Abstract. Development of newer and more sustainable systems requires a thorough understanding of the complex interactions in current systems. Therefore it is necessary to be able to switch between detailed knowledge of component systems and an overall appraisal of the entire system. Current efforts to develop ontologies capturing a "complete" and "universal" understanding of entire systems of systems often result in loss of depth and precision of knowledge contained in the participating systems. This further adds to the uncertainty and intractability in the management of the complex system. In addition, the absence of a single control and execution context makes it difficult to validate the system against desired intent and goals. All of these increase the likelihood of cost, effort and development time overruns in maintaining, enhancing, retiring and replacing systems.

In this paper, we propose a novel approach to address these concerns by the application of Omnispective Analysis and Reasoning (OAR), an epistemic framework for managing intellectual concerns. By creating "localized ontologies" for capturing the 'silos' of knowledge in component systems, we develop artifacts for only those concerns from the participating domains that are identified as relevant. These localized ontologies can unambiguously capture all relevant system artifacts with valuable information about their context of application within the system. With the OAR framework, we can analyze and manage large systems as an aggregation of all these localized ontologies with explicit specification of mutual interactions and influence at the concept, model and implementation levels. This omnispective outlook will not only enable better management of the system development lifecycle by taking into account details of individual subsystems and their interactions, but also facilitate validation and verification of the system. We illustrate this with an example from the Ubuntu software ecosystem.

Keywords: Complex Systems, Omnispective Analysis and Reasoning, Context, Localized Ontologies

INTRODUCTION

The study of large and complex systems – both engineered as well as naturally evolving – is gaining importance. This is mainly due to the wide-ranging impact of such systems on human society. Examples of such systems include large power networks, computer system networks, ecological systems etc. Though terms like large systems, complex systems, systems of systems and wicked problems are employed with varying and several connotations in the literature, complex systems are seen to possess a set of commonly recognized properties. Firstly, they consist of many component systems, interacting with each other and with the environment(s) within which they operate or exist. Secondly, they are open in nature in the sense that component systems may be added, modified, retired or replaced. Finally, the goals and objectives of such systems are very likely to change and evolve with time. Reviews of the salient features and concepts of complex systems in the context of systems of systems can be found in a number of references (Jamshidi, 2008; Sage and Cuppan, 2001; Sheard and Mostashari, 2009).

For the development of newer and more sustainable systems, it is usually necessary to analyze and understand the complex interactions in current systems. For this it is appropriate:
1. to be able to switch between detailed knowledge of the component systems and overall appraisal of the entire system. Current efforts to develop ontologies capturing "complete" and "universal" understanding of the entire system increase overall complexity, while simply providing minimally sufficient understanding across all interacting systems which may obscure details of component systems.

2. to develop the ability to treat each component subsystem and its constituents as a localized and single "control and execution context" so as to enable system validation against desired goals and intent.

Inadequate support for the above features, coupled with uncertainties due to incompletely recognized or unrecognized interactions of the component systems, may further result in uncertainties in the outcomes of the processes orchestrated by the system, together with the increased likelihood of cost, effort and development time overruns in maintaining, enhancing, retiring and replacing systems.

In this paper we propose a novel approach to address these issues by applying Omnispective Analysis and Reasoning (OAR) (Chemboli and Boughton, 2012), an epistemic framework for managing intellectual concerns. This omnispective outlook enables better management of the system development lifecycle, and facilitates validation and verification of the system by taking into account details of individual subsystems and their interactions.

The remainder of this paper is organized as follows. In the following section, we discuss some of the issues encountered in the design and management of large and complex systems. Next, we present an overview of the OAR framework, highlighting its main features and approach to concern and context refinement. We then discuss the application of OAR to complex systems, followed by an illustration of the application of the OAR framework to analyze and work through the problem of selecting the default music app for the Ubuntu 12.04 Long Term Support (LTS) distribution.

**DESIGN OF LARGE SYSTEMS AND SOME ISSUES OF COMPLEXITY**

**Some Characteristics of Large Systems**

As far as we have been able to determine, all the studies and attempts aimed at managing complexity are based on the implicit belief that proper design of a system can mitigate the issues/difficulties resulting from the complex interactions of the component systems comprising it.

According to Warfield (1994), a system is considered to be a large-scale and complex system only in relation to “human capacity to observe it, comprehend it, analyze it, steer it, amend it and tolerate it.” This concept of a complex system is based on consideration of both situational complexity and cognitive complexity, and any improvements to the management of complex systems should address both these kinds of complexity.

Systems that have technological underpinnings and have considerable sociological impact and interactions, are termed as socio-technical systems. Some of these, like forestscapes and water-body systems come into existence through a process of evolution. Some others, like human settlements, are designed and built (at least initially) and then continue to evolve (Alexander, 1964). Socio-technical systems evolve through synergistic interactions between technology and people.

Technological systems that are designed and built can be separated into two main groups (Warfield, 1994):

1. Systems that are designed entirely on the basis of well-established principles of science and engineering. *These systems can be validated against the standards of knowledge of science. Their failures are generally due to components and process parameter errors/inaccuracies and can be well-characterized. The impact of such failures can be managed well in most cases.* Examples of such systems are radio and television broadcast systems, chemical plants etc.

2. Systems that are designed and built on the basis of perceptions and models of the needs of the user and properties of the system, like computer software, programming languages etc. *Some of these systems may additionally include a number of entities of the first kind. For such systems, reference against primary standards of science is less direct because their reference is through the models used in the...*
design. Examples of such composite systems are information systems, management support systems, expert computer systems, hospital and health care systems, nuclear plants and banking organizations etc. The behavior of such systems is similar to socio-technical systems because their satisfactory performance depends on the synergistic interaction of their component systems.

Basic scientific exploration and research of complex systems consists of recognizing a system as complex, recognizing possible manifestations of such complexity, and formulating the concepts and models that may be used in their description. Theories of catastrophe (Arnold, 1986), chaos and entropy (Mitchell, 2009) provide directions to the efforts at gaining knowledge in this domain.

How Complexity Builds and Escalates in Large Systems

A typical scenario of managing complexity consists of careful design of the processes and working environment and assigning specialized roles to properly trained personnel. The situational complexity of a system arises due to interactions amongst the component systems as well as interactions with the environment. Added to this will be features of cognitive complexity – as felt and perceived by the personnel managing the system. As depicted in figure 1, additional resources inducted into the system for the purpose of managing complexity, themselves, will escalate complexity by creating new linkages (Warfield, 1994). Thus, managing a complex system should focus on control of situational complexity, at the same time keeping the scope of management within the cognitive abilities of the people managing the system.

![Figure 1: Escalation of problem complexity when additional inputs are provided.](image)

Effective management requires analyzing and understanding the design of the system in terms of the concepts, models and implementations involved. Recognizing the concerns that are vaguely understood but relevant to the outcomes of the system will enable the planning of strategies for managing failures due to these concerns. The OAR framework has features that are well-suited to address these problems in large systems, as will be described in the next section. The framework supports the maxim, what is understood with referential validity is science, and what is modeled and designed in terms of the science is technology.

**THE OAR FRAMEWORK**

The Omnispective Analysis and Reasoning (OAR) framework (Chemboli and Boughton, 2012) has been developed to address the problem of capturing and reusing the intellectual effort in scientific workflows. It utilizes an extended definition of a scientific workflow which encompasses any generic problem scenario that conforms to the fundamental nature of a scientific process and the method of science – that the investigation be logical and repeatable and the entities and actions involved be amenable to evidentiary validation. The process of analysis, design and operation of a system is considered as a workflow formulation, solution specification and implementation. An omnispective appraisal of all the identified concerns in a system in terms of the component systems, the concerns within the component systems and the interactions between them will be subjected to the OAR process for relating them to the modeling of the system and the formulating of a solution specification.
The basic concepts of the Domain of Science Model (DoSM) and the Science of Generic Design (Warfield, 1994) are kept in view in constructing the OAR framework. Since the scope of scientific workflows may range from “fairly simple” to “highly complex” scenarios, the design has to be such that it can be easily understood, comprehended and utilized. For this purpose the Law of Triadic Compatibility (Warfield, 1994) is used as a guide. This law has been demonstrated to be effective in producing design of complex systems in such a way as to avoid unexpected difficulties and failures – by bringing all the identified concerns surrounding the problem clearly into the compass of comprehension of the human practitioners designing and using such systems. Earlier studies of analyzing a language of folds in origami (Chemboli and Boughton, 2012), and contextualizing the design of a software engineering course using a Learning Management System (LMS) (Chemboli and Boughton, 2011; Chemboli, 2010) demonstrate the applicability of OAR to diverse problem situations. The key concepts and processes of the OAR framework are revisited in the following sections.

Omnispective Analysis

Omnispection provides an overall appraisal of a system; Analysis, a close examination of all the recognized entities contained therein; and Reasoning, a discovery of the relationships and interactions between the entities and also with their environment.

Application of Omnispection, Analysis and Reasoning, results in assembling all the relevant information of the inquiry domain abstracted into convenient units (concerns) encapsulating clearly all the relevant interrelationships forming the overall repertory from which one can define the scope of the problem and formulate a solution.

In carrying out omnispective analysis, a problem needs to be examined from all angles of all the interacting disciplines that are considered relevant. Iteration steps in the style of Observe-Orient-Decide-Act (OODA) loop (Boyd, 1986) are employed during the process to achieve well-formed concern groups to represent the problem in its entirety. During this process, it will be possible to identify areas considered relevant but not adequately enunciated or recorded which can then be marked as areas needing further study and experimentation. Some more benefits of the analysis and abstraction processes are the identification of concerns that are critical to the problem, revealing the precision of data and recognizing the possible interactions between different sub-domains to avoid unexpected results. This is very much similar to defining risk context (Standards Australia, 2009), indicating that OAR can be an effective risk management process.

A recipe consists of one or more concerns (that are related and interacting) which will be directly useful in the formulation of the system. Recipes are formed to facilitate a clear and repeatable representation of a problem scenario, and are considered at three levels of abstraction – concept, model and implementation levels as detailed in figure 2.

![Figure 2: Overview of the OAR framework.](image-url)
between them. These can be ideas, scientific and engineering principles and perception and vision of goals captured in napkin diagrams, descriptive documents, minutes of brainstorming meetings, etc. At the model level, they represent physical and logical systems in terms of formalisms incorporating theories and paradigms. These may be abstracted in mathematical and analytical models, vocabularies, data sets, natural language representations, ontologies and process guidelines. Implementation level entities capture system frameworks, mechanism and translation processes to satisfy specifications in terms of and conforming to the recipes at the concept and model levels. Every recipe within the repertory can be evaluated at any of the three levels depending upon the context of the problem analysis.

Prototypes, Archetypes and Constraints

OAR recipes are categorized into three types – prototypes, archetypes and constraints – depending on the degree of firmness and influence (firmness and influence are described later).

A prototype is any OAR recipe that is available in a given domain without particular consideration of its applicability, degree of formalism or robustness for any fit or purpose. Thus, a prototype can encapsulate either nascent or well-formed domain concerns that may be available to support the analysis of any problem situation using the OAR framework. Depending on the area of study, the prototypes can range from rudimentary outlines and sketch-ups to formal blueprints.

An archetype is a prototype that can be considered an exemplar or best practice recipe for a concern in a particular domain. The choice of an archetype is dependent upon the analysis of the problem under study and influences our nett understanding of the problem domain.

A constraint is a special instance of an archetype which is identified as imposing strict criteria on an OAR specification. A valid solution of the problem is required to sufficiently satisfy all the requirements of the constraint without exception, and is often subject to strict conformance.

Concern Refinement

Individual concerns (recipes) in OAR are managed as collections known as a shelf. A shelf is simply an unordered collection of recipes categorized by individual subject domains and their relevance to the problem under consideration. Each OAR shelf can accommodate zero or more prototypes which need not necessarily be atomic recipes. Overlapping prototype groups can exist depending on the granularity of the domains under study. Shelf management is illustrated in figure 3.

![Figure 3: Shelf management in the OAR framework.](image)

Various External Shelves hold all the known recipes (prototypes) from different domains of interest in the analysis of the problem situation. Each external shelf is populated with concepts, data, constraints, models, details of data collection procedures, and experimental processes – in a 'reasonably usable form'. These may range from loosely sketched napkin diagrams to precise specifications, descriptions of sys-
tems, mathematical theories, formulations etc. Any number of external shelves can be populated to accommodate all the recipes.

The **Problem domain shelf** holds recipes selected from various external shelves that satisfy given criteria in the problem under consideration. These exemplars or best practice recipes are effectively Archetypes which represent our nett understanding of the concerns in the problem domain.

The **Solution Shelf** consists of recipes which are specifications of the archetypes in the problem domain subject to Constraints we identify and impose on the particular instance of a Solution. These meta-recipes (recipes of recipes) are interconnected specifications of the archetypes. Depending on the context, a solution shelf may either be an executable domain or may require further translation.

Concern refinement is carried out as follows:

1. Initialize and populate external domain shelves with prototypes. This bootstrap step is not required if domain knowledge is available in identified preexisting external shelves.
2. Identify prototypes and possible archetypes and constraints from various external shelves which have a bearing upon the problem under consideration. Collect these recipes in the problem domain shelf. All recipes identified in this step need not be utilized in the solution shelf.
3. Analyze the archetypes in the problem domain using context refinement to specify a solution specification in the solution shelf. This step also identifies the problem domain archetypes which impose limiting criteria on the specification and are termed as constraints.

The OAR framework facilitates localized ontologies via shelf management. The external shelf need not represent a formally complete understanding and representation of all domains that are cataloged in the framework. The problem domain shelf enables building a localized ontology that is good enough for the particular problem situation even if it may be inadequate for universal use. The solution shelf captures all identified recipes and constraints. If further analysis of the problem situation reveals additional interacting concerns, the localized ontologies can be further extended and refined.

**Managing Context**

The context of an entity is considered in terms of its relationship with, and its influence on, other entities in the problem domain. Context is specified in terms of a set of attributes or dimensions in order to bring out the soundness and relevance of the intellectual concerns embodied in the recipes and to establish their role in the formulation and implementation of the system application. In the OAR framework, context is a function of two dimensions – Firmness and Influence (figure 4).

**Figure 4: Recipe context as a function of Firmness and Influence in the OAR framework.**

*Firmness* is a measure of the degree of well-formedness of a recipe. If a recipe is ambiguously defined, vague, or peppered with implicit characteristics, the knowledge encapsulated therein can be considered to be pliable and to have low firmness. On the other hand, a well-formed and explicitly defined recipe has high firmness.
Influence is a measure of the effect exerted by a recipe in the analysis of the problem domain. If an entity is identified as relevant to the problem domain, then it exerts a non-trivial influence. If it exerts a strong influence, then it is identified to encapsulate exemplar criteria or best practice for the problem situation. If it does not affect the problem domain, but may still be considered relevant, it is identified as exerting weak influence.

The contextual relationship between two entities is assigned through the process of reasoning. Firmness is an intrinsic property of a recipe whereas influence is a relational property with respect to another recipe. An influence relationship may be directional and asymmetric.

**Context Refinement**

The relevance of recipes to a problem domain is determined using the step-wise process of context refinement as depicted in figure 5.

1. **Identify recipes which exert influence.**
2. **Determine connections between the recipes.**
3. **Evaluate Influence and Firmness of the connection.**
4. **Develop the context mapping for the solution specification.**

![Figure 5: The process of context refinement.](image)

To start with, no *a priori* assumption is made regarding the influence and firmness of the recipes. If analysis suggests the use of a particular recipe, then it is identified as exerting a non-zero influence on the problem under study. In addition, if the recipe falls under the category of an *exemplar* or *best practice* in the discipline, then it is identified as firm. Consequently, the solution specification composed from the selected recipes will become increasingly firm as situational and imposed constraints are satisfied.

Though OAR recipe context is a continuous function of two dimensions, as shown in figure 4, in most circumstances, recipe context can be conveniently tagged by discrete labels of the function $C(I, F)$ affecting prototype selection.

The first two steps of context refinement in figure 5 may be carried out recursively to obtain a complete mapping of the context relationship in the final specification.

**APPLICATION OF OAR TO COMPLEX SYSTEMS**

As explained in the previous section, the process of analysis, design and operation of a system is considered as a workflow formulation, solution specification and implementation. In relation to complex systems, a particular outcome or application of the system is considered to be the problem scenario and the OAR process is applied to it.

Complex systems integrate many interacting (diverse) disciplines. As a result of this diversity, forcing a common ontology for entire systems of systems may result in loss of valuable domain insights. Conversely, a highly focused exploration of individual system domains may also lead to a loss of generality.
in the nature of systems understanding. Thus, a common ground between too much detail of individual domains and too limited a view of the entire system, is highly desirable. Current efforts (Staab and Studer, 2009) in this direction are focused on developing and building comprehensive language and concept interchange formats for “universal use” – ontologies for universal acceptance. Large ontologies have the side-effect of further adding to the nett system complexity owing to the difficulty of arriving at a common understanding that may be universally applicable across many/all the disparate component systems. Most strategies employed (Iordan and Cicortas, 2008; Sahoo et al., 2008; Truong et al., 2005) aim to create ontologies which furnish minimally sufficient understanding across the entire system, thus losing significant detail of intellectual concerns in the component systems.

A way forward is to make use of localized ontologies to capture and represent the individual 'silos' of knowledge in domains which constitute the large system and develop the interchange processes for only that subset of the intellectual concerns which are relevant for the application of the complex system. These localized ontologies need not be complete in the sense of universal applicability. But they will unambiguously capture all the system artifacts along with information about the context of their application.

As a result, large systems of systems may now be viewed as an aggregation of numerous localized ontologies, each of which can be further adapted, evolved, retired and replaced as appropriate to changes in the circumstances in which the systems operate and interact.

Localized ontologies are realized by shelf management in the OAR framework. Intellectual concerns across component subsystems are captured in recipes by the process of concern refinement. The collection of external shelves represent all available interdisciplinary knowledge which can be used to manage the complex system and its component systems. Interactions between component systems may themselves be the subject matter of further external shelves and the problem domain shelf.

Large and complex systems are typically characterized by the absence of a single control and execution context. This often results in difficulty in managing the component systems and the interactions between them. Using the OAR framework, context relations within and between individual component systems can be characterized by:

1. Strong influence and high firmness \((C(1, F = 1))\), i.e., the context relations between the recipes within each system and between systems are significant and well-understood. This means that the interaction between the systems will not result in any unexpected behavior.
2. Strong influence and low firmness \((C(1, F = 0))\), i.e., the concerns affecting the behavior of the component systems are recognized as significant, however the recipes are not well-formed. As a result, unintended system behavior cannot be ruled out.
3. Weak influence and low firmness \((C(0, F = 0))\), i.e., the system interactions are neither properly recognized nor understood – they are only suspected to be present, and may produce unpredictable and unexpected effects.
4. Weak influence and high firmness \((C(0, F = 1))\), i.e., although the recipes are well-formed, their interaction is either not known or is considered insignificant.

The last two context relations indicate the need for further research in identifying the particular influences.

Context refinement thus enables us to consider the system as an aggregate of component systems each with its goals and intent, and then working out a linkage between them. This facilitates local control and execution contexts for both the component systems and the interactions between them. The solution specification, an artifact arrived at by utilizing localized ontologies via shelf management, and concern and context refinement processes, corresponds to the outcome of the application of the system.

The nature of omnispective analysis will also:

1. Enable better management of system development lifecycle by taking in account component systems and their interactions;
2. Facilitate verification and validation of the underlying intellectual concerns in the component systems, their interactions and thus of the overall system;
3. Reduce the cognitive burden on the human interacting with the system by organizing the intellectual concerns as recipes in accordance with concept level, model level and implementation level.

**ILLUSTRATION: APPLYING THE OAR FRAMEWORK TO THE UBUNTU PLATFORM AS A COMPLEX SYSTEM-OF-SYSTEMS**

According to the criteria outlined by Abbott (2006), the Ubuntu platform ecosystem (Canonical Ltd, 2011) qualifies as a complex system. It is open and consists of numerous interacting and continually evolving component systems; which are added, modified, replaced or retired over time.

This illustration of the application of the OAR framework centers on analyzing and working through the problem of selecting the default music app for the Ubuntu 12.04 Long Term Support (LTS) distribution. It will be shown that the problem involves the interplay of a number of engineered, societal, legal and various other systems, each with its own cadence and pace of change and development. It will also be shown that the influence of changes in component systems will be felt locally as well as in other systems within the wider platform ecosystem.

The selection of the default music app for an operating system distribution may not result in “disaster” in the event of failure. Nevertheless, it amply illustrates the manner in which all the component systems and their interactions need to be examined in order to arrive at a solution specification using the OAR framework. Thus this illustration provides insight to the management of localized ontologies developed through the OAR process, and also demonstrates how the difficulty of not having a single control and execution context can be mitigated because the OAR solution specification provides for validation against desired intent and goals.

**Capturing Intellectual Concerns for the Ubuntu Ecosystem**

Identified knowledge in different domains within the Ubuntu ecosystem is captured in different recipes which are managed through external shelves in the OAR framework. Depending upon the granularity of the problem objective, the choice of the external shelves varies.

Recipe details incorporate outcomes of Internet Relay Chat (IRC) logs, document blueprints, Ubuntu Developer Summit proceedings, discussion forums and project guidelines. Specific details of the recipes are omitted for brevity.

**Initializing External Shelves**

Again for brevity, the external shelves with a non-exhaustive list of recipes shown in figure 6 are assumed or deemed relevant to the Ubuntu platform ecosystem.

![Ubuntu Ecosystem
Societal
Organizational
Technical
Societal Domain
Ethics
Free Software
Open Source
Community Responsibility
Ubuntu Code of Conduct
Technical Domain
Ubuntu Platform
Applications
Languages
Organizational Domain
Contributors
Users
LoCo Councils
Marketing
Finance
Administration
UDS Session
Ubuntu Platform
Toolkits
Desktop Experience
Core
Kernel
Core
Graphics
Security
Comms
Network
Multimedia
Accessibility](image)

*Figure 6: External shelves and recipes for deciding the default music app.*

The idea of localized ontologies can be seen in action here in selecting shelves that may be adequate for steering a solution specification.
Identifying Relevant Archetypes and Constraints

We now carry out concern refinement in order to select the specific prototypes which can be imported into the problem domain shelf. As a result, the prototypes for LTS, ISO Size, Licensing and Release Date are recognized as constraints (as shown in figure 7), and the problem domain shelf is populated with further prototypes which may now be considered as archetypes owing to their relevance to the problem domain, as shown in the figure.

![Diagram showing prototypes and constraints]

Figure 7: Archetype and constraint identification in the problem domain shelf.

Solution Specification for Selecting the Default Music App

A solution specification is arrived at by the OAR process of context refinement (figure 5). Accordingly, to define a solution specification for selecting the default music app in Ubuntu 12.04 LTS, we perform context refinement. This process is carried out as follows:

SS-1 A Long-Term Support (LTS) release of the Ubuntu Operating System is of five years duration. In addition, each LTS is guaranteed to provide a dependable upgrade path to the next LTS release. Hence, we see that Ubuntu 12.04 is subject to the constraint of rigid conformance to LTS guidelines, and is required to support the platform technologies for the entire duration.

\[
\text{LTS} \quad C \left( I = 1, F = 1 \right) \quad 5 \text{ Years}
\]

\[
\text{Ubuntu 12.04} \quad C \left( I = 1, F = 1 \right) \quad \text{LTS}
\]

SS-2 The Ubuntu platform distribution is built as a live and installable CD which has a constraint of maximum size (< 700 mb).

\[
\text{ISO Size} \quad C \left( I = 1, F = 1 \right) \quad <700\text{mb}
\]

SS-3 The release date is decided as per policy and is for most intents and purposes, non-negotiable. Therefore, this is a strict constraint.

\[
\text{Release Date} \quad C \left( I = 1, F = 1 \right) \quad 2012-04-26
\]

SS-4 The MonoGtk3 port is still under development and will not be ready before the Release Date. The Banshee music app is developed on the MonoGtk stack. Porting the app to Gtk3 cannot progress until the MonoGtk3 toolset is stable. On the other hand, the development of Rhythmbox is not dependent on the status of the MonoGtk3 port, since it is coded in the C programming language with libgobject bindings. Accordingly, the context states can be specified as:

\[
\begin{align*}
\text{MonoGtk 3} & \quad C \left( I = 0, F = 0 \right) \quad \text{Release Date} \\
\text{MonoGtk 3} & \quad C \left( I = 1, F = 1 \right) \quad \text{Banshee} \\
\text{MonoGtk 3} & \quad C \left( I = 0, F = 0 \right) \quad \text{Rhythmbox}
\end{align*}
\]

SS-5 One of the goals of the LTS distribution is substantial migration of the platform to the GTK+3 toolkit. This decision introduces a preference to include a GTK+3 app where possible if it provides reasonable feature parity with other alternatives. The Rhythmbox media player has been completely ported to GTK+3. As seen in SS-4, MonoGtk3 still requires...
further development.

<table>
<thead>
<tr>
<th>Application</th>
<th>Licensing</th>
<th>ISO Size</th>
<th>Release Date</th>
<th>GTK3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banshee</td>
<td>C (I = 1, F = 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rhythmbox</td>
<td>C (I = 1, F = 0)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SS-6** The Ubuntu One cloud service is fully implemented as a plugin for Banshee. A Rhythmbox plugin is currently in development, and cannot yet be considered firm.

<table>
<thead>
<tr>
<th>Service</th>
<th>Licensing</th>
<th>ISO Size</th>
<th>Release Date</th>
<th>GTK3</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1 Service</td>
<td>C (I = 1, F = 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U1 Service</td>
<td>C (I = 1, F = 0)</td>
<td></td>
<td></td>
<td>Rhythmbox</td>
</tr>
</tbody>
</table>

**SS-7** In terms of licensing, both media players satisfy the criteria for Free and Open Source software.

<table>
<thead>
<tr>
<th>Application</th>
<th>Licensing</th>
<th>ISO Size</th>
<th>Release Date</th>
<th>GTK3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banshee</td>
<td>C (I = 1, F = 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rhythmbox</td>
<td>C (I = 1, F = 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With these constraints, we obtain the following solution specification, which is illustrated in figure 8.

![Figure 8: Solution shelf for the default music app specification.](image)

The context refinement shows that Rhythmbox is the optimal choice for the default music app for Ubuntu 12.04 LTS, because it is satisfying the constraints imposed in the specification.

**Utilizing a SolutionSpecification**

The solution shelf illustrated in figure 8 is one instance of the analysis and intervention for the selection of the default music app in the Ubuntu 12.04 LTS system. Subsequently, this recipe (solution specification) may be treated as a member of another external shelf of recipes within the entire Ubuntu ecosystem. There is no requirement for recipe membership to be unique to a particular shelf, and depending on the granularity of analysis of the component system, we may utilize the same recipe across multiple shelves.

Thus, along with the problem domain shelf, the solution shelf also encapsulates a localized ontology which captures the intellectual concerns influencing the selection of the default music app within the Ubuntu 12.04 LTS system.

Moreover, multiple paths of context refinement may provide a family of related solution specification recipes populating related external shelves. For instance, a DVD re-spin of the Ubuntu 12.04 LTS system may include the Banshee media app since it is free of the ISO Size limitation. This will have further implications for other component systems and their interactions and behavior. Hence, even though shelf categorization need not be unique, a solution specification, or for that matter, any recipe is unambiguous in translation even though multiple pathways of implementation and further processing may exist depending on the system context. The solution specification of figure 8 provides a clear pathway for analyzing the interactions of other component systems which may be affected as a result of the constraint to remove the MonoGTK3 binding from the standard distribution. This approach provides us with the ability to treat each component subsystem and its constituents as a localized and single “control and execution context” while making it possible to conveniently switch between detailed component system analysis and an overall appraisal of the entire system.
Unintended side-effects may evolve due to the interactions between the recipe for the default music app and the rest of the Ubuntu ecosystem. To illustrate this, we consider how the solution specification in figure 8 affects the selection of the default app for note taking.

The external shelves (localized ontologies) for App Categories and some Default Apps are depicted in figure 9. The solution specification for the default music app is now a recipe in the default apps shelf.

![Figure 9: Local ontology for default apps.](image)

To analyze how the choice of Rhythmbox as the default music app affects the selection of the default note app, we perform concern refinement to populate the problem domain shelf illustrated in figure 10.

![Figure 10: The problem domain shelf for selecting the default note app.](image)

Context refinement yields the following solution specification statements.

**SS-8**  
It is a requirement for the note taking app to synchronize with the U1 Sync service. Tomboy supports U1 Sync, while GNote lacks this functionality.

- U1 Sync \( C(1, 1, F = 1) \) Tomboy
- U1 Sync \( C(1, 1, F = 0) \) GNote

**SS-9**  
As seen in [SS-4], Rhythmbox does not have a dependency on the Mono framework, which is not yet fully ported to Gtk3. In addition, GNote is also independent of the Mono stack, while Tomboy has a contextual dependency on the same.

- Mono \( C(1, 1, F = 1) \) Tomboy
- Mono \( C(1, 0, F = 0) \) GNote

**SS-10**  
Consequently, we see the emergence of a contextual relationship between the choice of the Default Music App and the Default Note App.

- Default Music App \( C(1, 1, F = 0) \) Default Note App
SS-11 Following from [SS-8], since GNote does not incorporate critical synchronization features, it is not suitable for selection in the LTS distribution.

GNote  \( C(1 = 1, F = 0) \)  LTS

SS-12 From [SS-10], [SS-4] and [SS-2], we see that the inclusion of Tomboy requires the entire Mono stack, affecting the ISO size.

ISO Size   \( C(1 = 1, F = 1) \)  Default Note App
ISO Size   \( C(1 = 0, F = 0) \)  Tomboy

SS-13 Thus, the choice of the Default Music App results in the exclusion of Tomboy. The relation between these two recipes is a strong constraint.

Default Music App  \( C(1 = 1, F = 1) \)  Tomboy

SS-14 In light of [SS-8], [SS-9], [SS-12] and [SS-13], neither Tomboy nor GNote can be selected as the default note app for the LTS distribution.

Tomboy  \( C(1 = 0, F = 0) \)  Default Note App
GNote  \( C(1 = 0, F = 0) \)  Default Note App
Default Note App  \( C(1 = 0, F = 0) \)  LTS

In this case, owing to the choice of the Default Music App, the Ubuntu 12.04 LTS distribution will not ship with a default note app. The note app can be seen to be a vaguely defined and unimplemented prototype.

The corresponding solution specification is illustrated in figure 11.

![Figure 11](image11.jpg)

**Figure 11: In this recipe, the Ubuntu 12.04 LTS distribution does not ship with a default note app.**

SS-15 On the other hand, if the presence of a default note app is considered as a strong constraint, then in spite of the arguments against it, the only valid choice is GNote. The resulting solution specification cannot be considered an archetype owing to the ambiguity in the choice of GNote (the app lacks critical synchronization features).

Default Note App  \( C(1 = 1, F = 0) \)  GNote

As a result, GNote cannot be considered an archetype for the resultant solution specification (figure 12).

![Figure 12](image12.jpg)

**Figure 12: In this recipe, the Ubuntu 12.04 LTS distribution can ship with GNote as the default note app even though it lacks critical synchronization features.**
SUMMARY AND CONCLUSIONS

In this paper we have presented a novel approach for the analysis and management of large and complex systems by the application of Omnispective Analysis and Reasoning (OAR), an epistemic framework for managing intellectual concerns. We have briefly considered the nature of large systems of systems, and identified in particular, that the analysis, understanding and management of complex systems can be further improved by utilizing the OAR framework to:

1. develop localized ontologies to facilitate switching between detailed knowledge of the component systems and an overall appraisal of the entire system; and
2. develop the ability to treat each component system and its constituents as a local and single control and execution context, which will enable validation of the component systems against overall goals and intent.

In the OAR framework, the process of analysis, design and operation of a system is considered as a workflow formulation, solution specification and implementation. An omnispective appraisal of large systems enables the capture of all identified intellectual concerns in component systems as individual recipes. A problem scenario can then be reformulated in terms of relevant recipes encapsulating the concerns in the component systems, their mutual interactions and the influence they have on one another. The management of these recipes (prototypes, archetypes and constraints in the problem scenario) in shelves provides the ability to organize and use localized ontologies for component systems, which are sufficient for analyzing, formulating and obtaining solution specifications for the given application. Those concerns that are vaguely understood but may be relevant to the outcomes of the system can be recognized.

The OAR processes of concern and context refinement facilitate the verification and validation of intellectual concerns in component systems as well as verifying the interactions between them and formulating their behavior and outcomes as solution specifications in terms of, and conforming to, the OAR recipes. This provides the ability to treat component subsystems as localized control contexts within the larger system scope.

As an illustration of the application of OAR to systems of systems, we consider the process of selecting the default music app for the Ubuntu 12.04 LTS distribution. A solution specification for this is obtained using the processes of concern and context refinement. Localized ontologies and fixing the control context enable the exposure of side-effects that might evolve across the Ubuntu platform due to a particular solution specification for the default music app.

Further studies towards applying the OAR framework to the analysis and design of complex systems are needed to fully explore the scope and extent of the benefits outlined in this paper.

ACKNOWLEDGMENTS

This work is based on initial research supported by the Australian National University, and the Commonwealth of Australia, through the Cooperative Research Centre for Advanced Automotive Technology.

REFERENCES


**BIOGRAPHY**

**Srinivas Chemboli** is a PhD student in the Research School of Computer Science at the Australian National University. His research focuses on the effective capture and reuse of intellectual concerns in scientific workflows.

**Clive Boughton** is an adjunct associate professor at the Australian National University in the Research School of Computer Science. Clive is the director of Software Improvements, a consulting company that produces software for large scale organizations.