as the output specification), Information Navigation assumes that the input specification will be used for all the tasks in between these two specifications in the UDP. Once a document is uploaded to a task workspace, Information Navigation (IN) will identify the input and output standard specifications in the Model, and then copy the links of the documents to the workspaces of the user-defined tasks in between the corresponding specifications in UDP.

As an example in Figure 22, 'Software Safety Validation Plan' has just been uploaded to the workspace of T1. Here the 'Software Safety Validation Report' is the post-condition of the next standard task. In the UDP, the tasks in between the input and output specifications are T2, T3 and T4. Therefore, links to the document are put in the workspaces of T3 and T4 also.

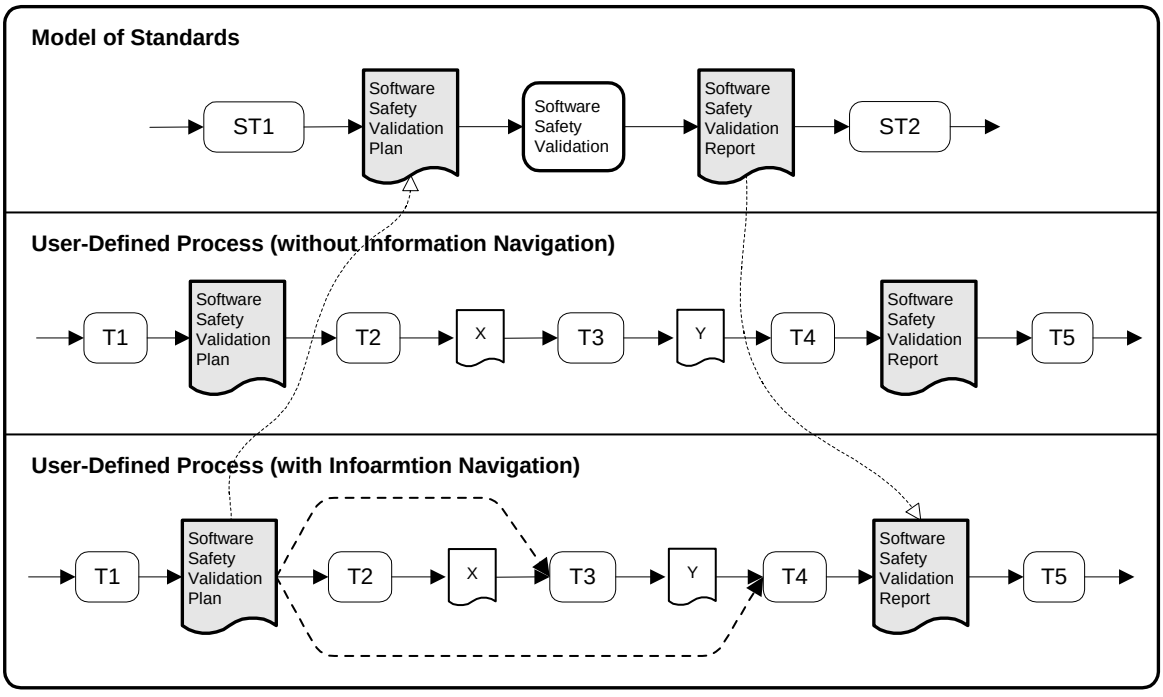

Figure 22. Information Navigation when a UDP and the Model are at the same level of abstraction.

When a UDP is at a different level of abstraction than the Model, the mechanism to identify the

corresponding specifications of the input and output specifications is the same as described in section 7.4.

For some tasks in the Model, a specification is required by more than one task. This implies that all the tasks in between that specification and every output specification require that specification too. An example is given in Figure 23 where specification B2.1.1 has just been uploaded to the workspace of T1. Its corresponding parent specification B2.1 in the Model is required for standard task ST3 and ST4 where B2.2 and B2.3 are the output specifications. The corresponding specifications for B2.2 and B2.3 are B2.2.1 and B2.3.1 respectively. Therefore, links to the documents are put in the workspaces of T3, T4 and T6.

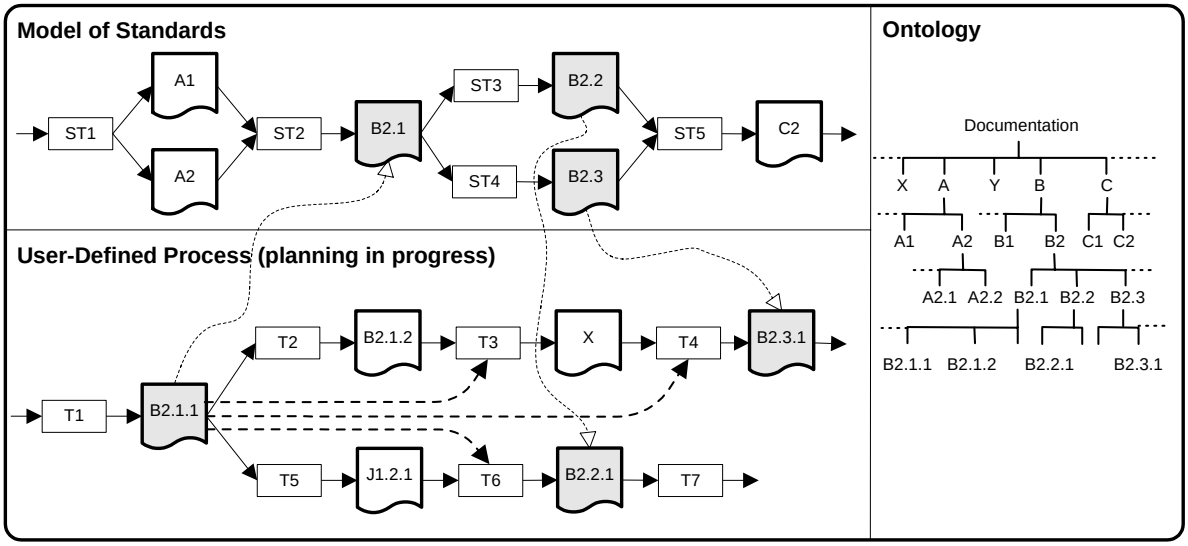

Figure 23. IN when a UDP is at a more detailed level of abstraction than the Model.

An algorithm for achieving this is shown in Figure 24.

```
Function NavigateDocument(doc, workspace, Std)
      doc: A document uploaded into a workspace of a
            user-defined task.
      workspace: The workspace where the document is uploaded.
      Std: The Model of a standard.
      task = GetTask(workspace);
      StdPreCond{…} = (x:x ∈ Standard Specification);
      StdPreCond = GetCorrespondingStdPreCond(doc,Std);
      StdPostCond{…} = (x:x ∈ Standard Specification);
      StdPostCond = GetStdPostCond(StdPreCond,Std);
      StdTask{…} = {x:x ∈ Standard Task};
      For each i ∈ StdPostCond
            StdTask = StdTask ∪ {GetAssociatedTask(i)};
      Next i
      TaskPreCond{…} = {x:x ∈ Task Pre-Condition};
      For each i in StdPostCond
            TaskPreCond = TaskPreCond ∪ _
            {GetCorrespondingTaskPreCond(i)};
      Next i
      UserTask{…} = {x:x ∈ User-defined Task};
      For each i in TaskPreCond
            UserTask = UserTask ∪ {GetAssociatedTask(i)};
      Next i
      PossibleTask{…} = (x:x ∈ User-defined Task);
      Workspace{…} = {x:x ∈ objects in a workspace};
      For each i in UserTask
            PossibleTask = GetTasksInBetween(task, i);
            For each j in PossbileTask
                  Workspace = GetInformationObjects(j);
                  If ({doc} ⊂ Workspace = 0 ) then
                        Workspace = {doc} ∪ Workspace;
                  End If
            Next j
      Next i
End Function
```

Figure 24. Algorithm of copying a document to the possible workspaces.

7.8 Cross-Referencing

Given a task or specification in a UDP, Cross-Referencing will locate the relevant part of the standard and the associated information. The inverse search is also allowed; it identifies all the user-defined tasks or specifications that relate to a particular part of the standard.

7.9 Error Prevention

Error Prevention is performed at run-time to prevent the execution of non-compliant tasks. It includes:

a) Correctness Check to ensure the required specifications are modelled properly.
b) Completeness Check to ensure the required specifications are included.
c) Capability Check to ensure the task agent possesses the required capability.
d) Recommendation Check to ensure the required techniques are considered.

When the workflow engine starts a task, Error Prevention will perform the checks. Execution can go

ahead only after passing all the checks or an explanation is provided, otherwise the task will be frozen until all the detected errors are resolved.

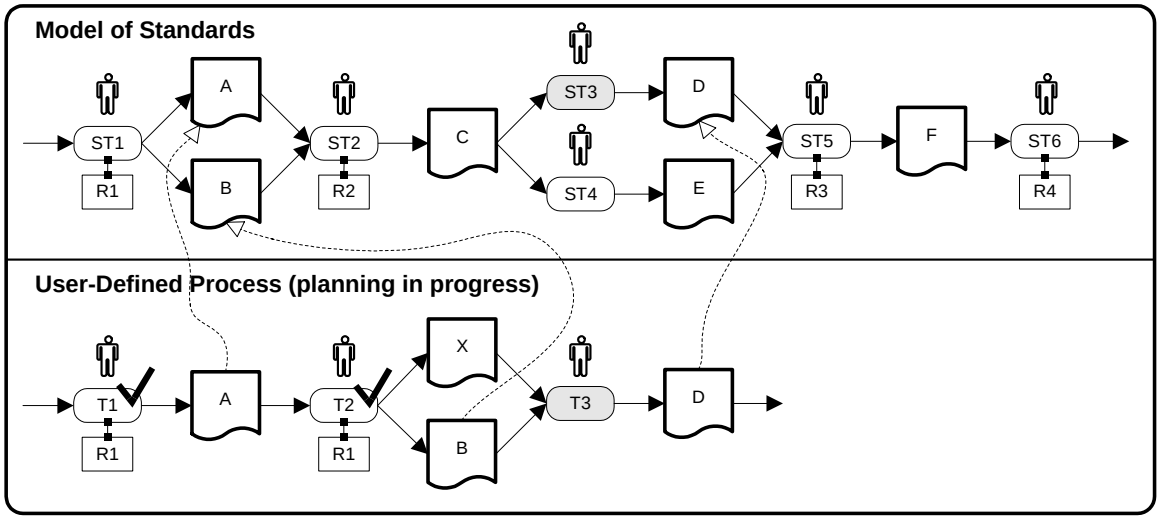

Figure 25. An example of Error Prevention.

In Figure 25, the user-defined tasks T1 and T2 are completed and the required specifications X and B are ready. When T3 starts, for which ST3 is the corresponding standard task, Error Prevention will be performed to ensure process compliance: Correctness Check ensures that specification D is placed in a correct sequence; Completeness Check ensures that specification C is available; Capability Check ensures that the task agent possesses the required capability; and Recommendation Check ensures that the required techniques or skills are used. Because specification C is not available, the Completeness Check fails and T3 is not allowed to proceed.

8. Case Study – Lightguard Development

The lightguard development project is one of three trial applications in the Assuring Programmable Electronic Systems (APES) project from ERA Technology Limited (ERA, 2000). The lightguard was originally developed to comply with BS EN61496 (EN, 2004)), Safety of Machinery - Electro-Sensitive Protective Equipment, and BS EN954 (EN, 1997), Safety of Machinery - Safety Related Parts of Control Systems. BS EN61496 is a product-family standard specifically with requirements for lightguards using active opto-electronic protective devices. BS EN954 is an application standard which provides guidance on the design and assessment of machinery control systems.

The lightguard development process is, however, incompliant with IEC61508. It has been analysed in the APES project and a number of correction issues towards IEC61508 compliance are given. A safety plan is proposed by ERA, however, neither the original process nor the corrected one are explicitly given in the publication. For the case study, a simulation using Compliance Flow to manage re-organised development tasks, based on available information, was performed. A number of errors in the

development process were identified, which demonstrate the effectiveness of the compliance management capability of Compliance Flow.

8.1 Lightguard Development Process

A lightguard performs a single generic safety function. Sets of infrared beams are used to scan a protected area and if a light beam is blocked, the machine it is guarding will be switched off. The main components of the lightguard and their interconnections are illustrated in Figure 26. A set of transmitter (Tx) and receiver (Rx) processors transmit and receive data to form a set of infrared light beams. Two independent controlling and monitoring channels (Control1 and Control2) are incorporated, which control the state of the two outputs signals that connect to the final switching devices (FSD1 and FSD2) in the machinery under control. Once any beams are blocked, Control1 and Control2 will be de-energised, which in turn de-energise FSD1 and FSD2. In order to limit common-mode failures of the equipment, the two channels make use of two different microcontrollers: an Atmel microcontroller and a PIC microcontroller. A cross-check is performed within the controlling and monitoring channels for fault detection purposes. The data from Control2 are shown on a local LED display that is controlled by a diagnostics processor. The programs for the transmitter, receiver, control and diagnostics processors are written in assembly language.

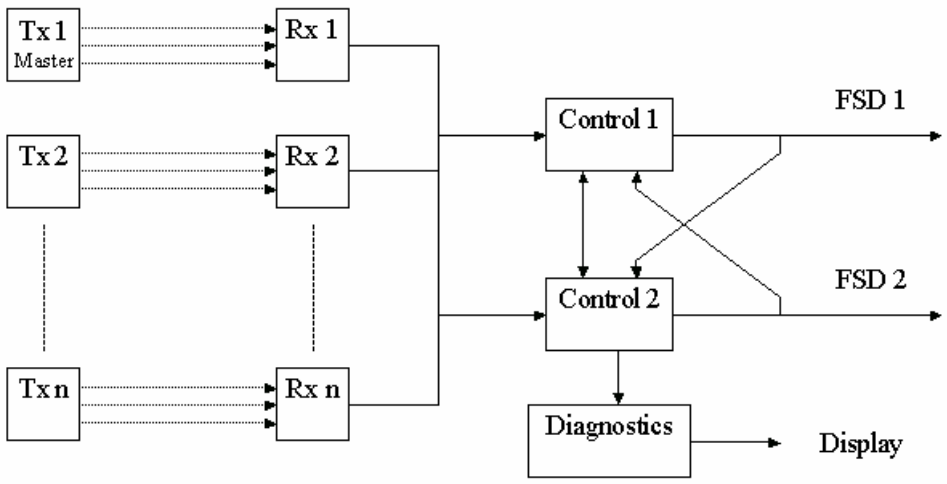

Figure 26. Lightguard components and interconnections (from ERA Technology, 2000)

The lightguard development process was designed to comply with BS EN61496 and BS EN954. Table 1 lists the high-level stages of the lightguard design process and the specification that is produced for each stage. The dependencies between the different stages are identified by their input and output requirements.

Stage	Specifications Produced	Dependencies (Required Input Specifications)
1	Requirements Specification for the Lightguard System	None
2	Hardware Specification for the Lightguard System	Stage 1 (Requirements Specification for the Lightguard System)
3	Functional Descriptions of individual modules	Stage 1 (Requirements Specification for the Lightguard System) Stage 2 (Hardware Specification for the Lightguard System)
4	Circuit Diagrams, Component Lists etc.	Stage 3 (Functional Descriptions of individual Modules)
5	PCB Layout Diagrams etc.	Stage 4 (Circuit Diagrams, Component Lists etc.)
6	Software Requirements Specification for the Lightguard	Stage 1 (Requirements Specification for the Lightguard System) Stage 2 (Hardware Specification for the Lightguard System)
7	Software Design Specification for the Lightguard	Stage 6 (Software Requirements Specification for the Lightguard)
8	Software Flowcharts	Stage 6 (Software Requirements Specification for the Lightguard) Stage 7 (Software Design Specification for the Lightguard)
9	Software Source Code Listings	Stage 8 (Software Flowcharts)

Table 1. Design stages and corresponding input and output specifications.

8.2 Compliance Management with IEC61508 Requirements

For the case study, the lightguard design process and its relevant information, such as the input (design requirements), the output (design specifications) and the techniques, involved in each stage were input to Compliance Flow. A screen shot of the model of the high level process structure using Compliance Flow is given in Figure 27.

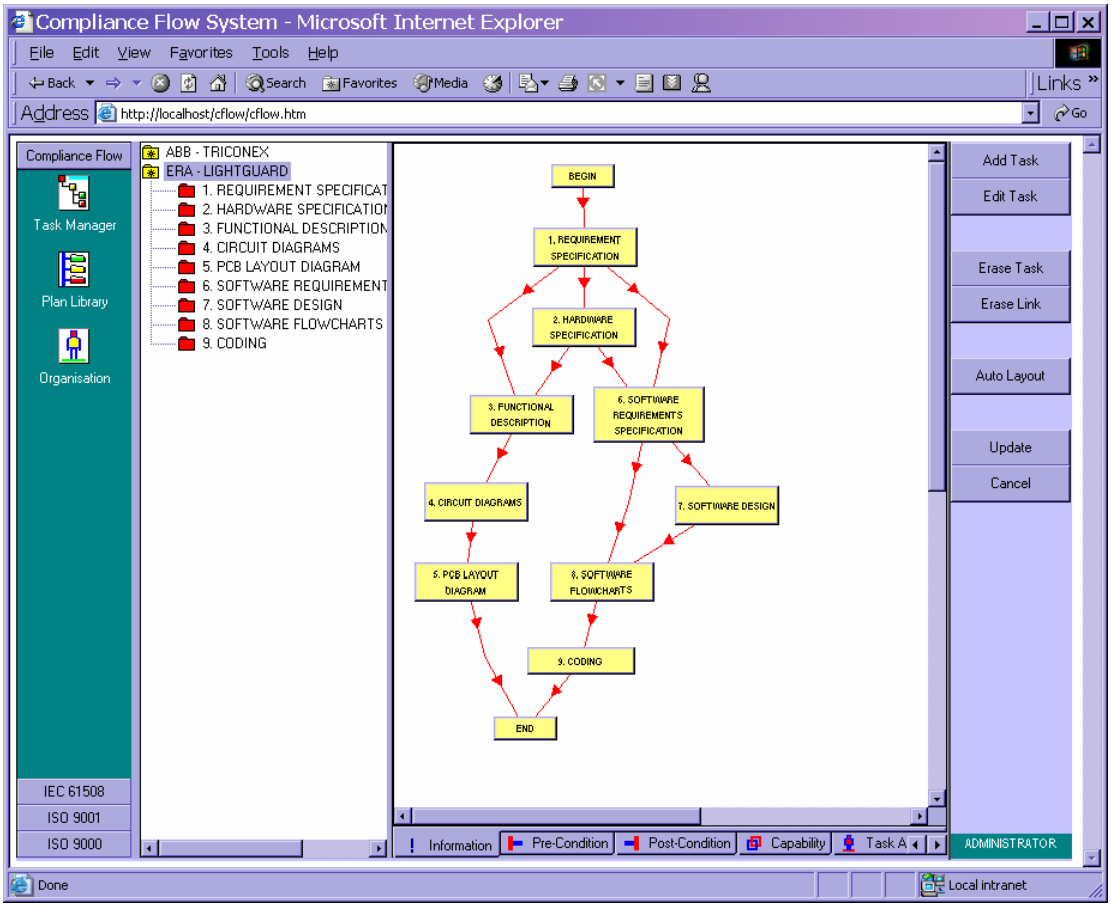

Figure 27. A screen shot of the high-level process model of a lightguard development process.

One of the purposes of the case study was to check whether the process complies with the IEC61508 application requirements outlined in section 3.3. The simulation demonstrates that Compliance Flow is able to provide intelligent assistance in detecting and managing many of the compliance errors. The findings of using Compliance Flow to deal with such errors are discussed below.

Requirement 1: The development shall be performed in accordance with a defined quality and safety plan.
Error: Identified some procedures are in place, which cover programmable electronics and software development activities carried out generally within the Design and Development stage. However, a document that defines the lightguard development process in full is missing.

Compliance Flow Solution: The use of a workflow system to mange the development process is a solution. This is due to the development process being modelled as a process plan in a workflow system, where all the details can be captured during task execution and relevant reports can be generated as required.

Requirement 2: Suitable techniques and measures shall be selected.
Error: Identified test specifications are not available, and therefore what testing techniques are employed is not clear.

Compliance Flow Solution: Completeness Check can ensure that test specifications are prepared before testing, and Recommendation Check ensures that the recommended techniques are considered for particular tasks. The recommended techniques for each test task can be listed by the system.

Requirement 3: The project organisation and allocated responsibilities shall be defined.

Error: Identified the lightguard project organisation and allocation of responsibilities as not documented explicitly.

Finding: The staff information, such as role and capability, can be maintained in the Organisation Server. As a task has to be assigned an agent before execution, the project organisation and allocation of responsibility must be defined and will be captured in the process model. In addition, the Capability Check can ensure that the assigned agents have the required authorities and skills to perform their tasks.

Requirement 4: Configuration management and change control procedures shall be defined.

Error: The Change Request Notes (CRN) and Change Notes (CN) are used. A spreadsheet is maintained of all firmware releases in which references are made to the corresponding CN. However, it is not clear that the impact of changes are systematically reviewed and documented.

Compliance Flow Solution: The procedures to deal with a change can be defined as a plan and maintained in Plan Library. Once an update is released, an appropriate plan is called and the required reviews will be performed and documented during the execution.

Requirement 5: Design reviews shall be planned and carried out.

Error: There is an ongoing process of review only during the lightguard development process rather than organised design review meetings. In addition, the formal minutes of these review meetings are not produced.

Compliance Flow Solution: The design tasks, including review meetings, can be modelled in the process plan. The post-condition is a set of minutes that will be produced after a meeting task is completed.

Requirement 6: Documentation shall be produced.

Error: Identified that there are no test specifications relating to testing of the system.

Compliance Flow Solution: Compliance Check can ensure the required test specifications are prepared before the testing.

Requirement 7: Hazard analysis and risk assessment shall be carried out.

Error: Potential failure modes of the lightguard equipment are identified and recorded in a database. However, the objectives and procedures for these activities are not clearly defined.

Compliance Flow solution: The required objectives will be dealt with by Completeness Check while Correctness Check will ensure that the related activities are defined in a proper sequence.

Requirement 8: A safety requirements specification shall be documented.

Error: No error can be identified even though the software design specification does not provide a complete and unambiguous specification of the safety requirements for the lightguard development. For

example, in the software design specification, it is simply stated that “the serial numbers shall be contained in the transmitter and by each independent microprocessor in the receiver”.

Compliance Flow Solution: Compliance Flow can provide limited assistance in managing errors that concern with the document content. It is only able to ensure that the required information (sections) is involved in a document by defining the structures of such information in the hierarchical documentation ontology, and which will be defined as the post-conditions of relevant tasks. A document with such contents will be produced after the tasks are completed. However, the document context is unable to be assessed.

Requirement 9: There shall be clear traceability from requirements through design.

Error: Because Compliance Flow is unable to assess the document context, no error can be detected even though the structure of the requirements and design documentation for the lightguard does not exhibit explicit traceability between the safety requirements and the associated implementation.

Compliance Flow Solution: As the development process is captured by the process model and the details of the task executions are recorded in Tracking Server, limited support can be provided by replaying the implementation process and the decision paths. However, tracking the process is far away from what the standard requires.

Requirement 11: Appropriate design techniques shall be employed depending on the required safety integrity level.

Error: Identified that only timing diagrams and software flow charts are used in the software design. Some highly recommended techniques are not used and required explanations are not provided.

Compliance Flow Solution: The recommended techniques for different SILs are provided, ordered by their importance, by Recommendation Check. Suitable techniques are selected and defined as pre-conditions of particular tasks. Explanations of not using the recommended techniques have to be provided by users.

Requirement 12: Test specifications shall be prepared prior to testing.

Error: Identified a test plan and test specifications are not documented. Functional testing of the integrated software and target hardware are informal.

Compliance Flow Solution: Completeness Check and Correctness Check can ensure the test plan and test specifications are developed in advance of testing. Recommendation Check can ensure every test task is performed with appropriate techniques while Capability Check ensures the task is performed by qualified people.

Requirement 13: The results of test and analysis activities shall be recorded.

Error: Identified software testing is performed informally and only recorded in project notebooks.

Compliance Flow Solution: The test procedures and activities are specified in a process plan with their execution under the control of the workflow engine, and the required test results and analysis activities are recorded.

Requirement 14: The development shall be subjected to independent safety validation and assessment.
Error: It is unclear how the requirement for independent validation of the lightguard design is to be handled.
Compliance Flow Solution: As procedures and activities can be specified in a process plan, the validation and assessment activities should be included. The independence of an assessor can be defined as a capability so that suitable assessors can be identified in the agent selection process supported by capability matching. The concept of workspace enables the assessors to retrieve the required information effectively.

The case study highlighted the kind of errors that Compliance Flow can identify and how they can be handled. As Compliance Flow deals with processes and does not deal with the contents of documents, there are some aspects of standards that it cannot handle. Integrating Compliance Flow with a content-based compliance checker will provide a very powerful tool.

9. Conclusions

In this paper we have identified standard compliance as an important issue in process management. The proposed solution is to model the standard in a way that the required information can be used for automatic compliance checking. Compliance checks are used to check a UDP against the standard model, during both process build-time and run-time, to identify compliance errors, assist in process specification and prevent non-compliant tasks in a process from being performed accidentally.

The proposed approach of standard modelling is capable of capturing the main elements of a wide range of standards for compliance checks. Correctness Check, Completeness Check, Capability Check and Recommendation Check can identify errors at process build-time while Error Prevention is able to prohibit the execution of non-compliant tasks at run-time. The Cross Referencing function allows the user to locate the relevant part of a standard by giving a UDT or specification, or vice versa. Planning Assistance and Information Navigator provide planning support and information management at build-time and run-time respectively.

A study is performed. The original lightguard development project does not comply with IEC61508 standard as it does not fulfil all the requirements. The case study shows that the non-compliance errors are successfully detected and managed by the system. Insights gained from the case study are discussed.

The principle contribution of this paper is the proposed solution of compliance management. From the compliance management perspective, Compliance Flow provides process-level compliance management, pro-active error identification and document production control. These three features are critical in

compliance management, but are not supported by current systems. From the workflow management perspective, Compliance Flow is able to support all the flexible features provided in the four adaptive WfMS discussed before. The ability to check process compliance against a standard is an innovation in workflow management.

References

Campbell A. E. and Shapiron S. C., 1995. *Ontological Mediation: An Overview*, Proceedings of the IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing, Menlo Park CA: AAAI Press.

Cheung Y.C., Chung P.W.H. and Dawson R.J., 2002, *Supporting Engineering Design Process with Compliance Flow – An Intelligent Workflow Management System*, Engineering Design Conference 2002 King's College London, 2002.

Easterbrook S. M., W. Finkelstein A. C., Kramer J., and Nuseibeh B. A., 1994 Coordinating Distributed ViewPoints: The anatomy of a consistency check. *Journal of Concurrent Engineering: Research and Applications* , Vol 2, No 3 (Special Issue on Conflict Management)

Emmerich, W. and Finkelstein, A. and Montangero, C. and Antonelli, S. and Armitage, S. and Stevens, R., 1999, Managing *standards compliance*. IEEE Transactions on Software Engineering, 25 (6). pp. 836-851. ISSN 00985589, 1999

EN, 1997, *EN954-1:1997 - Safety of machinery. Safety related parts of control systems. General principles for design*, 15 June, 1997

EN, 2004, *EN61496-1:2004 - Safety of machinery. Electro-sensitive protective equipment. General requirements and tests*, 1 June, 2004

ERA Technology, 2000, *Assuring Programmable Electronic Systems (APES) Project.*

Finkelstein A., Gabbay D., Hunter A., Kramer J., and Nuseibeh B., 1994. *Inconsistency handling in multi-perspective specifications*. In IEEE Transactions on Software Engineering, volume 20, pages 569--578, August 1994.

IEC, 1997, *Draft Standard IEC61508 Functional safety of electrical/ electronic/ programmable electronic (E/E/PES) safety-related systems, Parts 1 to 7*, December 1997.

ISO, 2000, *ISO 9001 Quality Management Systems - Requirements*
International Organization for Standardization, 01 December, 2000

Marshak, R.T., 1994, *Workflow white paper: An overview of workflow software*, in Proceedings of WORKFLOW'94, San Jose, August, 1994.

Marshak R.T., 1997, *InConcert Workflow: Independent from XSOFT, InConcert Inc. Provides Flexible Workflow Underlying Engineering Team Support*, Workgroup Computing Report, Patricia Seybold Group, March 1997.

McCarthy D.R., Sarin, S.K., 1993, *Workflow and Transaction in InConcert*, Bulletin of the Technical Committee on Data Engineering, Vol. 16 N2 - IEE, June. Special Issue on Workflow Extended Transaction Systems.

Moore J., Stader J., Chung P., Jarvis P., and Macintosh A., 1999. *Ontologies to Support the Management of New Product Development*, in proceedings of the International Conference on Engineering Design, ICED 99 Munich, August , 1999.

Moore L., 2002, *Epigram Profit from Safe Systems*, Spring 2002, Edited by Nunns, S.

Uschold M., Gruninger M., 1996. *Ontologies: Principles, Methods and Applications*, The Knowledge Engineering Review, Vol. 11, No. 2, 1996, pp. 93-136.

WfMC, 2002, *Workflow Process Definition Interface—XML Process Definition Language (XPDL) WFMC-TC-1025 Final*, Workflow Management Coalition, 25, Oct, 2002.