Modeling Internet-Based Software Systems Using Autonomous Components

Wenpin Jiao, Pingping Zhu, Hong Mei

Institute of Software, School of Electronics Engineering and Computer Science
Peking University, Beijing 100871, China
{jwp, zhupp}@cs.pku.edu.cn, meih@pku.edu.cn

Abstract. Internet-based software systems (called as Internetware) are dynamically formed task-specific coalitions of distributed autonomous components. Autonomous components were modeled from five aspects, i.e., goal, service, use contract, operating context, and implementation. The model can semantically reason about the autonomous behaviors of autonomous components. Based on the model, Internetware can be constructed using autonomous components from two directions, i.e., goal-driven refinement from top to bottom and cooperation-based composition from bottom to up. Then, Internetware is feasible if the refinement process can find a group of autonomous components that are willing to cooperate and can cooperatively achieve the goals of Internetware.

Keywords: Internet-based Software System, Internetware, Autonomous Component, Model

1 Introduction

The widespread used Internet is forming a new computing environment, which is becoming increasingly open and dynamic and differs from traditional environments [4] [12] on many aspects. For example, there is no clear boundaries for software systems or central authority for control or validation; resources (*e.g.*, information, calculation, communication, control, or services) are usually autonomous and heterogeneous.

Correspondingly, software systems on the Internet can be viewed as dynamically formed task-specific coalitions of distributed autonomous resources. Internet-based software systems (we call them as *Internetware* in this work) have many new characteristics:

- Autonomy. Components constructing Internetware may be independent, active and adaptive entities.
- Cooperativity. Internetware can be considered as a federation of Internet-based components. To perform the tasks of Internetware, components should cooperate in a coalition.
- Evolutivity and adaptability. Components selected to assemble Internetware could be varied occasionally due to the dynamics of the Internet environment, for instance, components dynamically enter or leave the Internet or alter their service content.
- Environment-awareness and dependency. Internetware situated in the open and dynamic environment should be able to perceive and adapt to the changes of the environment so that it could evolve properly.

In this work, we introduce the concept of autonomous component for modeling autonomous computing resources on the Internet. Autonomous components are modeled as a tuple of goals, services, use contract, operating context and implementation. Based on the autonomous component model, we present an approach based on goal-driven refinement and cooperation-based composition to constructing Internetware.

In the following context, Section 2 describes the autonomous component model and its semantics. Section 3 presents the approach to constructing Internetware. Section 4 discusses related work and makes some conclusion remarks on our work.

2 Autonomous Component

General speaking, components are software entities that can be deployed independently and are subject to composition by third parts [14]. A software component is specified via its interface and the contract that define how other components can directly invoke the services *provided* by the component [2]. Existing models for software components are usually based on provided and required services (*e.g.*, [5] [7] [15]) and they can just successfully manage functional interfaces and can only be used to specify passive components.

However, computing resources on the Internet differ from traditional components on many aspects:

- They are independently developed and delivered and are distributed over the whole network, and their executions are supported and controlled locally by the network nodes where they locate. Thus, they may not be free again, they can autonomously decide whether or not to provide services, and they may change their behaviors or performance without notification.
- Even though they can be assembled into software systems for performing specific tasks, many of them can also execute alone for achieving their own goals. They are no longer passive entities developed only for assembly so they may or may not respond to service requests when they are busy with their own affairs.
- They may provide services by themselves but they may also request services while implementing their functionalities. Due to the dynamics of the Internet, they may provide the same services in different ways; and they may request the same service dynamically from different components on the other hand.

In addition, Internet-based computing resources are greatly diversiform in terms of the type of service, the operating context, the quality of service, the interoperability, the usage and so on. All of these indicate that existing component models are not sufficient for specifying Internet-based computing resources.

2.1 Autonomous Component Model

Autonomous components are computing resources (or software entities) distributed over the Internet, which are developed independently to provide services for outside use besides pursuing local goals. Autonomous components interconnect via the Internet and interact by using varied communication technologies supported on the Internet, for instance, RPC (remote procedure call) and HTTP. To announce the services that an autonomous component can provide, those who deliver the component should explicitly specify how the component can provide services successfully and how people can use the services, including how to locate or find the component and how to assemble or interconnect the component.

The specification of an autonomous component refers to the following aspects of contents:

- Goals. Autonomous components are active and even self-interested and that they take actions (*e.g.*, providing services) is because they want to gain benefits, *i.e.*, to achieve their goals.
- Services. Since autonomous components are often developed for serving the outside world and their local goals are generally user-invisible, autonomous components should declare what services they can provide for the outside use.
- Use contract. When autonomous components announce the services they can provides, they should also notify the ways to use the services so that users could obtain the services.
- Operating context (or running environment). Autonomous components are situated in different environments and they would not be able to achieve their goals or provide services if their environment could not supply sufficient resources and technology supports.
- Implementation. Although the interfaces for providing services and the implementations of autonomous components are generally separable, we should be clear about how autonomous components achieve their goals and implement their services.

2.1.1 Goal

A goal of an autonomous component represents an intention (or objective) to gain benefits or utilities. In most cases, there must be some reason (or stimulus) for an autonomous component to raise an intention. In addition, when several goals are stimulated simultaneously, an autonomous component should be able to schedule its actions for achieving goals by ranking the priorities of goals.

A goal can be specified as a triple $G = < Pre, Eff, Pri >$ where

- *Pre* is the *premise* of pursuing the goal. *Pre* can be expressed as a function of the environment to indicate that the environment evolves into a specific state or some special stimulus (*e.g.*, a request for services) appears in the environment.
- *Eff* is the *effectiveness* of the goal, which can also be expressed as a function of the environment to indicate what state the environment will evolve into after the goal is achieved. For a self-interested autonomous component, *Eff* may also imply how many benefits the component gains.
- *Pri* is the *priority* of the goal. When the priorities are not specified, goals can be achieved concurrently.

2.1.2 Service

General speaking, a service is a functionality provided for outside use. A service can be described as a pair $S = < SID, F >$ where

- *SID* is the *identifier* of the service to be referenced by users.

- *F* is the *functionality* of the service. The functionality can be specified as a pair < *Pre*, *Post* >, where *Pre*/*Post* are the pre-condition and the post-condition, respectively.

2.1.3 Use Contract

Internet-based services may not be free or public to everyone so users should first be authenticated and pay for the services when they are trying to acquire services. In addition, autonomous component distributed over the Internet are heterogeneous, the ways to request services may be varied.

For every service provided by a component, there is a contract specified. A contract is a quintuple *C* = < *SID*, *Auth*, *Payment*, *Arg*, *Req* > where

- *SID* is the *identifier* of the service.
- *Auth* is the *authentication* information that users should provide while they are going to request the service. When the service is public to everyone, *Auth* can be blank.
- *Payment* in the software world can be considered as *benefits* (or *resources*) that users bring to the autonomous component. Only when the autonomous component obtains the required benefits can it start to provide services. When the service is free, *Payment* can be empty.
- *Arg* is the set of *arguments*. Arguments can be either input or output data items or both.
- *Req* is the way to *request* the service. The way to request the service should be conformed to the architectural style [11] that the autonomous component is supposed to support.

2.1.4 Environment (or Operating Context)

The environment in which an autonomous component situated consists of resources related with the achievements of the component's goals, other autonomous components cooperating with the component to achieve the component's goals, information (*e.g.*, requests for services) transmitted among components, and constraints affecting the behaviors of components.

An environment of autonomous component *P* can be specified as a quadruple *E* = < *Rsrc*, *Com*, *Info*, *Cons* > where

- *Rsrc* is the set of *resources*. A kind of resources can be abstracted as an object class and a concrete resource can be considered as an instance object of the class.
- *Com* is the set of *autonomous components* involved in the environment.
- *Info* is the *information* transmitted through the environment. In the real world, information flowing in the environment could be varied. Nevertheless, in the software world, the information we care most is the requests for services and replies to the requests transferred among components.
  - *Info* = *REQ* ∪ *REP*, where *REQ* are requests for services conforming to specific use contracts and *REP* are replies corresponding to requests.
- *Cons* specifies the *constraints* to autonomous components involved in the environment. The constraints of the environment can be specified into two categories, *i.e.*, *interconnection* constraints and *behavior* constraints. Firstly, the environment is unlikely to support all kinds of communication and interoperation technologies so services provided by autonomous components have to be requested and provided in a way that the environment supports. Secondly, because of the limitation of resources and interactions among components, the behaviors of components involved in an environment must be affected by the environment, especially the state transitions of the environment. The behavioral constraints of the environment can be divided into three sub-categories further, *i.e.*, *prohibitions*, *permissions*, and *obligations* [8] [9].
  - *Cons* = < *Interconn*, *Beh* > where *Interconn* is the *interoperation technology* the environment supports and *Beh* is the set of *behavior constraints* the environment imposes on autonomous components. An autonomous component perceives requests for services according to *Interconn* and takes actions by referencing to *Beh*.
  - *Beh* = *Proh* ∪ *Perm* ∪ *Obl*, where *Proh*, *Perm*, and *Obl* are the sets of *prohibitions*, *permissions* and *obligations*, respectively and they have the similar definition with different semantics.
  - *Proh* (*Perm* or *Obl*) = < *Condi*, *Action* > where *Condi* is a function of the environment representing the state of the environment and *Action* is an action (*e.g.*, providing a service) of a component.

2.1.5 Implementation

Traditionally, the interface (*i.e.*, the use contracts) of a component is independent of the component's implementation and users need not care how the component is implemented. However, from the perspectives of developers, both the use contracts and the implementation of a component should be modeled. Nevertheless, while modeling the implementation of an autonomous component,

we do not care how the component is coded, either. Instead, what we are concerned with more is how a component depends on its environment, especially resources and other components involved in the environment, to achieve its goals or provide its services.

To achieve goals or provide services, autonomous components should possess sufficient resources and skills, including the capabilities (*i.e.*, *provided services*) owned by components themselves and the abilities obtained via asking for assistances (*i.e.*, *required services*) from other components.

With respect to every single goal G or service S, there is an implementation.

- $Imp(G)$ or $Imp(S) = < Rsrc, Sk >$ where $Rsrc$ is the set of *resources* appearing in the environment and Sk is the set of *skills* (*i.e.*, *provided* and *requested services*) required by the component.

2.1.6 Autonomous Component

From the point of views of different roles who deliver or use autonomous components, the model for describing autonomous components may be varied.

For venders who deliver autonomous components, they should make clear how to develop, use and run autonomous components. From venders' perspective, the autonomous component model can be defined as a sextuple.

- $M_{vender} = < ID, \mathcal{G}, \mathcal{S}, \mathcal{C}, \mathcal{E}, \mathcal{I} >$ where ID is the identifier of the component, $\mathcal{G}$ is the set of goals of the component, $\mathcal{S}$ is the set of services provided by the component, $\mathcal{C}$ is the set of use contracts corresponding to the provided services, $\mathcal{E}$ is the environment and $\mathcal{I}$ is the implementation of the component.

Differently from the venders of autonomous components, users usually only care what services autonomous components provide and how to use those services. From users' perspective, the autonomous component model can be defined as a triple.

- $M_{user} = < ID, \mathcal{S}, \mathcal{C} >$ where ID, $\mathcal{S}$ and $\mathcal{C}$ are with the same meanings as above.

2.2 Semantics of Autonomous Components

When an autonomous component is modeled, we should be able to reason about the behaviors and properties of the component according to its model.

2.2.1 Activating Goal

Autonomous components situated in the environment can perceive changes happening in the environment and react to the changes. When autonomous components perceive some stimuli and figure out that the occasions (*i.e.*, the premises) for achieving goals appear autonomous components can start pursuing their goals.

$$G \in \mathcal{G} \wedge G.Pre \wedge G.Pri \geq \varepsilon \models CanActivate(G) \quad (1)$$

I.e., a goal of an autonomous component can be activated if the premise of achieving the goal is satisfied and the priority of achieving the goal is high enough, for instance, higher than a threshold ε, where ε may be adjusted dynamically.

2.2.2 Responding Request for Service

Since autonomous components are goal-driven, they provide services because doing so can help them to achieve their own goals.

When a user's request for service, *Req* for S, appears in the environment, an autonomous component will respond the request if the request stimulates the component to pursue some goal and the achievement of the goal engages providing the requested service. Naturally, the user requesting service should pass the authentication and his payment is acceptable.

$$\begin{aligned} &Req \in \mathcal{E}.Info \wedge Req = S.Req \wedge S.F.Pre \wedge \\ &Authenticated(User) \wedge PaymentOf(User) \supseteq UseContractOf(S).Payment \wedge \\ &\exists_{G \in \mathcal{G}}(CanActivate(G) \wedge S \in Imp(G).Sk) \models WillRespond(Req, S) \end{aligned} \quad (2)$$

I.e., an autonomous component will respond a request for a service if the component can perceive the request and providing the service can benefit the achievement of its goal.

2.2.3 Feasibility of Goal

After a goal is stimulated, the goal is achievable if the environment evolves into an appropriate state, *i.e.*, there are sufficient resources and skills for achieving the goal and there are no constraints obstructing the behaviors of the component.

$$\begin{aligned} &Imp(G).Rsrc \subseteq \mathcal{E}.Rsrc \wedge Imp(G).Sk \subseteq (\mathcal{S} \cup \bigcup_{\forall c \in \mathcal{E}.Com} c.\mathcal{S}) \wedge \\ &\neg\exists_{p \in \mathcal{E}.Cons.Beh.Proh}(p.Condi \wedge \exists_{s \in Imp(G).Sk}(s = p.Action)) \models IsFeasible(G) \end{aligned} \quad (3)$$

2.2.4 Availability of Service

Similarly, a service is available if the environment evolves into a state that supplies adequate resources and skills for performing the service.

$$Imp(S).Rsrc \subseteq \mathcal{E}Rsrc \wedge Imp(S).Sk \subseteq (\mathcal{S} \cup \bigcup_{\forall c \in \mathcal{E}Com} c.\mathcal{S}) \wedge \neg\exists_{p \in \mathcal{E}.Cons.Beh.Proh}(p.Condi \wedge \exists_{s \in Imp(S).Sk}(s = p.Action)) \models IsAvailable(S) \quad (4)$$

2.2.5 Behavior of Autonomous Component

The behaviors of an autonomous component are related to pursuing goals and providing services.

$$CanActivate(G) \wedge IsFeasilbe(G) \models Achieving(G)$$
$$WillRespond(Req, S) \wedge IsAvailable(S) \models Providing(S) \quad (5)$$

I.e., when a goal is feasible and activated, the component will start achieving the goal; and similarly when a service is available and is reasonable to respond a request for the service, the component will provide the service.

After a goal is achieved or a service is provided, the environment will be effected.

$$G.Eff \models Achieved(G)$$
$$S.F.Post \models Provided(S) \quad (6)$$

Proposition 1. If the achievement of a goal depends on providing some service, then the goal is feasible only if the service is available, and the goal is achieved only if the service has been performed.

$$(IsFeasible(G) \wedge S \in Imp(G).Sk) \Rightarrow IsAvailable(S)$$
$$(Achieved(G) \wedge S \in Imp(G).Sk) \Rightarrow Provided(S) \quad (7)$$

2.3 Composition and Refinement of Autonomous Components

Although autonomous components are modeled from five aspects, the composition of two autonomous components can be considered as the mergence of their goals, services and environments since their use contracts and implementations completely depend on their services and environments.

Suppose that $P = < ID_P, \mathcal{G}_P, \mathcal{S}_P, \mathcal{C}_P, \mathcal{E}_P, \mathcal{I}_P >$, $Q = < ID_Q, \mathcal{G}_Q, \mathcal{S}_Q, \mathcal{C}_Q, \mathcal{E}_Q, \mathcal{I}_Q >$ and R is the composition of P and Q. While composing P and Q, we should take the following situations into consideration.

- P and Q are completely independent of each other, *i.e.*, P (or Q) neither requests nor provides services from/for Q (or P) and they are situated in two isolated environments.

  $$R = < ID_R, \mathcal{G}_P \cup \mathcal{G}_Q, \mathcal{S}_P \cup \mathcal{S}_Q, \mathcal{C}_P \cup \mathcal{C}_Q, \mathcal{E}_P + \mathcal{E}_Q, \mathcal{I}_P + \mathcal{I}_Q > \quad (8)$$

  Here, $\mathcal{E}_P + \mathcal{E}_Q$ means simply putting all resources and components involved in the environments together whilst $\mathcal{I}_P + \mathcal{I}_Q$ means collecting all implementations of goals and services into a single collection.
- P and Q are situated in the same environment and share resources, but they do not depend on one another in terms of services.

  $$R = < ID_R, \mathcal{G}_P \cup \mathcal{G}_Q, \mathcal{S}_P \cup \mathcal{S}_Q, \mathcal{C}_P \cup \mathcal{C}_Q, \mathcal{E}_P \oplus \mathcal{E}_Q, \mathcal{I}_P + \mathcal{I}_Q > \quad (9)$$

  Here, $\mathcal{E}_P \oplus \mathcal{E}_Q$ is similar as $\mathcal{E}_P + \mathcal{E}_Q$ except that the shared resources will be re-counted.
- P may request services from Q, *i.e.*, Q is a part of P's environment, on which P will rely to implement its functionality.

  $$R = < ID_R, \mathcal{G}_P \cup \mathcal{G}_Q, \mathcal{S}_P \cup \mathcal{S}_Q, \mathcal{C}_P \cup \mathcal{C}_Q, (\mathcal{E}_P \oplus \mathcal{E}_Q) \backslash Q, \mathcal{I}_P + \mathcal{I}_Q > \quad (10)$$

  Here, $(\mathcal{E}_P \oplus \mathcal{E}_Q) \backslash Q$ means excluding Q from the environment.

Inversely, the refinement is referred to how an autonomous component is decomposed into one or more concretely implemented components that can realize the functionality of the refined component. Differently from composition, while discussing the refinement of an autonomous component, we need not be concerned with the environment and a component can be considered successfully refined if the concrete components can achieve the component's goals and provide the component's services using the same use contracts.

An autonomous component, R, is refined by another component P, which can be a individual component or a composition of a group of components, if P achieves R's goals, implement R's services and preserve the use contracts, *i.e.*,

$$\forall_{G \in \mathcal{G}_R} \exists_{G' \in \mathcal{G}_P}(Achieved(G') \Rightarrow Achieved(G))$$
$$\forall_{S \in \mathcal{S}_R} \exists_{S' \in \mathcal{S}_P}(Provided(S') \Rightarrow Provided(S))$$
$$\mathcal{C}_R \subseteq \mathcal{C}_P \quad (11)$$

The composition and refinement of autonomous components provide a foundation of developing Internet-based software systems.

3 Internet-Based Software Systems – Internetware

Traditionally, when we refer to a software component, we always assume that the component's interface contract and operating context are explicitly specified. Furthermore, according to the specification of a component, we can clearly know how to compose the component into software systems and can predict what outcome we will obtain after the component is deployed and executes.

However, on the Internet, internet-based computing resources, *i.e.*, autonomous components are autonomous and active, and we cannot be sure and guaranteed that an autonomous component will always be committed to the assembly of software systems. When we are requesting services from autonomous components, the outcomes will be uncertainty. In addition, for a service, there often exist multiple autonomous components that can provide it and the ways providing it may be varied. While constructing Internetware systems, we may even not know what autonomous components are actually assembled into systems.

We cannot construct Internetware systems simply via assembling autonomous components through requesting and providing services of components as people did before when assembling traditional components. Therefore, we put forward an approach to constructing Internetware from two complementary directions in this paper.

First, from the perspective of software as service, the goals of Internetware are to provide services so we can model Internetware as a pair.

- $IW = < \mathcal{R}, \mathcal{D} >$ where $\mathcal{R}$ is the requirement of Internetware and $\mathcal{D}$ is the design for implementing Internetware.
  - $\mathcal{R} = \{G \mid G$ is a goal pursued by Internetware$\}$. A goal is in fact to provide a service, *i.e.*, $Imp(G).Sk = \{S\}$, and goal G is achieved if and only if service S is provided successfully.
    $$Achieved(G) \Leftrightarrow Provided(S) \quad (12)$$
    That is, the requirement of Internetware is to achieve a set of goals of providing services.
  - $\mathcal{D} = \{c \mid c$ is an autonomous component participating in Internetware for realizing the requirement.$\}$

Second, from top to bottom, taking the goals as the center of our concerns, we refine Internetware into a group of participant components and Internetware is considered implemented if its goals are achieved and its use contracts are preserved.

Third, from bottom to up, around the design of Internetware, we compose a collection of existing cooperative autonomous components into Internetware in order to achieve the goals cooperatively.

3.1 Goal-Driven Refinement

According to the definition of refinement of autonomous components described above, for every goal of Internetware, there should be an autonomous component that can provide the service specified by the goal and implement the use contract of the service. Furthermore, if the autonomous component requests services from other components while implementing its functionality, then providing the required services can be considered as sub-goals of Internetware.

Suppose $\mathcal{R} = \{G_1, G_2, \ldots, G_n\}$ is the set of goals of Internetware, and for each G_i, $imp(G_i).Sk = \{S_i\}$. The refinement of Internetware can be described recursively as follows, where $\mathcal{P}$ is the set of autonomous components obtained in the refinement process.

1) For every goal G_i ($1 \le i \le n$) in $\mathcal{R}$, search (or implement) an autonomous component P_i that provides S_i (*i.e.*, $S_i \in \mathcal{S}_{Pi}$) and implements the use contract corresponding to S_i, and then let $\mathcal{P} = \mathcal{P} \cup \{P_i\}$.
2) If P will require other services while providing S, *i.e.*, $Imp(S_i).Sk = \{S_{i1}, \ldots, S_{im}\}$, then consider providing $S_{i1}, \ldots, S_{im}$ as sub-goals of Internetware, *i.e.*, let $Imp(G_{Sij}).Sk = \{S_{ij}\}$ ($1 \le j \le m$) , and let $\mathcal{R} = \mathcal{R} \cup \{G_{Si1}, \ldots, G_{Sim}\}$.
3) Stop searching until all goals occurring in $\mathcal{R}$ can be achieved by autonomous components.

Theorem 1. By using the above refinement process, we can obtain a refinement of Internetware, *i.e.*, $\mathcal{P}$ is a refinement of Internetware.

According to the first step, for every $G_i \in \mathcal{R}$, there is a $P_i \in \mathcal{P}$, a goal $G' \in \mathcal{G}_{Pi}$ and $S_i \in Imp(G').Sk$.

Since $Achieved(G') \Rightarrow Provided(S_i)$ (formula 7) and $Provided(S_i) \Leftrightarrow Achieved(G_i)$ (formula 12), we can obtain $Achieved(G') \Rightarrow Achieved(G_i)$, *i.e.*, $\mathcal{P}$ will achieve all goals of Internetware.

According to the second step, because all required services for implementing S_i will be considered as sub-goals to be refined further and these sub-goals will be achieved recursively as well, S_i will be available (formula 4). That is, all services for achieving the goals of Internetware can be provided by $\mathcal{P}$.

Furthermore, according to the definition of refinement (formula 11), we can conclude that $\mathcal{P}$ is a

refinement of Internetware.

3.2 Cooperation-Based Composition

General speaking, there are two types of cooperation [3] among autonomous components involved in Internetware.

- Cooperation with interaction. In this case, autonomous components support mutually in terms of capabilities (*e.g.*, one provides services for another) and they are usually involved in the achievement of a single goal of Internetware.
- Cooperation without interaction, *i.e.*, autonomous components independently take actions of pursuing multiple (sub-)goals of Internetware.

In the discussion of composition of autonomous components, we listed three situations under which the compositions are different. Correspondingly, the first type of cooperation will happen in the third situation whilst the second will take place in the first two situations.

According to the refinement process discussed above, the refinement of Internetware just shows a possibility of achieving the global goals of Internetware under the hypothesis that all services provided by autonomous components are always available and all selected autonomous components will respond requests for services. However, the refinement does not consider whether autonomous components obtained in the refinement process are willing to provide required services.

Differently, when autonomous components intend to cooperate, they must have believed that their actions of providing services would benefit themselves, *i.e.*, being able to achieve their local goals. That is to say, while autonomous components are participating in cooperation, they must have been willing to respond requests for services.

Therefore, we say that Internetware is *feasible* if there is a refinement of Internetware and all autonomous components occurring in the refinement will cooperate.

Theorem 2. If there exists a refinement $\mathcal{P}$ and a composition $\mathcal{D}$ and $\mathcal{P} = \mathcal{D}$, then Internetware is feasible, *i.e.*, for each goal $G \in \mathcal{R}$, G is achievable.

Suppose $Imp(G).Sk = \{S\}$. According to the definition of refinement, S must be one of the provided services of the refinement $\mathcal{P}$, *i.e.*, $S \in \mathcal{S}_P$. Furthermore, according to the refinement process (theorem 1), S will be implemented by autonomous components in $\mathcal{P}$, *i.e.*, *IsAvailable*(S).

On the other hand, since $\mathcal{P} = \mathcal{D}$ and autonomous components in $\mathcal{D}$ are cooperative, $S \in \mathcal{S}_D$ and *WillRespond*($S.Req$, S).

Based on formula 5, the composition $\mathcal{D}$ will provide service S for achieving goal G.

3.3 Refining Internetware into Cooperative Components

As discussed above, the goal-driven refinement and the cooperation-based composition just provide a theoretical possibility of implementing Internetware. On the Internet, due to the autonomy of components and the dynamics of environments, autonomous components may no longer be willing to provide services or they are temporarily (or permanently) inaccessible. The feasibility of Internetware will be affected because the required services are not available again.

When we are discussing the cooperation-based composition of Internetware, we consider two aspects of factors that affect the composition: 1) participating autonomous components are desired to cooperate and 2) they can find adequate capability supports from their environments to implement their functionalities. Obviously, when the environments change, some autonomous components previously involved in the composition may not be willing to cooperate or cannot obtain adequate support from the environments. Thus, the composition for implementing Internetware may be different correspondingly to different environments.

On the contrary, in the refinement process, we ignored the autonomy of components and the dynamics of environments so the obtained refinement of Internetware lacks of dynamic. Thus, no matter how the environments change, we may always obtain the same refinement if those previously obtained autonomous components do not leave the environments.

That is to say, in the dynamic environments involving autonomous components, we cannot ensure that there exists a refinement $\mathcal{P}$ and a composition $\mathcal{D}$ and $\mathcal{P} = \mathcal{D}$ is always held for Internetware, *i.e.*, we cannot always guarantee the feasibility of Internetware.

Nevertheless, if we check the cooperativity of autonomous components obtained in the refinement process simultaneously when we are refining Internetware, we can more possibly get a feasible Internetware.

Therefore, we improve the above refinement process. In the improved refinement, the procedure of searching autonomous components is like an auction using the contract-net protocol [10] [13], in which autonomous components that are requiring services announce the required services and

autonomous components that are willing to cooperate will bid on the tasks of providing services.

- For every goal G_i ($1 \leq i \leq m$) in $\mathcal{K}$, since $Imp(G_i).Sk = \{S_i\}$, there is only one task (*i.e.*, providing the service S_i) to be announced. After the task is announced,
  - Recruit autonomous components that are willing to cooperate, and then
  - Select the autonomous component that requires least payment for carrying on the task of providing the service.

4 Related Work and Conclusions

The Internet forces the environment of software systems to be open and dynamic, which makes it significant to develop flexible, trustworthy, and collaborative software systems to use and share information resources scattered on the open and dynamic Internet. However, traditional component-based software development methodologies failed to meet these challenges due to the following problems.

First, they generally assume that the system goals are determined before the development and the system structures are fixed after the development. They do not take the openness and dynamics of the Internet into consideration and cannot meet the requirements for developing adaptive, evolutional, and collaborative software systems on the Internet.

Second, they always consider the components involved in systems are static and passive, which makes them difficult to develop software systems that are required to be autonomous, pro-active and re-active to the changing demands of the environment and the users.

Third, they usually require that the ways of interconnections and interactions among components are uniform and cannot provide different collaboration supports for components, which makes the components tightly coupled and the structures and behaviors of software systems stiff.

Differently, Internetware is autonomous, cooperative and adaptive and it provides a new solution for developing flexible, multi-purpose, and evolving Internet-based software systems.

To implement Internetware, we should make breakthroughs on the aspects of software theory, methodology, technology and platform of Internetware to enable software systems to be applied to multiple purposes, to integrate autonomous and cooperative software entities, to support flexible interconnections, and to adapt to changes of the dynamic environment.

To study the characteristics of Internetware and develop a feasible way to integrate autonomous software components into Internetware, one of the most important issues that we should first address is to specify autonomous components in a uniform formal model.

So far as we had known by now, there is no published work on modeling autonomous components like us. Even if some literature states it creates a model for autonomous components, autonomous components are assumed unable to self-determine whether, when, or how to provide their services.

For example, [1] presents a component model for encapsulating services, which allows for the adjustment of structure and behavior of autonomous components to changing or previously unknown contexts in which they need to operate. [6] introduces service-oriented concepts into a component model and execution environment and defines a model for dynamically available components, where components provide and require services (*i.e.*, functionality) and all component interaction occurs via services.

In addition, there is much work on modeling traditional components. For example, [15] proposes a unifying component description language for integrated descriptions of structure and behavior of software components and component-based software systems. [7] describes how components are specified at the interface level, design level and how they are composed. A component consists a set of interfaces, provided to or required from its environment and an executable code that can be coupled with other components via its interfaces. [5] introduces a framework dealing with interface signature, interface constraints, interface packaging and configurations, and non-functional properties of software components.

However, those models pay little attention on the autonomy of components or the impacts of the environments on components' autonomous behaviors. Contrarily, most of those models assume that components are not completely self-controlled and will always respond requests once when they perceive requests for services.

In this paper, we introduce the notion of autonomous component and put forward a new definition for it. First of all, autonomous components are software entities for use in the assembly of Internetware. Furthermore, autonomous components are self-controlled, environment dependent, and going in for goals locally.

Based on the definition, autonomous components are modeled from five aspects, *i.e.*, goals they are committed to achieving, services they promise to provide in order to achieve their goals, use contracts

how users use them, environment on which they depend, and implementation how they use environmental resources and skills to achieve their goals. In the model, we give out a very concrete environment description. The environment of an autonomous component consists of not only the resources and skills on which the autonomous component depends to implement its goals and services but also constraints that will affect the behaviors and relationships of autonomous components.

Differently from traditional component models, our model can semantically reason about the autonomous behaviors of autonomous components, such as when and how autonomous components start to pursue their goals, whether they are committed to providing services, and so on. The model can also reason about the relationships among autonomous components, such as composition and refinement.

By using the autonomous components modeled in this paper, we can construct Internetware from two directions, *i.e.*, goal-driven refinement from top to bottom and cooperation-based composition from bottom to up. Then, Internetware is feasible if the refinement process can find a group of cooperative autonomous components that can cooperatively achieve the goals of Internetware.

At next stage, we will develop a running environment for supporting the automated assembly of Internetware from autonomous components.

References

1. Ben-Shaul, I., Holder, O., Lavva, B. Dynamic adaptation and deployment of distributed components in Hadas IEEE Transactions on Software Engineering, Volume 27, Issue 9, Pages:769-787, Sept. 2001.
2. Brown, W. and Wallnau, K. The Current State of CBSE, IEEE Software, pp 37-46, October 1998.
3. J. E. Doran, S. Franklin, N. R. Jennings, and T. J. Norman. On Cooperation in Multi-Agent Systems. The Knowledge Engineering Review, 12(3), 309-314, 1997.
4. Garlan, D. Software Architecture: a Roadmap. In The Future of Software Engineering, Anthony Finkelstein (ed.), ACM Press, 2000.
5. Han, J. A Comprehensive Interface Definition Framework for Software Components. The 1998 Asia-Pacific Software Engineering Conference, Taipei, Taiwan, pp. 110–117, 1998.
6. Humberto, C. and Richard, H. Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model. 26th International Conference on Software Engineering (ICSE'04), Edinburgh, Scotland, United Kingdom. pp. 614-623, 2004.
7. Liu, Z.M., He, J.F., and Li, X.S. Contract-Oriented Development of Component Software. In the proceedings of IFIP WCC-TCS2004, Toulouse, France, pages 349–366, August 2004.
8. Martin, C. E., Macfadzean, R. H., and Barber, K. S. Supporting Dynamic Adaptive Autonomy for Agent-based Systems. Proceedings of 1996 Artificial Intelligence and Manufacturing Research Planning Workshop, Albuquerque, NM, pp. 112-120, June 1996.
9. Nickles, M., Rovatsos, M., and Weiss, G. A Schema for Specifying Computational Autonomy. Proceedings of the Third International Workshop "Engineering Societies in the Agents World" (ESAW'02), Madrid, Spain, Lecture Notes in Computer Science, Vol. 2577, pages 82–95. Springer-Verlag, 2002.
10. Sandholm, T.W. An implementation of the contract net protocol based on marginal cost calculations. In Proceedings of the National Conference on Artificial Intelligence, Washington, D.C., pages 256-262, July 1993.
11. Shaw, M., and Garlan, D. Software architecture: perspectives on an emerging discipline. Prentice Hall, 1996.
12. Shaw, M. Architectural Requirements for Computing with Coalitions of Resources. 1st Working IFIP Conference on Software Architecture, San Antonio, TX, USA, Feb 1999.
13. Smith R G, The contract net protocol: high-level communication and control in distributed problem solver, IEEE Trans on Computer, 29(1), pp.1104-1113, Dec. 1980.
14. Szyperski, C. Component Software –Beyond Object-Oriented Programming. Addison-Wesley, 1998.
15. Teschke, T. and Ritter, J. Towards a Foundation of Component-Oriented Software Reference Models. In Gregory Butler, Stan Jarzabek (Eds.): Generative and Component-Based Software Engineering, Second International Symposium, GCSE 2000, Erfurt, Germany, October 9-12, 2000, Revised Papers. Lecture Notes in Computer Science 2177 Springer, pp.70-84, 2001.

作者简介：

焦文品 男，1969 年生，博士，副教授，主要研究方向包括软件工程、智能软件、构件技术和形式化方法等。电话：010-62757670，E-mail：jwp@sei.pku.edu.cn。

朱萍萍 女，1980 年生，硕士。主要研究方向为软件复用与软件构件技术。

梅 宏 男，1963 年生，博士，教授，博士生导师，主要研究方向为软件工程、软件复用与软件构件技术、分布式对象技术等。

Wenpin Jiao, received his BA and MS degree in computer science from East China University of Science and Technology in 1991 and 1997, respectively, and Ph.D. degree in computer science from the Institute of Software at Chinese Academy of Sciences. From 2000 to 2002, he was a postdoctoral

fellow in the Department of Computer Science at the University of Victoria, Canada. Since 2004, he has been an associate professor in the School of Electronics Engineering at Peking University. His major research focus is on the autonomous component technology, multi-agent systems, and software engineering.

Pingping Zhu, received her BA and MS degree in computer science from Peking University in 2002 and 2005, respectively. Her research interest includes software reuse and software component technology.

Hong Mei, received his BA and MS degrees in computer science from Nanjing University of Aeronautics and Astronautics in 1984 and 1987, respectively; and Ph.D. degree in computer science from Shanghai Jiaotong University in 1992. From 1992 to 1994, he was a postdoctoral research fellow at Peking University. Since 1997, he has been a professor and Ph.D. advisor in the Department of Computer Science and Engineering at Peking University. He has also served as vice dean of the School of Electronics Engineering and the Capital Development Institute at Peking University, respectively.

His current research interests include: Software Engineering and Software Engineering Environment, Software Reuse and Software Component Technology, Distributed Object Technology, Software Production Technology, and Programming Language.

He is a member of the Expert Committee for Computer Science and Technology of State 863 High-Tech Program, a chief scientist of State 973 Fundamental Research Program, a consultant of Bell Labs Research China, the director of Special Interest Group of Software Engineering of China Computer Federation (CCF), a member of the Editorial Board of Sciences in China (Series F), ACTA ELECTRONICA SINICA and Journal of Software, and a guest professor of NUAA. He also served at various Program Committees of international conferences.