Requirements Model Driven Adaption and Evolution of Internetware

LIU Lin[1], WANG JianMin[1], YE XiaoJun[1], LIU YingBo[1]

YANG HongJi[2] & LIU XiaoDong[3]

[1]*School of Software, Tsinghua University, Beijing, China*

[2] *Software Technology Research Laboratory, De Montfort University, Leicester, England, UK*

[3]*School of Computing, Edinburgh Napier University, Edinburgh, Scotland, UK*

Abstract

Today's information systems need to support complex business operations and processes. The development of web-based systems has been pushing up the limits of traditional software engineering methodologies and technologies. Applications are required to be used and updated almost real-time, which allows users to interact and share the same applications over the internet. The applications have to adapt quickly to the dynamic changes in the physical, technological, economical and social environments. As a consequence, we are expecting a major paradigm shift in software and data engineering to reflect such changes in computing environment in order to better address the fundamental needs of organisations in this new era.

Existing software technologies, such as model driven development, business process engineering, online (re)configuration, composition and adaptation of managerial functionalities are being "repurposed" to reduce the time taken for software development by reusing existing software codes. The ability to dynamically combine contents from numerous web sites and local resources, and the ability to instantly publish services worldwide has opened up entirely new possibilities for software development. This is the research mission undertaken by the Internetware project team with memberships being listed in the author list.

In retrospect to the ten years applied research on Internetware, we have witnessed such a paradigm shift, which brings about many changes to the developmental experience of conventional web applications. Several related technologies, such as cloud computing, component computing, cyber-physical systems and social computing, have converged to address this emerging issue with emphasis on different aspects. In this paper, we first outline the requirements that the new software paradigm should meet to excel at web application adaptation; we then propose a requirements model driven method for adaptive and evolutionary applications; and we report our experiences and case studies of applying it to an enterprise information system for large manufactures. Our goal is to provide high-level guidelines to researchers and practitioners to meet the challenges of building adaptive industrial-strength applications with the spectrum of processes, techniques and facilities provided within the Internetware paradigm.

Keywords requirements, information system, Internetware, adaptation, evolution

1 Introduction

Tremendous changes in the Internet environment bring the possibility of building enterprise application software online. The "Internetware", as a new software paradigm, referring to the integration of software systems based on a set of software components distributed over the Internet, together with a set of connectors [3, 38, 40]. The software components, or rather Internetware entities, are autonomous, self-contained and deployed in distributed nodes across the Internetwork. Individual components are able to respond to perceived changes in the environment by means of reconfiguration and

reorganization. Internetware software entities collaborate with one other on demand. It follows a bottom-up and spiral development process, which forms a continuous and iterative composition of various "disordered" resources into an "ordered" software system. Just like the widely adopted component-based paradigm, the major methodological framework of Internetware brings flexibility and reusability in building distributed enterprise information systems. Users may get access to required information and components easily and timely. As new requirements and software assets emerge, existing problem-solution bindings can be updated and further optimized. However, it is yet to be fully realised due to the diversity and constant changes of components requirements, limited reliable sources of components available, and the lacking of a supporting platform matching requirements with available components.

Conventional requirements elicitation and analysis are conducted in advance of design. Once the system under design is deployed, requirements related activities become secondary – handling change requests, while today's web-based systems are required to execute under a more open and dynamic environment, new requirements emerge constantly, and systems keep evolving to cope with these run-time behaviours. There are little existing research work in the requirement engineering (RE) and software engineering (SE) literature handling run-time requirements. The ones we have come through include: Fickas and Feather [24] have studied the significance of monitoring requirements, based on the analysis of two commercial software cases; Jureta et al. [29] have proposed formalism in response to the emergence of services computing and its requirements for a transformation from static requirements to dynamic requirements. A more comprehensive probe into the research issues and challenges in this area is the research roadmap by Cheng et al. on software engineering for self-adaptive systems [15]. All these efforts agree on the importance of the research problem of requirements modelling, monitoring and evolution in an on-going basis for a running, live and ever-changing system, in the sense of, self-adaptivity, automation and minimal disturbance to the running system.

In this paper, we introduce an approach to modelling web-based enterprise management systems and evolving them based on requirements models. Section 2 introduces a generic conceptual modelling framework for web applications using a four-tuple model. Section 3 defines a typology of systems that explains how static systems, reactive systems, adaptive system and evolvable systems differ from each other. Section 4 describes the composition mechanism for Internetware systems, where details about the basic operations are given. Section 5 introduces the adaptation process and three major algorithms. Section 6 uses case studies to illustrate and evaluate the proposed approach. Section 7 discusses related work and Section 8 summarises the contributions of the paper.

2 Modelling Internetware Systems using Goals, Environments, Processes and Strategies

Internetware is a natural evolution of the traditional software paradigm to satisfy the need of Internet as a software component development and running platform. There are many preferable characteristics in this context such as autonomy, evolution, collaboration, polymorphism, reaction and so on. This section aims to convey our understanding to the general requirements of Internetware systems. In other words, the paper proposes a conceptual framework to illustrate the shift of paradigms from the conventional design-time requirements model, to a run-time reactive model considering changes in the environment, to today's on-demand capability provisioning. The fundamental goal of the framework is to point out explicitly what has to be defined prior to running, and what is changeable during run time, how to make the system adaptable to run-time demands without taking the system offline. Since we use a top-down approach, the conceptual framework sets out from very simple concepts, and then is gradually refined and incorporated with more design and implementation knowledge. Here we examine how the characteristics of the Internetware software can be concretized with the proposed approach.

- Autonomy of individual components

Autonomy refers to the capacity of a rational individual to make an informed, un-coerced decision, who determines the responsibility for the consequences of its own actions. Software components, equipped with a goal-environment model facility, are autonomous agents [16, 58, 66] which can determine what goal they pursue, which action plan they may select, based on a set of predetermined strategic rules and criteria. This is independent of other's goals. In other words, the autonomy of an Internetware component is put in place due to the fact that its goal model, environment model, action space and strategies are maintained and dealt.

- Collaboration

Internetware collaboration [14, 59] means that the software entities are aware of the existence of others, and could collaborate to achieve a common goal. There are delegation and communication mechanisms between them. The states of other components are observable by other components, as environment variables. Each of them has an associated goal-action refinement structure, these goal-action refinement structures are interconnected, and maintained through protocols supporting the requesting, publication, searching, binding, delegation and revoking relationships among

individual components.

- Polymorphism

Polymorphism is related to inherent diversity, variation and adaptation; it usually functions to retain variety of form in a population living in a varied environment. Polymorphism is a result from evolutionary processes. Polymorphism in Internetware refers to the capabilities of Internetware components to inherit a stable set of behaviour from its conventional software component ancestor, which exhibit different interfaces or quality level while needed. It has a switching mechanism that determines which "morph" (customised version, variation) is shown in accordance to the environment. In this paper, this switching mechanism is the matching, selection, and planning facility.

- Reaction

Internetware entities should have the ability to sense the environment and provide useful information for adaptation and evolution. The running platform supports the monitoring of the environment variables. The environment model contains related environment variables to be monitored. Based on the environment and the goal of the software, the software entity could select an action accordingly. In a reactive systems scenario, the reaction is the result of non-deterministic goal refinements at design time. In other words, the achievement of goals depends on the run-time states of the environment variables.

- Evolution

Software evolution focuses on adaptation and migration. In terms of adaptation, it requires modification of a software product performed after delivery to keep a software product usable in a changed or changing environment. It takes place when there is a known requirement for change. The evolution property of Internetware requires it observes changes in the environment and responses to the changes based on its current capabilities (the actions and adaptation strategies). The Internetware component could evolve dynamically according to the environment and user's requirements. The evolution occurs when the current functionality, performance, organization of Internetware cannot sufficiently satisfy the need of the environment.

3 Matching and Composing Internetware Components

When proper components implementing functions in a required process are chosen, component providers need to integrate and develop a system to fulfil user requirements in reality. Usually, the chosen components should be composed according to certain business constraints and be executed in specified temporal order. The resulted system will be composed of two major parts: functionality and quality. The functionality of system describes what the system does and quality indicates how well the system performs. In particular, quality dedicates to the non-functional properties of system, such as cost, performance, reliability, security and so on. In a component system, composite components also have quality, which is essentially the function of quality attributes of its component components. If the quality of a component system satisfies the non-functional requirements (softgoals) of users, the system is suitable for the current context. Otherwise, it has to take some actions to adapt itself according to the users' soft-goal and its operation environment. To help effectively organize and compose components and build adaptive application, we will present an adaptive composition framework in this section, including a meta-model and a set of composition rules.

3.1 A Meta Model

The basic elements of our proposed adaptive composition framework are shown in Fig. 1. According to the meta-model, component can be characterized as *component type* and *component implementation*. A *component type* is the abstraction of the functionality of a group of concrete components, which can be either atomic or composite, has both functionality and quality attributes. All *concrete components* associated with a component type implement the same functionality but may have different component interfaces and quality level.

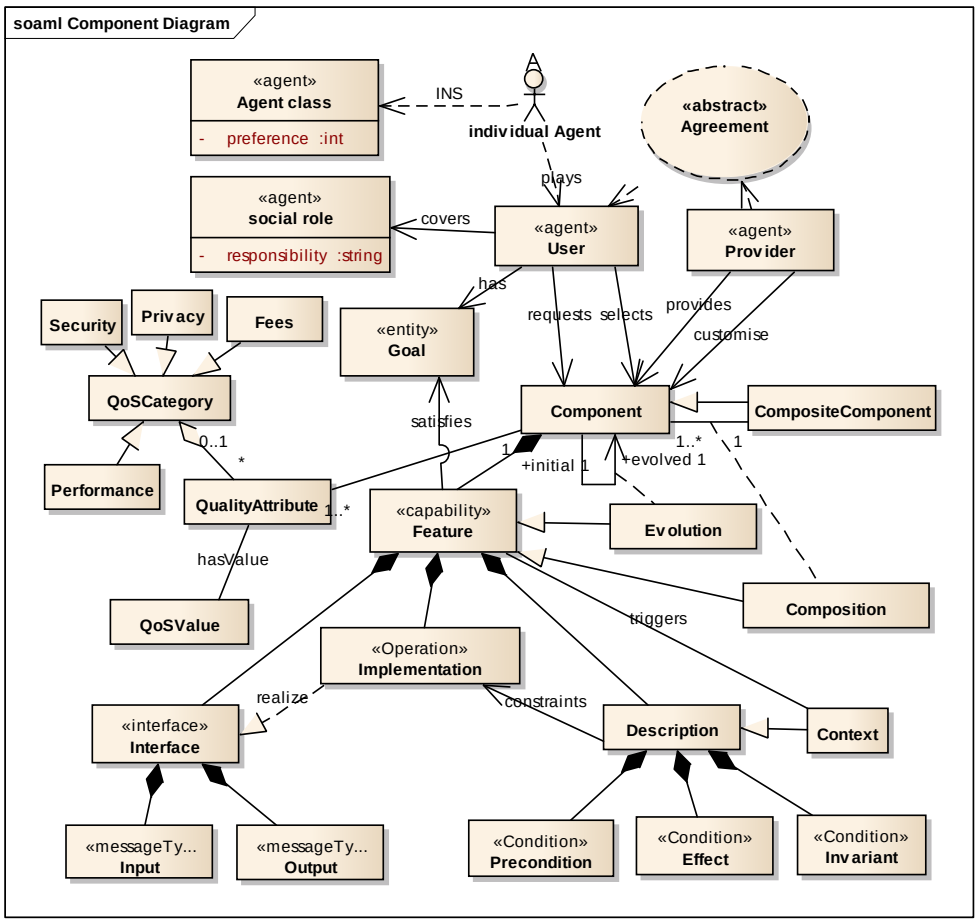


Fig. 1 Meta-Model of Internetware Components

The model also manifests that component functionality needs context constraints, QoS is able to satisfy quality constraints, which are the operationalisation of softgoals. The operation environment influences both the context constraints and the status of component. Such influences will be further propagated to component functionality and quality constraints. To further explain the framework, we define a set of functions in Table 1 and to explain the relations among functionality, quality, quality constraints and context constraints. We use the symbol Function to represent the functionality of both component type (component class) and concrete component (component instance). If S is a component type and s_i one of its instances, then *Function* (s_i) |= *Function* (S). That's to say, each concrete component is able to fulfil the functionality that its corresponding component type declares.

Table 1. Major Functions Used in the Meta Model

Symbol of Function	***Meaning***
Function (s)	*The functionality of a component class or a component instance s*
Quality ($[q_{1...m}]$, s)	*A set of value or range measurements of component s on quality attributes* $q_{i...j}$ $(1 \le i \le j \le m)$ *The omission of quality attribute is defaulted as the complete set* $q_{1...m}$
Constraint ($[q_{1...n}]$, G_s)	*A set of derived quality constraints of a soft-goal* G_s *on specified quality attributes* $q_{i...j}$ $(1 \le i \le j \le n)$. *The omission of quality attribute is defaulted as the complete set* $q_{1...n}$
Minimal/ Optimal/ Maximal ($[q_{1...n}]$, G_s)	*User desired demand degree of soft-goal* G_s *on each specific quality attribute* $q_{i...j}$ $(1 \le i \le j \le n)$. *The omission of quality attribute is defaulted as the complete set* $q_{1...n}$
Constraint (s)	*A set of constraints imposed on component s's operation environment*

We represent quality through using quality attributes and use the function *Quality* to assigned values for each quality attribute of concrete components. For example, if the average response time of component s is 1 second, then *Quality* (*average response time*, s) = 1 *sec*. Multiple quality attribute constraints can be expressed as follows: *Quality* $(q_1 \ldots q_m, s)$ = {*Quality* $(q_1, s) \cup \cdots \cup$*Quality* (q_m, s)}. We define *Quality* (q, s_i) |= *Quality* (q, s_j) if component s_i has better performance on the attribute q than component s_j does. Based on this definition, we can deduce that *Quality* (s_i) |= Quality (s_j) if on each attribute q_l of a fixed attribute set Q, *Quality* (q_l, s_i) |= *Quality* (q_l, s_j) holds. It means component s_i has better quality than component s_j.

Meanwhile, each soft-goal G_s in the goal model is operationalised as quality constraints through the symbol

function *Constraint*. A concrete component *s* satisfies G_s on attribute *q* if *Quality* (*q*) |= *Constraint* (*q*, G_s). E.g., soft-goal *responsiveness* could be stated as constraint "*response time < 2sec*", and a component *s* satisfies this soft-goal if *Quality* (*response time*, *s*) = 1 *sec*, because "*response time = 1 sec*" entails the constraint "*response time < 2 sec*". Similarly, the annotation *Quality* (*s*) |= *Constraint* (G_s) is used to represent that a component *s* is able to satisfy a soft-goal G_s. It is true if for each quality attribute *q* in the quality set *Q* we have *Quality* (*q*) |= *Constraint* (*q*, G_s) holds. It is common that a soft-goal needs the contributions from more than one component. Also, for each soft-goal related quality attribute, we define frequently-used constraints, such as *minimal (q, G_s), optimal (q, G_s), and maximal (q, G_s),* to express user desired demand degree for soft-goal G_s on quality attribute *q*. In addition, components have associated context constraints on the expected value, or range for their operating environment. The function *Constraint* maps each concrete component to a set of associated constraints. For example, if the component "*Provide Video Tutorials*" only requires "*Network speed ≥ 50kbps*", then *Constraint* (*Provide Video Tutorials*) = {*Network speed ≥ 50kbps*}.

3.2 Component Lifecycle

In our framework, components are assumed to be maintained in a component repository and monitored to ensure their Quality requirements are met. Since the (concrete) components in the repository are the basic building blocks of component systems, thus they play a significant role in component composition for fulfilling user goals and in component adaptation for handling change, especially their lifecycles.

To describe the lifecycle of components, we define a set of operations that could change the state of components as shown in Table 2: the status of a concrete component changes through *new(s)* and *remove (s)*; the Boolean function *availability(s)* to indicate whether or not the component s is available; the Boolean functions, *bound(*s) and *done(s)* indicate whether a concrete component is bound to a component contract, or whether it has completed its execution. Fig. 6 models the life cycle of concrete components in terms of a state transition diagram.

Table 2. The Set of Component Manipulating Operations

Symbol of Function	***Meaning***
New(*s*)	*A new component instance s is added to the component pool*
Availability(*s*)	*A Boolean value indicating whether component s is available.*
Bound(*s*)	*A Boolean assignment indicates whether a concrete component is bound to a component contract*
Done(*s*)	*A Boolean value indicates whether a concrete component has completed its execution*
Remove(*s*)	*Remove a component s from the component pool*

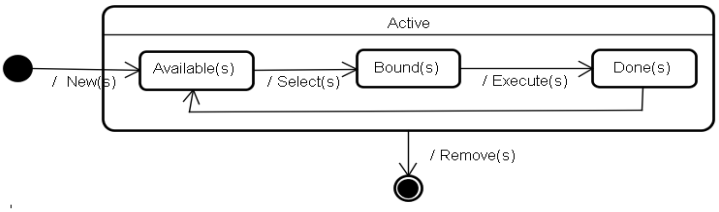


Fig. 2 The Lifecycle of a Concrete Component *s*

3.3 Component Composition Rules

Using requirement modelling techniques adopted from *i**, we are able to derive alternative sets of tasks which's execution (in some unspecified order) fulfils a user target goal *G*. Each such specification $S_p = \{T_0 \ldots T_m\}$ has the property that S_p |= *G*. Each task in a specification can be mapped to an abstract component, which is then instantiated at runtime by assigning a corresponding concrete component to it.

Based on the refined goal graph structure and the hidden temporal/casual constraints between all abstract components in specification S_p, we can derive an abstract component composition structure. Specifically, we use the following set of composition operators: *seq(s^+), sel(s^+),par_and(s^+), par_or(s^+),* where *s* denotes an atomic component, s^+ denotes a set of one or more components, and meanings of the operators are introduced in Table 3.

Table 3. Meaning of Operators

Operators	*Meaning*
seq(s^+)	*Sequential execution of atomic components* s^+
sel (s^+)	*Conditional selection of atomic components* s^+
par_and (s^+)	*Concurrent execution of atomic components* s^+*(with complete synchronization)*
par_or (s^+)	*Concurrent execution of atomic components* s^+ *(with 1 out of n synchronization)*

We compose functionalities, where, sequential *seq*(s^+), conditional selection *sel*(s^+), concurrent execution with complete synchronization *par_and*(s^+) and concurrent execution with 1 out of *n* synchronization *par_or*(s^+) are used. Sequential composition *seq*(s^+) and concurrent execution *par_and*(s^+) are derived from AND-decomposition of goals, while conditional selection *sel*(s^+) and the other concurrent execution *par_or*(s^+) are derived from OR-decomposition of goals. When a user request quality level exceeds the quality of any individual components of S_i, the system may run compose component (say s_{ij}, s_{ik}, s_{il}) concurrently (with complete synchronization) to meet the demand, by triggering *par_and*(s_{ij}, s_{ik}, s_{il}) composition as described in Table 4.

Table 4. The *par_and* Composition

Operation	*Par_and* (s_{ij}, s_{ik}, s_{il})
Precondition	*Constraint* (s_{ij}, s_{ik}, s_{il}) *holds* $\wedge$ (*Function*(s_{ij}) |= *Function*(s_{ik}) $\wedge$ *Function* (s_{ij}) |= *Function*(s_{il})) $\wedge$ *minimal* (*throughput*, G_s) $\geq$ *Quality* (*throughput*, s_{ij}) $\wedge$ *minimal* (*throughput*, G_s) $\geq$ *Quality* (*throughput*, s_{ik}) $\wedge$ *minimal* (*throughput*, G_s) $\geq$ *Quality* (*throughput*, s_{il}) $\wedge$ *available* (s_{ij}) $\wedge$ *available* (s_{ik}) $\wedge$ *available* (s_{il})
Trigger	*Throughput* (*Par_and* (s_{ij}, s_{ik}, s_{il})) $\geq$ *minimal* (*throughput*, G_s)
Effect	*Bound* (s_{ij}) $\wedge$ *Bound* (s_{ik}) $\wedge$ *Bound* (s_{il})

When a user needs a high-reliability component, while existing component has a certain level of risk, the system may execute composite component (for example, s_{ij} and s_{ik}) concurrently (with *1* out of *n* synchronization) to meet the demand, by triggering *par_or*(s_{ij}, s_{ik}) in Table 5.

Table 5. The *par_or* Composition

Action	*Par_or*(s_{ij}, s_{ik})
Precondition	*Constraint* (s_{ij}, s_{ik}) *holds* $\wedge$ (*Function* (s_{ij}) == *Function* (s_{ik}) $\wedge$ *Quality* (*reliability*, s_{ij}) $\leq$ *optimal* (*reliability*, G_s) $\wedge$ *Quality* (*reliability*, s_{ij}) $\geq$ *minimal* (*reliability*, G_s) $\wedge$ *Quality* (*reliability*, s_{ik}) $\leq$ *optimal* (*reliability*, G_s) $\wedge$ *Quality* (*reliability*, s_{ik}) $\geq$ *minimal* (*reliability*, G_s) $\wedge$ *available* (s_{ij}) $\wedge$ *available* (s_{ik})
Trigger	*Quality* (*reliability*, *Par_or* ($s_{ij}, s_{ik},$)) $\geq$ *Quality* (*reliability*, s_{ij}) $\wedge$ *Quality* (*reliability*, *Par_or* (s_{ij},s_{ik})) $\geq$ *Quality* (*reliability*, s_{ik})
Effect	*Bound* (s_{ij}) $\wedge$ *Bound* (s_{ik})

4 Adapting Components

4.1 Adaptation Strategies

When a component system is built up and is put into operation, changes could occur to both user requirements and component operation environment. In order to provide satisfactory components to users effectively, a component system should be aware of the requirements and environment changes, and be adaptive to such situations. In response, adaptation actions, strategies and processes should be in place.

Changes are mainly from user requirements, especially their quality requirements. When user requirements change, the goal model and its refinement will be updated accordingly, and at last the lowest level of tasks will be updated: new task(s) are derived out, or existing task(s) are removed. As a result, the abstract components should be re-composed according to the new goal graph structure and concrete component will be re-selected. For example, the users of *online shopping component* usually want the goods to be delivered quickly, but in some cases they desire the shipped goods be

packaged well and not be damaged in delivering. To resolve this problem, a new soft-goal "*Good Shipping Quality*" and a new task "*Deliver with High Quality*" is probable to be added, and the *shipping component* will be reselected.

Changes also come from components' operating environment. Usually, components are maintained in a repository and monitored to ensure their quality requirements are met. This is accomplished by monitoring component behaviour (e.g. average user waiting time, response time, and the percentage of users being served) and a set of environment variables *E* (e.g. CPU utilization, memory usage, bandwidth availability, network stability, or number of concurrent online users). Component environment influences both the status of components as well as user goals and their refinements. For example, if one of the constraints associated with component "*Provide Video Tutorials*" is "*Network speed* $\geq$ *50kbps*", and environmental variable "*Network speed*" has a value that is less than 50kbps, then the component is unavailable.

At last but not least, the component repository would also change dynamically, such as adding a new concrete component, removing an existing component and recomposing a component. When changes occur, component system needs to perform adaptations to satisfy given requirements. Adaptive actions are needed to handle changes in user Quality requirements, environment variables, or components. Usually, such actions include reorganizing the structure of abstract component composition, reselecting and recomposing concrete component(s).

The composition actions shown in Table 4 could be reused in component adaptation. Adaptive actions have associated preconditions, triggers and effects with usual semantics. All of them consist of propositions constraining environmental variables and the internal state of a component. E.g., for an abstract component s_i, when a new atomic component s_{ik} emerges, s_{ij} is an atomic component of a composite components that suddenly becomes not available, or waits to be executed at the moment, if *Function* (s_{ij}) |= *Function* (s_{ik}), and *Quality* (s_{ij}) |= *Quality* (s_{ik}), the system may take the "*Substitute*" action as shown in Table 11.

Table 6. Substitute Operation

Action	*Substitute* (s_{ij}, s_{ik})
Precondition	*Constraint* (s_{ik}) *holds* $\wedge$ (*Bound* (s_{ij}) $\wedge$ (*Function* (s_{ik}) |= *Function* (s_{ij}) $\wedge$ (*Quality* (s_{ik}) |= *Constraint* (G_s) $\vee\neg$*Available* (s_{ij}))
Trigger	*Quality*(s_{ik}) |= *Quality* (s_{ij}) $\wedge$ *Available* (s_{ik})
Effect	*Bound*(s_{ik}) $\wedge$ *Remove*(s_{ij})

Based on the origin of changes, strategies are formed to decide what adaptive actions to be carried out, based on the user's goals to be satisfied, the environment conditions to be monitored and the component repository dynamically changing.

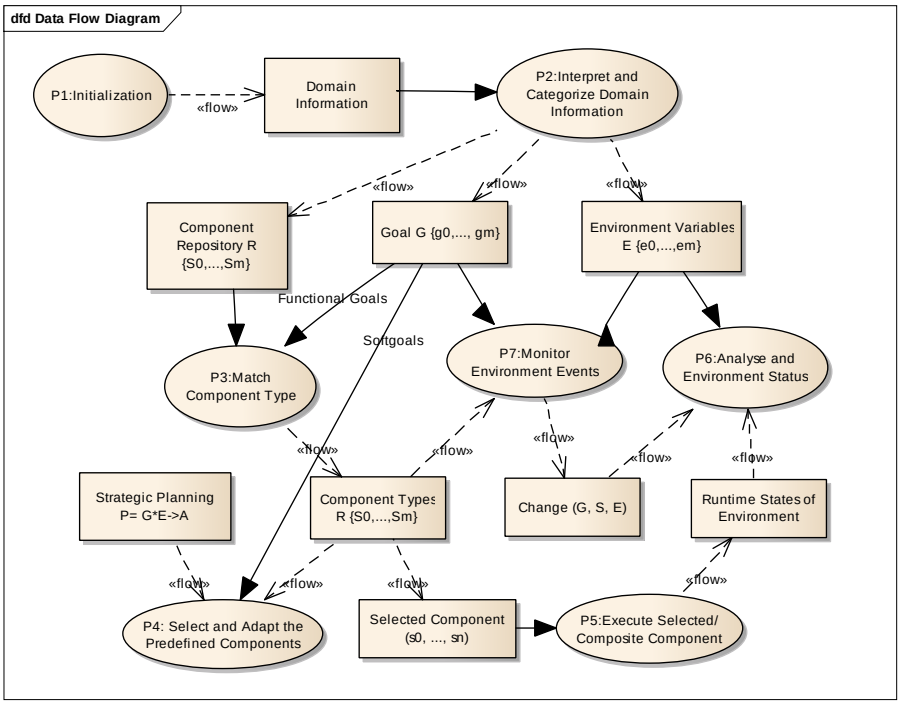


Fig. 3 A Generic Monitoring and Adaptation Process Model

4.2 Adaptation Process

This section introduces a composite adaption process (Fig. 3), which contains a feedback loop executed iteratively to satisfy the changes detected from monitoring. The changes from user soft-goals, system operating environment or component repository are viewed as three possible triggers to the substitution or re-composition of components. Therefore, adaptation should be taken trying to keep the constant satisfaction of user requirements and the operations of composite system. The main procedures associated with the process are Initialization (P_1), Interpret and Categorise Domain Information (P_2), Initialize Predefined Components (P_3), Select and Adapt Predefined Components (P_4), Execute Selected Components (P_5), Analyze Environmental Variables (P_6), and Monitor for Events (P_7). Next we introduce these proposed procedures in detail.

Step 1: The initialization module (P_1) receives inputs from users and environment, such as user goals and environmental variable values.

Step 2: The domain information is interpreted and categorized (P_2). The input goals are used to build goal model whose leaf goals can be matched with and be used to select components. Also, environmental variables will be monitored.

Table 7. Matching Algorithm

Algorithm 1: MatchGoalwithComponentFunction (ComponentRepository S, GoalGraph G, GoalComponentMap GC_p)
Input: ComponentRepository S, GoalGraph G,
Output: a set of identified goal and component map GC_p

```
01  Goals G= ϕ
02  GoalComponentMap GCp =ϕ, Temp =ϕ
03  Goals.add(G)
04  while Goals!=ϕ
05      Goal  g_t = Goals.get()
06      if ∃ s_i∈S & Function(s_i) ⊇ Function (g_t)
07          Goals.remove (g_t)
08          GC_p.add(<g_t, s_i>)
09      else if g_t is or-decomposed
10              for each g_i ∈ or-decompose(g_t)
11                G= G.replace(g_i, g_t)
12              MatchComponentType(R, G, Temp)
13              GC_p.=GCp∪Temp
14            end for
15      else if g_t is and-decomposed
16        for each g_i ∈ and-decompose(g_t)
17        G = G.add(g_i)
18          MatchComponentType(R, G, Temp)
19        GC_p.=GCp∩Temp
20      end for
21  end while
22  return GC_p
```

Step 3: The component repository is categorized as a set of abstract components, each of which includes a set of concrete atomic components. The Match Abstract Components operation (P_3) match the abstract component $\{S_0,\ldots, S_n\}$ from the component repository with specification $S_p\{T_0,\ldots,T_m\}$ refined iteratively from the user's goals. Algorithm 1 defines the matching process, in which component repository R and goal graph G are input parameters, while S_a contains the set of identified abstract components.

Step 4: The Select and Adapt Concrete Components operation (P_4) selects atomic components or compositions thereof from the abstract component according to user soft-goals. If the Monitor operation (P_7) identifies certain changes, the selection module (P_4) undertakes adaptive action to tackle the problems according to the current strategy (R, G, E, A). The Select and adapt the concrete components (P_4) would output a series of selected components. Algorithm 2 (Table 8) defines the process (P_4), in which component repository R and abstract components A are input parameters.

Table 8 Selection Algorithm

Algorithm 2:SelectComponentInstance (ComponentRepository S, GoalComponentMap S_p , Goals G)
Input: ComponentRepository S, GoalComponentMapS_p, Goals g_0
Output: a set of identified ComponentInstancesST

```
01  Components ST, DN
02  MatchGoalwithComponentFunction (ComponentRepository S, GoalGraph G, GoalComponentMap GC_p)
03  For each g ∈ G
04    Si = GCp.getComponents(g)
05    For each s_ij ∈ Si
```

```
06      if Quality(sij) |= Quality(g)
07            ST.add(sij)
08          else
09            DN.add(sij)
10       end for
11      if ST= ϕ
12       compose ( Si, g0) by (seq(), sel(), par_and(), par_or()
13          if Quality(scomp)|= Quality(g)
14            ST.add(scomp)
15          End for
16      return ST
```

Step 5: Selected components generated in Step 4 are now executed (P_8). The execution affects the run-time environment, whose changes can be monitored. And when abnormal behaviours are observed, the system will trigger predefined actions to maintain the expected effect.

Step 6: The runtime status is analysed and new environmental variables may be obtained (P_6). If current values do not satisfy expected values of environmental variables, the Monitor for events operation (P_7) identifies the problem.

Step 7: During the execution of the component process (P_5), user goals, environmental variables and component repository are allowed to change. These changes (G, S, E) need to be captured and submitted to the select and adaptation engine for concrete components (P_4).

Table 9 below describes the adaptation process, the input of which is Change(G, S, E), componentAssembly is the major data object, both appears as an input, and also as a returned parameter, the algorithm captures changes, restructure and compose components processes to adapt to changes to ensure the effective execution of components.

Table 9 Adaptation Algorithm

***Algorithm-3*: *Adapt* (Goal *G*, ComponentRepository *S*,** *GoalComponentMap GC*)
Input: ComponentRepository S, GoalGraph G,
Output: ComponentRepository S', GoalGraph G'

```
1     if   new(gn)
2            G=G.∪{gn}
3          then MatchGoalwithComponentFunction (S, {gn}, GoalComponentMap GC)
4          if ComponentInstances CI = SelectComponentInstance (S, GC, G) ==ϕ
5              S=S ∪.compose( S, gn) using    structure(Seq, Sel, Par_and, Par_or)
6              return
7          if   remove(go)
8                   G= G\{g0}
9          if ComponentInstances CI = SelectComponentInstance (S, GC, {g0}) !=ϕ
10         For ∀si ∈CI, if getGoal(si, GC) \{g0} == ϕ
11             S=S \ {si};
12           else S=S \.remove( S, gn)
13               if   new(sn)
14                  MatchGoalwithComponentFunction ({sn}, G, GoalComponentMap GC)
15               if Goals Gsn = GC.getGoal(sn) ==ϕ
16                    G = G∪Function (sn)∪Quality(sn)
17                  return
18          if   remove(so)
19               S= S\{s0}
20           Goals Gsn = GC.getGoal(s0) !=ϕ
21               if ComponentInstances CI =SelectComponentInstance (S, GC, Gsn)== ϕ
22               G= G \{gsn}
23           return
```

As shown in Fig. 3, the steps P_4, P_5, P_6 and P_7 form a feedback loop that executes iteratively to support run-time adaptation. Those changes on goals and component repository come from the user and the platform correspondingly, so they are not part of the feedback loop.

5.Internetware Testbed and Case Studies

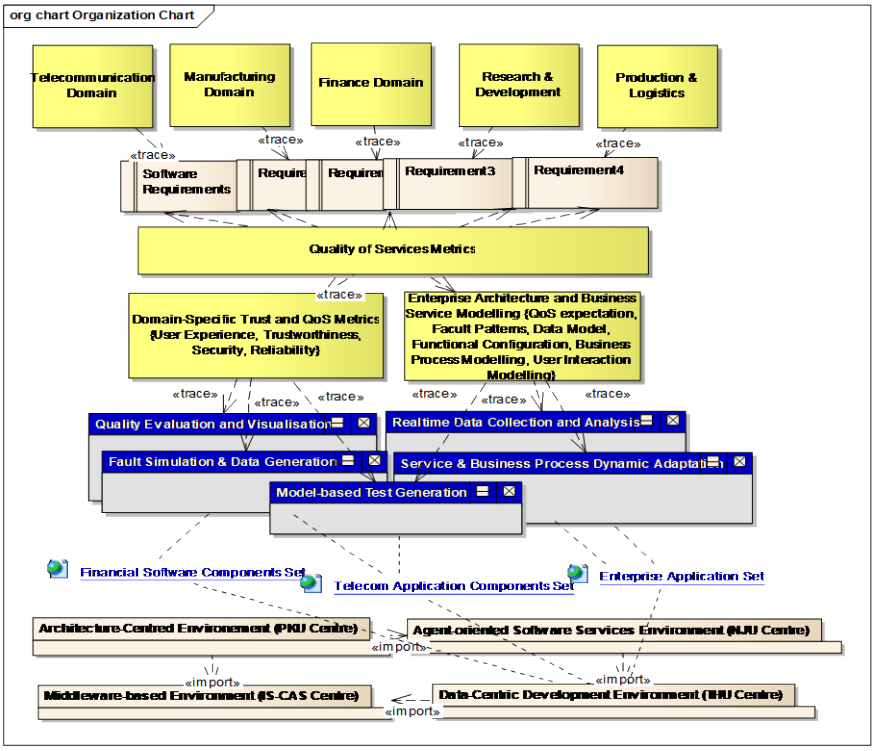


Fig. 4. Architecture of the Internetware Testbed

5.1 Internetware Testbed

Most web-based applications potentially target users all over the world; it is very difficult (if not impossible) to predict the access pattern of enterprise applications before they are ready to serve users. To cope with peak workload on demand or cater for potentially high growth rates, service providers often over-provision their servers by as much as five hundred percent, which may lead to a considerable waste of resource for normal workload, not to mention the management cost spent on the over-provisioned infrastructure. Recent years, cloud computing emerges as a new paradigm for hosting IT services over the Internet. From cloud users' perspective, cloud provides a seemingly unlimited resource pool. Users can apply for infrastructure from this pool at any time and use it through network; they will only be charged for the actual resources used per unit time. The elasticity of cloud-computing infrastructure and the pay-as-you-go price model attract more and more enterprises to deploy their applications to a minimal set of cloud-computing infrastructures, which will later be scaled in or out to cope with the workload fluctuation. Cloud computing and dynamic resource provisioning has been a hot topic recently in both industry and academic fields. For examples, Amazon, Microsoft and Google all propose their own Cloud products and regard the infrastructure, the platforms and the software as services.

Based on the collaborative project teams' research results on Internetware theory and model and the run-time support environment, methodology, techniques and tools, a complex internetwork software deployment, simulation and evaluation platform is constructed, called Internetware Testbed[1]. This Testbed is an environment in which software components conforming to the Internetware paradigm can be implemented. It is also an execution environment configured for Internetware function conformance test, various quality assurance test or the performance evaluation under configurable workloads. To construct the Testbed, project teams developed an instance of IaaS based on virtualization technology. The purpose of the Testbed is to support both Internetware innovation and Internetware applications research within a common experimenter platform. It can address the requirements of Internetware characteristics simulation such as autonomous, evolutionary, cooperative, polymorphic, and context-aware, as well as for Internetware conformance test, various QA test or the performance test under configurable workloads. The system architecture of this platform includes 4 physical clusters residing in four project partner organisations, connected locally via Local Area Network, via Wide Area Network between nodes, which is further extensible. The software systems architecture deployed is a heterogeneous cloud computing environment with over 1000 virtual machines running in parallel across the network. Internetware development and management tools, as well as application prototypes in different domain are deployed in this environment, which can support more than a million system users working online at the same time. Fig. 4 illustrates the system deployment model of the Internetware Testbed.

1 http:// icloud.internetware.org.

5.2 Model-driven Development and evaluation of Internetware Applications

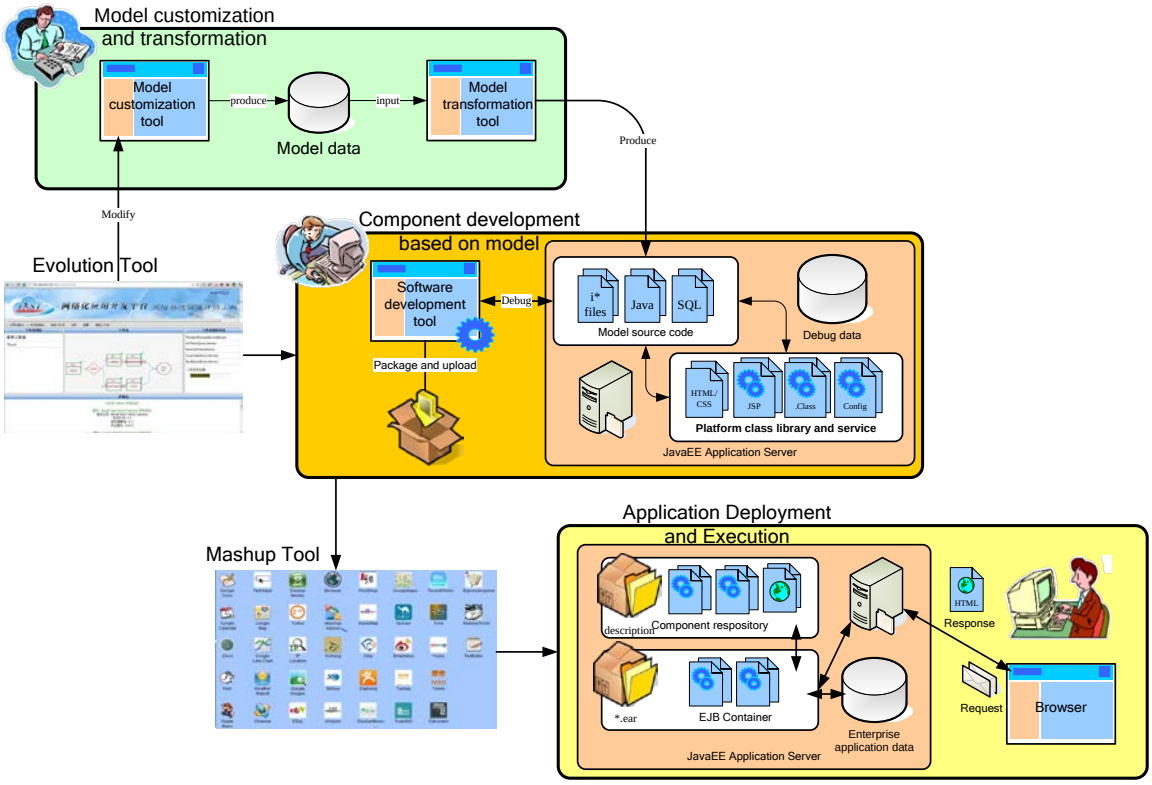


Fig. 5. Development and Evolution Life Cycle of Internetware Applications

The application development and deployment tool set is depicted in Fig. 5. It uses a collaborative adaptation model as. We assume that the environment consists of software agents who have their own goals to fulfil, at the same time provide certain functionalities through which agents form a collaborative network to serve their goals collectively. As indicated in the above algorithms, we evaluate Internetware components from two perspectives: one is functional satisfaction of components, which involves the satisfaction of all alternative ways of delivering the services given functional and the other is the satisfaction level of non-functional goals. In each application domain, there can be several Internetware components offering similar functionalities. Depending on their concrete implementation, this contributes to component consumer satisfaction differently, this in return impacts agents' decision. Accordingly, we need to evaluate the satisfaction degree of each component and rank them accordingly. Here, we introduce the steps of the functional satisfaction evaluation process. The *i** modelling framework is adopted to model and evaluate alternatives, and a simple logistics component example to illustrate our proposed method.

Step 1. Model the requirements of client-end application components.

In the Internetware environment, component delivery involves two kinds of agents: consumers, who need to achieve their goals, and suppliers, who can provide specific components to others. Requirements determine the functions that need to be bundled in a component, e.g., "Sell product", and for qualities that are expected by consumers, e.g., "Low cost". There are many factors that impact the final goal, functional or non-functional. Functional requirements indicate what the component provider needs to achieve. Non-functional requirements, on the other hand, indicate the criteria by which alternative components are evaluated. So, for candidates match the functions of a given goal, evaluation is mainly focused on non-functional properties.

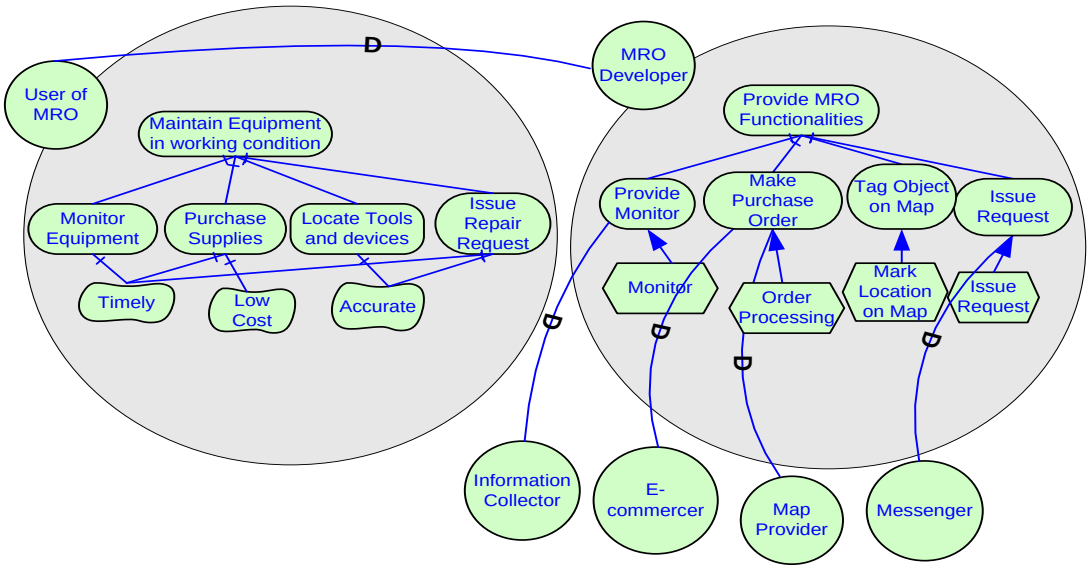


Fig. 6. Modelling MRO high-level requirements in *i

Fig. 6 is an *i** SR model of a generic MRO application. User has goals, such as monitoring equipment, purchase supplies, locate tools and devices, issue repair requests, and also softgoals such as purchase supplies at low cost, and timely, locate tools and devices accurately, issue repair request timely and accurately, etc. To achieve these goals, application developer needs to execute tasks, such as provide monitoring function, make purchase order, tag tools and devices on map, . There are dependencies between the two actors and other actors if external components are used or referenced. Fig.9 illustrates the relationship of these actors. To evaluate the satisfaction degree of each candidate, we should find out all the quality attributes concerned by the customer. From Fig. 9 which shows the requirements of consumer, we can find criteria for evaluating components. If there are plenty of quality attributes to be handled and they have non-trivial relationship to others, correlation analysis should be conducted to identify key factors. In our example, we consider three attributes contributing to component satisfaction: time, cost, and accuracy.

Step 2. *Quantify the importance of each quality attribute*.

There are different methods to quantify the weights of factors identified in the requirements model above, such as the Delphi technique [19], Analytical Hierarchy Process (AHP) [57], Decision Alternative Ratio Evaluation (DARE), and Fuzzy Synthetic Evaluation. Among these methods, Delphi and AHP are best known. The general process for the Delphi technique includes several steps: a) a group of experts give their evaluation independently; b) evaluations are collected and sent out to experts; c) experts refer to other evaluation and re-evaluate; d) repeat step b) and c) when experts stop changing their attitude. The result has good reliability, but assumes expert availability, often a strong assumption. AHP works by comparing relative importance of every factor pair, followed by calculation of relative weights. It can be easily conducted and the process can be understood not only by experts but also by general decision-makers. People can choose among these methods, or others. Once the chosen method has been applied, the weights of these four factors can be estimated**.**

Step 3. *Evaluate candidates based on existing knowledge related to the quality contributions*.

It is very hard to quantify satisfaction degree for these quality factors which are “soft” in nature. Domain expert knowledge is widely used in this situation and many algorithms are proposed based on experts’ knowledge (such as set-valued statistics approach).

Step 4. *Calculate the satisfaction degree of each candidate.*

Given the weights of factors and component satisfaction for each factor, a straightforward way to evaluate the overall score is by their weighted sum. An algorithm proposed in [41] applies Ordered Weighted Averaging (OWA) Measure to Logic Scoring Preference (LSP) method. It solved the problem of dynamic returning aggregation function (at the same time, it represents factors’ logic relations such as simultaneity and replaceability). This method also has been used for automated web services selection. If we add weights and evaluation values (the format is “weight: evaluation”) on the model showed in Fig. 10 and mark the final score of candidate component (we use weighted sum based on data obtained from step 2 and 3). Following these four steps, we can evaluate the satisfaction degree of each candidate Internetware component.

5.3 Enterprise Applications Example

A case study of the Enterprise Product Lifecycle Management domain is used to further illustrate the proposed approach. Product lifecycle management is the process of an enterprise managing the entire lifecycle of a product from its conception, design, manufacture, to service and disposal. A typical PLM product supports for systems engineering, product and portfolio management, product design, manufacturing process management and product data management. There are a rich set of components for architects to choose from in order to form an integrated solution for an enterprise. While all systems are providing similar capabilities to organizations, there are also different ways to construct a specific system. All these differences can be traced back to the variations in customer requirements or constraints in the organizational environment. When project team receive customer requests, components are selected based on basic information about the usage context of the customer organization. For instance, the size of the enterprise, the industry sector the organization belongs to, the usual functionalities needed by the customer organization, and the amount of technical efforts and system expenses the organization can afford, etc. In order to make a rational selection among different design alternatives and to identify a group of features that can best fit the customer's needs, we have identified the following non-functional goals for these products:

- Fitness: the degree of a product fitting the intended purpose and market
- Time-To-Market: Fast delivery of products to market
- Project Risk: Risks in product development and project execution
- Cost: Reduction of cost
- Productivity: Improved productivity of product development and design
- Rich Know-how: the richness of know-how for the product lifecycle management
- Quick Information Retrieval: the ability to retrieve information when needed
- Customization: Customization cost
- Integration: Integration cost
- Training: Training cost
- Security Risk: Security risk likeliness

We have also identified the following components making up the final software system.

- ERP (Enterprise Resource Management): Since enterprise information system use ERP software to conduct resource planning and management, they may either consider an ERP-included PLM solution, or adopt their existing ERP system. This may have influence on the easiness of information retrieval and level of integration among system components.
- MRO (Maintenance, Repair and Overhaul): For many enterprises involving a large number of devices, the maintenance, repair and overhaul of these essential assets is a major task. The decision on including a MRO component will influence the efficiency and accuracy of management process.
- Globalized Collaboration support. Lack of globalized collaboration support will hamper the productivity of distributed teams.
- Virtual Design: Virtual Design is one of the features under the umbrella of global community collaborated design supporting environment. It has influence on the communication efficiencies of product development.
- Visual Product: Visual Product Design is also under the umbrella of global community collaborated design supporting environment. It has influence on the communication efficiencies of product development and visualization.
- Secure PLM: There are a group of functional features implementing the Secure Product Lifecycle Management and data communication. It mainly influences the level of security risks for the overall system.
- Product Development: Product Development Management is the main feature of the system. It determines the efficiency and productivity of the organization.
- Retirement: Product Retirement Management is one of the main features in the product lifecycle management, which plays an essential role in providing data for compliance to recycle and waste disposal regulations, which could save cost if proper management process and policies are made.
- Service Sustainability: Service Sustainability manages the process after a product is put into use or deployed, which could influence the satisfactory level of customers and maintain the image of an enterprise.
- Portfolio Planning: Portfolio Planning Capability is an essential feature provides strategic planning for a product as well as maintaining the portfolio for products in the same product line. It provides necessary information for customization, influence the effort and efficiency of customization.
- BOM Mgmt: Bill of Material Management provides unified product data management support to the overall system, it is the key data component to be accessed and retrieved by most information queries. Its existence will influence the easiness of integration and information retrieval.
- Content Management: Content Management functionality provides support to the management of contents not limited to structured data, this include graphical data files required by the CAD feature and simulations, etc. It influences the efficiency of accessing such data.
- Report Analysis: Report and Analysis Functionality mainly provide decision support data for high managements. Thus, it influences the quality of knowledge support and accessibility to required information.
- Community: Community collaboration support allows a global organization to work collaboratively across the

communication barriers.

- Simulation: Simulation and Process Management provides visual support to difference analysis results, and makes the change impact analysis easier.
- Requirements Management: Requirements Management Supporting Tool plays an important role for managing product requirements and tracing requirement changes.
- Compliance: Management and assuring the Compliance to different industrial Regulations will help control the risk caused by non-compliance.

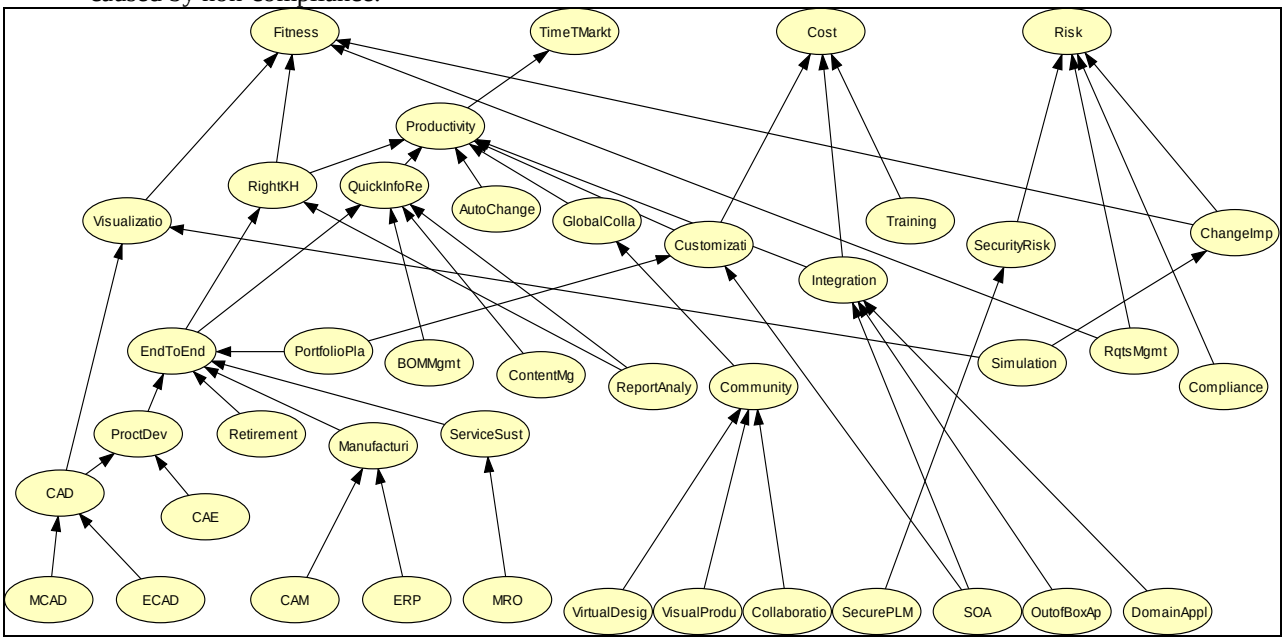
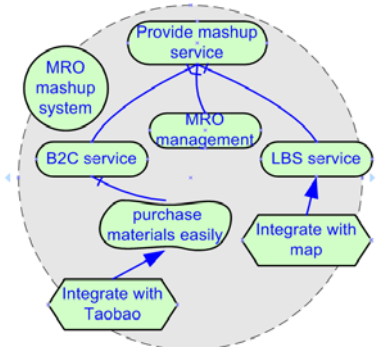


Fig. 7. Goal decomposition for components in the enterprise PLM product family

5.4 Experiment Scenarios on the collaborations and evolutions of Internetware Components

Taking the componentization of MRO System as an example, we adopt the Mashup tool developed within the project team to wrap-up the existing product family into a set of Internetware components, so that the functionality of the current MRO system can be extended at run-time by the collaboration of MRO component with other components accessible within the platform. Assume that at the beginning of developing mashup system, there are 3 high-level goals: r_1, providing MRO management; r_2, Purchase materials on the Internet when needed and providing B2C service; r_3, integrating with BaiduMap component and providing LBS Services. The goal model of the initial requirements is shown in Fig. 12.



r1: MRO management
$r2$: Purchase materials on the Internet
r3: get the exact location of branch company
s1: MRO management service
s2: Taobao service
s3: Baidu map service

Fig. 8. Goal model of MRO mashup system requirements at Phase t_1

$$(R, t_1) = \{r_1, r_2, r_3\} \qquad Active\_queue: \{(r_1, t_1), (r_2, t_1), (r_3, t_1)\} \qquad Inactive\_set: \{\}$$

The initial priorities of these 3 goals are as the following. In the development stage of t_1, component s_1 will satisfy r_1, while component s_2 and s_3 satisfy r_2 and r_3 respectively. Table 10 shows the completion of components and corresponding requirement maturity after first release of MRO Mashup System.

$$P(r_1, t_1) > P(r_2, t_1) = P(r_3, t_1)$$

$$s_1 \rightarrow (r_1, t_1) \qquad s_2 \rightarrow (r_2, t_1) \qquad s_3 \rightarrow (r_3, t_1)$$

Table 10. Completion of component at the end of t1 in Active_queue

	Satisfied components	Components completion	Weight of component in requirement	Priority initial factor u	Requirement maturity
r1	S1	100%	1	1	1
r2	S2	100%	0.3	0.8	0.3
r3	S3	80%	0.4	0.6	0.32

At the end of t_1, component s_1 is available, resulting in the high maturity of its corresponding requirement r_1. Due to the full completion and low priority at this moment, r_1 is picked out from the Active_queue, added into the Inactive_set. At the end of time t_1, the situation of requirements and components is as following:

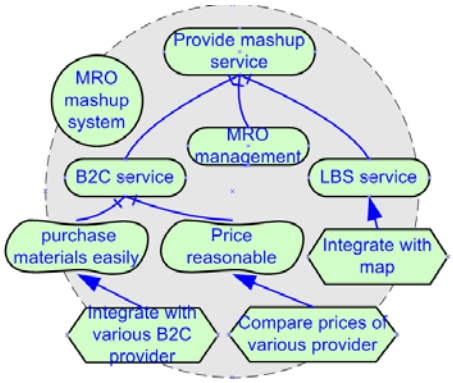
$$M(r_1,t_1) > M(r_3,t_1) > M(r_2,t_1) \qquad P(r_2,t_1) > P(r_3,t_1) > P(r_1,t_1)$$

$$Active\_queue:\{(r_2,t_1),(r_3,t_1)\} \qquad Inactive\_set:\{(r_1,t_1)\}$$

Although Requirement r_1 is in Inactive_set, it will barely become a motive for system evolution ever since. Nevertheless, r_1 is still equipped with the possibility for stimulating the appearance of new requirement. Requirement r_2 and r_3 are still pending for consideration. They both have the potential for further improvement, as well as stimulate new requirements.

Phase t_2:

Fig. 13 shows the goal model of requirements at Time t_2. Notice that a new sub-requirement appears. The new requirement r_4, comparing prices of materials among various B2C providers, is on the basis of component s_2. Treat r_4 as a sub-requirement of requirement r_2. It is obvious that r_2 is stimulated by component s_2, which shows that the requirement pyramid is influenced by the system component pyramid. On the other hand, component s_3 has not been completed at the end of Time t_1. The major target of s_3 in Time t_2 is for further improvement.



$r2$: Purchase materials on the Internet
r3: get the exact location of branch company
s3: Baidu map service
$s4$: Integrate more B2C provider and construct price comparision service

Fig. 9. Goal model of MRO Mashup System requirements in t2

The initial state of MRO Mashup System in the beginning of t_2 is as follows:

$$(R,t_2)=\{r_2,r_3\} \qquad Active\_queue:\{(r_2,t_2),(r_3,t_2)\} \qquad Inactive\_set:\{(r_1,t_1)\}$$

In the development stage of t_2, component s_4 will satisfy the requirement r_2, while component s_3 satisfies requirement r_2. Table 11 shows the completion of components and corresponding of requirement priority after second release of MRO Mashup System.

$$s_3 \rightarrow (r_3,t_2) \qquad \{s_2,s_4\} \rightarrow (r_2,t_2) \qquad M(r_3,t_1) > M(r_2,t_1) \qquad P(r_2,t_1) > P(r_3,t_1)$$

Table 11. Completion of component at the end of t2 in Active_queue

	Satisfied components	Components completion	Weight of component in requirement	Priority initial factor u	Requirement maturity
r1	S1	100%	1	0	1
r2	S2	100%	0.3	1	0.6
	S4	100%	0.3		
r3	S3	100%	0.4	0.5	0.4

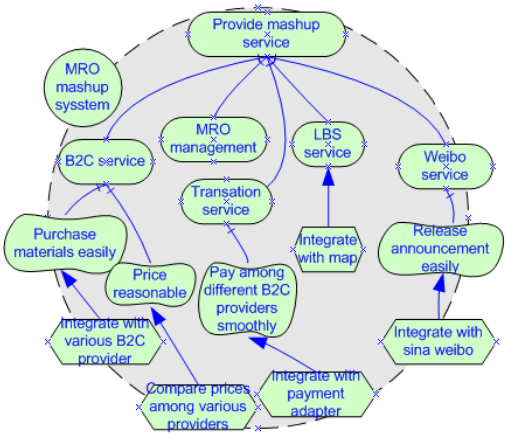
At the end of phase t_2, component s_3 and s_4 are both totally completed, whereas requirement r_2 and r_3 are neither completed yet. r_2 and r_3 both have the potential for further improvement.

$$M(r_2,t_2) > M(r_3,t_2) \quad P(r_2,t_2) > P(r_3,t_2)$$

$$Active\_queue:\{(r_2,t_2),(r_3,t_2)\} \quad Inactive\_set:\{(r_1,t_1)\}$$

Phase t_3:

Fig. 14 shows the goal model of requirements in Time t_3. Notice that in the beginning of t_3, there have been two totally new requirements. One is transaction requirement for smooth payment between different B2C providers, the other one is Weibo component requirement. It is obvious that transaction requirement is stimulated by requirement r_2.



r4: Pay between different B2C providers smoothly

r5: Release announcement on Weibo

s5: Payment service and adapter

$s6$: Payment adapter

$s7$: Integrate sina weibo service

Fig.10. Goal model of MRO Mashup System requirements in t3

The initial state of MRO Mashup System in the beginning of t_3 is as following:

$$(R,t_3)=\{r_4,r_5\} \quad Active\_queue:\{(r_4,t_3),(r_5,t_3),(r_2,t_3),(r_3,t_3)\} \quad Inactive\_set:\{(r_1,t_1)\}$$

In the development stage of t_3, component s_5 will satisfy the requirement r4, while component s_6 satisfies requirement r_5. Neither r_2 nor r_3 is completed totally, they have none improvement tasks in time t_3. As a consequence, r_2 and r_3 are both suspended in t_3 period. Table 24 shows the completion of components and corresponding of requirement priority after second release of MRO Mashup System.

$$s_5 \to (r_4,t_3) \quad s_6 \to (r_5,t_3) \quad M(r_5,t_3)=M(r_6,t_3)=0 \quad P(r_5,t_3) > P(r_6,t_3)$$

Table 12. Completion of component at the end of t3 in Active_queue

	Satisfied components	Components completion	Weight of component in requirement	Priority initial factor u	Requirement maturity
r1	S1	100%	1	0	1
r2	S2	100%	0.3	1	0.6
	S4	100%	0.3		
r3	S3	100%	0.4	0.5	0.4
r4	S5	100%	0.5	1	1
	S6	100%	0.5	0.8	
r5	S7	70%	0.5	0.6	0.35

At the end of time t_3, component s_3 and s_4 are both completed, whereas neither requirement r_2 nor r_3 are satisfied fully yet. r2 and r3 both have the potential for further improvement.

$$1 = M(r_4,t_3) > M(r_5,t_3) > M(r_3,t_2) > M(r_2,t_2)$$

$$Active\_queue:\{(r_2,t_2),(r_5,t_3),(r_3,t_2)\} \quad Inactive\_set:\{(r_1,t_1),(r_4,t_3)\}$$

It is obvious that the higher level requirements always depend on lower level requirements. Based on this, the evolution trend is somehow traceable and it is possible to make requirements prediction and components optimization. Suppose that engineers are developing system components according to the user requirements at certain stage. If engineers can foresee the possibility of some evolving components, and undertake responding enhancement and optimization, the system is more likely to satisfy emerging requirements in the next stage, resulting in better user experience and shortened system development cycle. The system adapts to changes in the environments and evolves with higher efficiency and better quality.

6. Related work

The importance of software adaptation and evolution is widely acknowledged by many researchers. This assumption has been adopted as the basis of N. Chapin et.al. [13] and J. Buckley et.al.[10] 's refined taxonomies for maintenance changes. Also, some researches hold a different opinion. For example, in the stage model proposed by K. H. Bennett et. al. [8], maintenance refers to general post-delivery activities, and evolution refers to a phase that adapts the application to the constantly changing requirements and run-time environment. Nowadays in most of the literature, it is widely accepted that continuous changes are the key feature of evolution. It has been associated with the change of code [26][27] [35], modules[2][62] or architecture [34][1][32] of a software system typically. In addition, there are also studies on reverse engineering [1][32] at the code and process level. At present, the state-of-the-art work focuses on capturing and handling the changes in requirements and the environment [21][56][30].

Internetware, as a new software paradigm, aims to make the process of adaptation and evolution automated by using software assets accessible in the open Internet environment. The most closely related literature is in the services computing domain, where services adaptation and evolution is a topic attracts many interesting research work. Generally speaking, there are two perspectives on service evolution: macro and micro. [44][46] consider the problem from the perspective of a service system community, and apply evolutionary computing to simulate the biological evolution process; the rest majority concentrate on the micro perspective, i.e., the evolution of a composite software service, which is in line with the evolution of Internetware components.

In [48], M. Papazoglou interprets service evolution as the continuous development of a service through a series of changes through the service's different versions. The key challenge is the forward compatibility between different versions, which is further explained as: a guarantee that an older version of a client application should be able to interpret and use newer message/ data formats of the provider. A few work discussed the taxonomy of evolutionary changes [48][60] [50], and there are more research efforts commit themselves to address the compatibility problem using different tactics, including versioning, design pattern/adaptor [31][60][25] and theory/model[3][4]. Two types of changes are introduced: shallow (confined to services or clients) and deep (leading to cascading and side effects) changes. Also, he proposes a contract-based approach and a change-oriented lifecycle methodology to address these issues. As for shallow changes, typically it could leads to two levels of mismatches [45]: (1) interface-level, i.e. structural (2) protocol-level (i.e. ordering mismatches between service interaction protocols). Most of the work currently focuses on WSDL services and their interface, and the evolutionary changes are further discussed on message, operation, service and binding [9]. In addition, change of QoS and the semantic has also been considered to the subcategory of service evolution [60][50][36].

Versioning is a traditional and practical way to address the above challenges. A common trait of them is maintaining multiple service versions for a specified interface. A simple naming scheme that appends a date or version stamp to the end of a namespace is suggested to ensure the uniqueness of the version identifier. At the technical level, they rely heavily on the SOA technology – SOAP, WSDL and UDDI. Usually, it is used together with the design pattern and related tools [36]63][23]. Design pattern is a widely adopted and effective approach in practice. A typical class of approach is to maintain multiple versions of a service on the server side, and provide a proxy that enable dynamic binding and invocation for client applications [36][23]. In a similar spirit, P. Kaminski et. al. [31] propose a Chain of Adapters (multiple service interfaces for one implementation version) and D. Frank etc. [25] suggest a service interface proxy (one service interface for multiple implementation versions) for dealing with the incompatibility. In addition, Z. Le Zou [37] and [47] try to keep client applications synchronized with the evolved service through (semi-) automatic client side update. To address possible interface mismatches, M. Dumas etc. [22] introduces an algebra-based interface adaptation method, in which each interface is represented as an algebra expression that could be transformed and linked accordingly. H. R. Motahari Nezhad et. al. [45] provide semi-automatic support for adapter generation to resolve the mismatches at the interface-level and deadlock-free interaction protocol level.

M. Treiber in [60] proposes a Service Evolution Management Framework (SEMF) that relies on an information model (e.g. usage statistics, logging) to manage the web service changes on Interface, QoS and interaction pattern. V. Andrikopoulos et. al. [45] develops a formal abstract service (service schema) model for service evolution management, which provides an understanding for change tracking, control and impact. In [3], Andrikopoulos develops a theory to identify and reason whether the changes to a WSDL service are permitted based on type and set theory as well as the service specification model. In [27], an algorithm for automatically assessing the forward compatibility between two revisions of a service specification is proposed.

Besides structural evolution, service protocol [45][55], composition schema [51][52][49] and process [55][53] evolution has also been carefully studied. [7][51] are good examples in analyzing and addressing service incompatibility at the interaction protocol level. [51][52][49] focus on service equivalence and substitutability in evolving composition schema. In [55][53], the authors introduce business process evolution, of which the challenge is to dynamically migrate ongoing instances into the new version of a process.

Also, there are some other interesting researches on service evolution. For example, [54] [64] discuss about the two well-understood strategies: "just enough" (ignoring unknown content) and adding schema extension points, which could be helpful for the semantics of evolution. In [61], S. Wang proposes a quantitative impact analysis model based on inter- and intra-service dependency. And [43][12] concentrate on whether the designed system covers the requirement and how human intention drives the evolution of software system using the situ framework.

Service adaptation is also an important methodology to keep the system behave as expected under certain condition. It generally includes the following scenarios: the system may need to dynamically (re)select candidate service when suffer from failure [5][6], (re-) composite services to fulfil changed business process [11][17], replicate and distribute more service instances to adapt to emergent work load [18].

Adaptation has much to do with evolution but also has subtle distinctions. Generally, the change of requirement and/or environment would not always lead to software evolution if the variation is within anticipation at design time. At the moment the system is able to adopt an alternative mechanism in response (adaptation). When the situation is new / unknown to the system, the software system has to evolve to handle this case. More specifically, adaptation is conservative which only takes advantage of existing building units to satisfy the changed requirements/ environment; while evolution is aggressive which may change the system boundary, e.g. adding/removing/modifying a service in a system.

7. Conclusion and Future work

The adaptation and evolution characteristics are two of the most prominent properties of Internetware components, as well as user requirements. Across the software life cycle, the evolution of Internetware system is driven by the need of requirements. In this article, we illustrate our understanding and practice on Internetware adaptation and evolution.

First of all, we propose to model Internetware systems using a four tuple including goals, environments, processes and strategies. The four tuple covers the major properties required to represent the requirements and system behaviours in the process of Internetware adaptation and evolution. In particular, goals and environments are concepts capturing the problem space, processes and strategies are used to capture the solution space. Internetware components are the system entities implementing processes and strategies. Then we defined a typology of system based on the four tuple model, so that the differences between conventional static systems, reactive systems, adaptive systems and compositional/collaborative adaption systems can be explicitly defined. Essentially the difference lies in the management of gradual changes in the goal model. Then we refined the four tuple model into a meta-model for Internetware components and defined the algorithms for matching and composing Internetware components with the proposed requirements and system modelling elements. This further develops into the adaptation process through compositional actions. Examples in the enterprise application domain are used to illustrate the different steps in the components selection, evolution process. An experimental environment on the cloud is deployed by the project team to integrate and showcase the various tools and applications of the Internetware paradigm.

In summary, user requirement and system component are both evolving continuously, in which they constantly influence the improvement of each other stepwise. User requirements have a direct impact on the design and development of system, while the implementation of system components would also stimulate the generation of new user requirements, either directly or indirectly. If we bear in mind a co-evolution model capturing the interaction between user requirement and system development. It can help extend the life cycle of software systems and achieve high quality and efficiency at the same time. This paper proposes a requirements-driven evaluation framework for Internetware-based components, on both their functionality and quality. In particular, we offer an account of how to model these requirements, how to derive from them a space of component functionality alternatives, and how to select among these alternatives on the basis of desired qualities. In essence, the selection of component functionality is framed as a satisfaction problem for functional requirements, while component quality is addressed as an optimisation problem. The research work shows that the Internetware paradigm presents a promising direction to compose software applications from existing software assets on demand. In the future, the proposed approach in this paper can be further enhanced with other automated evaluation measures. Their efficacy in supporting components development and composition can be studied and evaluated. Another possible future line of research is to develop domain specific components selection knowledge base and integrate with widely used software development and deployment platform.

ACKNOWLEDGMENTS

This paper receives financial support from the National Basic Research Program of China (973) (Grant No. 2009CB320706). We thank Professor John Mylopoulos, University of Trento, Italy and Professor Eric Yu, University of Toronto, Canada for sharing their insights during this project, and Fenglin Li, Wenting Ma, Yunsong Jian, Zhanlei Ma, Haihua Xie, Jun Wang, Ziyan Xu, Haihong Zhao, Sheau Ling Tan for their contributions to the project.

REFERENCE

[1]. Alawairdhi M., and H. Yang, "A Business-Logic Based Framework for Evolving Software Systems," in Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International, 2009, vol. 2, pp. 300–305.

[2]. S. Ali and O. Maqbool, "Monitoring software evolution using multiple types of changes," in Emerging Technologies, 2009. ICET 2009. International Conference on, 2009, pp. 410–415.

[3]. Andrikopoulos V., "A Theory and Model for the Evolution of Software Services," Open Access publications from Tilburg University, 2010.

[4]. Andrikopoulos, V. S. Benbernou, and M. Papazoglou, "Managing the evolution of service specifications," in Advanced Information Systems Engineering, 2008, pp. 359–374.

[5]. Ardagna D. and B. Pernici, "Adaptive service composition in flexible processes," Software Engineering, IEEE Transactions on, vol. 33, no. 6, pp. 369–384, 2007.

[6]. Ardagna D., M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani, "Paws: A framework for executing adaptive web-service processes," IEEE SOFTWARE, pp. 39–46, 2007.

[7]. Becker K., A. Lopes, D. S. Milojicic, J. Pruyne, and S. Singhal, "Automatically determining compatibility of evolving services," in Web Services, 2008. ICWS'08. IEEE International Conference on, 2008, pp. 161–168.

[8]. Bennett K. H. and V. T. Rajlich, "Software maintenance and evolution: a roadmap," in Proceedings of the Conference on the Future of Software Engineering, 2000, pp. 73–87.

[9]. Borovskiy V. and A. Zeier, "Evolution Management of Enterprise Web Services," in Advanced Management of Information for Globalized Enterprises, 2008. AMIGE 2008. IEEE Symposium on, 2008, pp. 1–5.

[10]. Buckley J., T. Mens, M. Zenger, A. Rashid, and G. Kniesel, "Towards a taxonomy of software change," Journal of Software Maintenance and Evolution: Research and Practice, vol. 17, no. 5, pp. 309–332, 2005.

[11]. Chafle G., K. Dasgupta, A. Kumar, S. Mittal, and B. Srivastava, "Adaptation inweb service composition and execution," in Web Services, 2006. ICWS'06. International Conference on, 2006, pp. 549–557.

[12]. Chang C. K., H. Jiang, H. Ming, and K. Oyama, "Situ: A situation-theoretic approach to context-aware service evolution," Services Computing, IEEE Transactions on, vol. 2, no. 3, pp. 261–275, 2009.

[13]. Chapin N., J. E. Hale, K. M. Khan, J. F. Ramil, and W. G. Tan, "Types of software evolution and software maintenance," Journal of software maintenance and evolution: Research and Practice, vol. 13, no. 1, pp. 3–30, 2001.

[14]. Chauvel, F., H. Song, X. Chen, G. Huang, H. Mei: Using Quality-Contracts to Drive Architecture-Centric Self-adaptation. QualityA 2010: 102-118.

[15]. Cheng, B. H. C., H. Giese, P. Inverardi, J. Magee, and R. de Lemos. Software Engineering for self-adaptive systems: A research road map. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, eds. Software Engineering for Self-Adaptive Systems, Vol. 08031 of Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.

[16]. Clement, B. J. and E. H. Durfee. Theory for coordinating concurrent hierarchical planning agents using summary information. In Proceedings of the National Conference on Artificial Intelligence (AAAI-99),pages 495-502,July 1999.

[17]. Colombo M., E. Di Nitto, and M. Mauri, "Scene: A service composition execution environment supporting dynamic changes disciplined through rules," Service-Oriented Computing–ICSOC 2006, pp. 191–202, 2006.

[18]. Chohan N., C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski, "AppScale: Scalable and Open AppEngine application development and deployment," Cloud Computing, pp. 57–70, 2010.

[19]. Dalkey N. C. The Delphi method: An experimental study of group opinion, 1969. The Rand Corporation.

[20]. Dardenne, A., A. van Lamsweerde, S. Fickas: Goal-Directed Requirements Acquisition. Sci. Comput. Program. 20(1-2): 3-50 (1993).

[21]. Desai N., A. K. Chopra, and M. P. Singh, "Amoeba: A Methodology for Requirements Modelling and Evolution of Cross-Organizational Business Processes," Transactions on Software Engineering and Methodology (submitted, 2008), 2008.

[22]. Dumas M., M. Spork, and K. Wang, "Adapt or perish: Algebra and visual notation for service interface adaptation," Business Process Management, pp. 65–80, 2006.

[23]. Fang R., L. Lam, L. Fong, D. Frank, C. Vignola, Y. Chen, and N. Du, "A version-aware approach for web service directory," in Web Services, 2007. ICWS 2007. IEEE International Conference on, 2007, pp. 406–413.

[24]. Fickas, S. and M.S. Feather. Requirements monitoring in dynamic environments. In Proceedings of the Second IEEE International Symposium on Requirements Engineering, IEEE Computer Society. p. 140.

[25]. Frank D., L. Lam, L. Fong, R. Fang, and M. Khangaonkar, "Using an interface proxy to host versioned web services," in Services Computing, 2008. SCC'08. IEEE International Conference on, 2008, vol. 2, pp. 325–332.

[26]. Godfrey M. and Q. Tu, "Growth, evolution, and structural change in open source software," in Proceedings of the 4th international workshop on principles of software evolution, 2001, pp. 103–106.

[27]. Herraiz I., J. M. Gonzalez-Barahona, and G. Robles, "Determinism and evolution," in Proceedings of the 2008 international working conference on Mining software repositories, 2008, pp. 1–10.

[28]. Jian, Y., T. Li, L. Liu, E. Yu. Goal-Oriented Requirements Modelling for Running Systems. In Proceedings of 1st

Requirements @ Run-Time Workshop, Sydney, Australia, 2010.
[29]. Jureta, I.J., Stéphane Faulkner, Philippe Thiran. Dynamic Requirements Specification for Adaptable and Open Service-Oriented Systems. In Proceedings of the 5th international conference on Component-Oriented Computing. Vienna, Austria, Springer-Verlag. p. 270-282.
[30]. Kazhamiakin R., M. Pistore, and M. Roveri, "A framework for integrating business processes and business requirements," in Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International, 2004, pp. 9–20.
[31]. Kaminski P., H. Müller, and M. Litoiu, "A design for adaptive web service evolution," in Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems, 2006, pp. 86–92.
[32]. Kim T., K. Kim, and W. Kim, "An Interactive Change Impact Analysis based on an Architectural Reflexion Model Approach," in Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual, 2010, pp. 297–302.
[33]. Lapouchnian A., S. Liaskos, J. Mylopoulos, Y. Yu. Towards Requirements-Driven Autonomic Systems Design. In Proc. ICSE 2005 Workshop on Design and Evolution of Autonomic Application Software (DEAS 2005), St. Louis, Missouri, USA, May 21, 2005. ACM SIGSOFT Software Engineering Notes 30(4), July 2005.
[34]. Le Goaer O., D. Tamzalit, M. Oussalah, and A. D. Seriai, "Evolution Shelf: Reusing Evolution Expertise within Component-Based Software Architectures," in Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International, 2008, pp. 311–318.
[35]. Lehman M., "Laws of software evolution revisited," Software process technology, pp. 108–124, 1996.
[36]. Leitner P., A. Michlmayr, F. Rosenberg, and S. Dustdar, "End-to-end versioning support for web services," in Services Computing, 2008. SCC'08. IEEE International Conference on, 2008, vol. 1, pp. 59–66.
[37]. Le Zou Z., R. Fang, L. Liu, Q. B. Wang, and H. Wang, "On Synchronizing with Web Service Evolution," in Web Services, 2008. ICWS'08. IEEE International Conference on, 2008, pp. 329–336.
[38]. Li, Y., M. Zhou, C. You, G. Yang, H. Mei: Enabling on Demand Deployment of Middleware Components in Componentized Middleware. CBSE 2010: 113-129.
[39]. Lv J, Tao X, Ma X, et al. On agent-based software model for Internetware. Science China Series E-Tech Science (in Chinese), 2005, 35(12): 1233—1253
[40]. Lv, J., Xiaoxing Ma, Xianping Tao, Feng Xu, Hao Hu. Research and Progress in InternetWare. Sciences in China, Series F: Information Sciences. 2006 Vol. 36(10), 1037-1080.
[41]. Ma, W. L. Liu, H. Xie, etc. Preference Model Driven Services Selection. Advanced Information Systems Engineering, 2009. pp.216-230. Springer Berlin.
[42]. Ma, W. L. Liu, W. Feng, etc. Analyzing Project Risks within a Cultural and Organization Setting. 2009 ICSE Workshop on Leadership and Management in Software Architecture, 2009. pp.6-14.
[43]. Ming H., C. K. Chang, K. Oyama, and H. I. Yang, "Reasoning about Human Intention Change for Individualized Runtime Software Service Evolution," in Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual, 2010, pp. 289–296.
[44]. Miorandi D., L. Yamamoto, and P. Dini, "Service evolution in bio-inspired communication systems," Int. Trans. Syst. Sc. and Appl, vol. 2, no. 1, pp. 51–60, 2006.
[45]. Motahari Nezhad H. R., B. Benatallah, A. Martens, F. Curbera, and F. Casati, "Semi-automated adaptation of service interactions," in Proceedings of the 16th international conference on World Wide Web, 2007, pp. 993–1002.
[46]. Nakano T. and T. Suda, "Self-organizing network services with evolutionary adaptation," Neural Networks, IEEE Transactions on, vol. 16, no. 5, pp. 1269–1278, 2005.
[47]. Ouederni M., G. Salaün, and E. Pimentel, "Client Update: A Solution for Service Evolution," in Services Computing (SCC), 2011 IEEE International Conference on, 2011, pp. 394–401.
[48]. Papazoglou M., "The challenges of service evolution," in Advanced Information Systems Engineering, 2008, pp. 1–15.
[49]. Pathak J., S. Basu, and V. Honavar, "On context-specific substitutability of web services," in Web Services, 2007. ICWS 2007. IEEE International Conference on, 2007, pp. 192–199.
[50]. Ponnekanti S. and A. Fox, "Interoperability among independently evolving web services," Middleware 2004, pp. 331–351, 2004.
[51]. Rinderle-Ma S., M. Reichert, and M. Jurisch, "On Utilizing Web Service Equivalence for Supporting the Composition Life Cycle," International Journal of Web Services Research (IJWSR), vol. 8, no. 1, pp. 41–67, 2011.
[52]. Rinderle-Ma S., M. Reichert, and M. Jurisch, "Equivalence of web services in process-aware service compositions," in Web Services, 2009. ICWS 2009. IEEE International Conference on, 2009, pp. 501–508.
[53]. Rinderle S., B. Weber, M. Reichert, and W. Wild, "Integrating process learning and process evolution–a semantics based approach," Business Process Management, pp. 252–267, 2005.
[54]. Robinson I. "Consumer-Driven Contracts: A Service Evolution Pattern." 12 June 2006. [Online]. Available: http://martin fowler.com/articles/consumerDrivenContracts.html. [Accessed: 21-Feb-2012].
[55]. Ryu S. H., F. Casati, H. Skogsrud, B. Benatallah, and R. Saint-Paul, "Supporting the dynamic evolution of web service protocols in service-oriented architectures," ACM Transactions on the Web (TWEB), vol. 2, no. 2, p. 13,

2008.
[56]. Salifu M., Y. Yu, and B. Nuseibeh, “Specifying monitoring and switching problems in context,” in Requirements Engineering Conference, 2007. RE’07. 15th IEEE International, 2007, pp. 211–220.
[57]. Saaty T. L. Decision making with the analytic hierarchy process. Int. J. Components Sciences, Vol. 1, No. 1, 2008, pp. 83-98.
[58]. Tang J. and Z. Jin, Assignment Problem in Requirements Driven Agent Collaboration and its Implementation, Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2010): 839-846, May 10-14, 2010.
[59]. Tang J. and Z. Jin, Requirement Driven Agent Collaboration and QoS based Negotiation, Proceedings of the 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT2009: 585-588, Milan, Italy, 15-18 September 2009
[60]. Treiber M., H. L. Truong, and S. Dustdar, “On analyzing evolutionary changes of web services,” in Service-Oriented Computing–ICSOC 2008 Workshops, 2009, pp. 284–297.
[61]. Wang S. and M. A. Capretz, “A dependency impact analysis model for web services evolution,” in Web Services, 2009. ICWS 2009. IEEE International Conference on, 2009, pp. 359–365.
[62]. Wang Y., D. Guo, and H. Shi, “Measuring the evolution of open source software systems with their communities,” ACM SIGSOFT Software Engineering Notes, vol. 32, no. 6, p. 7, 2007.
[63]. Weinreich R., T. Ziebermayr, and D. Draheim, “A versioning model for enterprise services,” in Advanced Information Networking and Applications Workshops, 2007, AINAW’07. 21st International Conference on, 2007, vol. 2, pp. 570–575.
[64]. Wilde E., “Semantically extensible schemas for Web Service evolution,” Web Services, pp. 30–45, 2004.
[65]. Yu, E., “Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering”. Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97) Jan. 6-8, 1997, Washington D.C., USA. 226-235.
[66]. Zheng L., J. Tang, Z. Jin. An Agent Based Framework for Internetware Computing. International Journal of Software and Informatics, 2010,4(4):401~418.