Soft-Device Inheritance in the Knowledge Grid

Hai Zhuge

China Knowledge Grid Research Group, Institute of Computing Technology,
Chinese Academy of Sciences, Beijing, China
zhuge@ict.ac.cn

Abstract. Soft-devices are configurable and adaptive software virtual mechanism, representing distributed network software and devices. Soft-devices and human users constitute a cyber-world in the Knowledge Grid environment—an intelligent interconnection environment that enables people and machines to work in cooperation. The cyber world determines its social evolution mode according to the development of nature, science, technology and human society. New soft-devices are generated by fusing or inheriting from existing ones. Soft-devices evolve according to the optimization and effectiveness principles. Interactions of different types in this cyber world form the self-organization structure. This keynote presents the framework of the soft-device world, the method for soft-device inheritance in the environment, and demonstrates the application in culture heritage exhibition.

1 Introduction

The future interconnection environment will be a cyber world where human society and the machinery interconnection world will interact each other, co-exist, and co-evolve [18].

The Knowledge Grid is a future intelligent interconnection environment. It enables people and machines to effectively capture, publish, share and manage knowledge resources. It also provides appropriate on-demand services for scientific research, technological innovation, cooperative teamwork, problem-solving, and decision making [17].

The Knowledge Grid involves in the following three basic scientific issues:

- *Interconnection semantics*—the study of the semantics in the interconnection environment for supporting intelligent applications by meaningfully interconnecting resources in semantic spaces where machines and human can understand each other.
- *Normalized Resource Organization*—the study of organizing resources in semantic normal forms to eliminate redundant, disorder and useless resources so as to ensure the correctness and accuracy of resource operations, and to realize complete and effective resource sharing.
- *Intelligent clustering and fusing*—the study of self-organization and optimization of complex systems.

V. Gorodetsky, J. Liu, and V.A. Skormin (Eds.): AIS-ADM 2005, LNAI 3505, pp. 62 – 78, 2005.
© Springer-Verlag Berlin Heidelberg 2005

This paper intends to answer the following fundamental questions:

What are the individual and community, and their structure in the Knowledge Grid environment? And, how the individuals are generated? How they evolve? And how they are organized to perform tasks?

To simplify complex systems is a task of science. Object-oriented methodology and programming languages use the notions of object and class to simplify the conceptualization of complex objective existence [3, 4]. The only way to generate a new object is by inheritance. But the object is too passive to represent active and adaptive organisms in real world.

Along with the simplicity, the Knowledge Grid environment also needs the diversity and effectiveness. On one hand, we need to ensure the simplicity by uniformly structuring the versatile resources. But on the other hand, we also need to guarantee the diversity of individuals by enabling adaptation and inheritance. To uniformly specify versatile passive and active resources in the Knowledge Grid environment, the framework of soft-device mechanism is proposed [16]. Soft-devices interact with each other and with human to form an evolving cyber world. Establishing competition mechanism in the world helps improve the effectiveness of the cyber world.

Inheritance is a natural phenomenon in the biological world. The heredity information keeps the evolution of species, and the mutation of the hereditary information makes the diversity of species.

Human society assigns inheritance social characteristics. For example, law and morals set the condition for marriage and inheritance. An individual is assigned the right to own the heritage of his/her parents. These social characteristics of inheritance differentiate human society from the animal world.

The rapid development of Internet applications challenges traditional software structure and development methodology. Previous inheritance mechanism is only suitable for stable and local environment [9]. How to realize inheritance mechanism in dynamic and distributed Knowledge Grid environment challenges software engineering in the global networking age.

2 Related Work on Inheritance

Inheritance in biology is about the principles that govern the inheritance of genes on sex chromosomes. Males and females differ in their sex chromosomes. Inheritance patterns for X-chromosome linked genes vary between sexes. Evolutionary computing borrowed this idea in finding solutions to problems of optimization. As a special relationship, inheritance was investigated in artificial intelligence and relevant areas [7, 10, 11, 12].

In object-oriented programming, inheritance mechanism is mainly for software reuse. Inheritance relationship forms class hierarchy, in which a subclass can inherit the states (variable declarations) and methods from the super-class. Subclasses can add variables and methods to the classes they inherit from the super-class. Subclasses can also override the inherited methods and provide specialized implementations for those methods. Inheritance enables programmers to reuse the code in the super-class

and implement subclasses by specializing the super-class. Such inheritance mechanism could raise the efficiency in software development.

From the viewpoint of interface management, inheritance was regarded as a combination operation $R=P \oplus \Delta R$ on record structures P and ΔR with possibly overlapping identifiers [13]. The combination operation $\oplus$ relates the interface of descendant R to the interface of parent P. Computer memory only has two relationships: references or contiguity. Therefore, two elementary strategies for implementing inheritance can be distinguished: *delegation* and *concatenation*. Delegation is a form of inheritance that implements interface combination by sharing the interface of the parent, i.e., using references. Concatenation achieves the same effect by copying the interface of the parent into the interface of the descendant. So, the descendant's interface will be contiguous. Due to different implementation, the way of message sending is different. In delegation, those messages that are not accepted by the most specialized part of the object must be delegated to parents. In concatenation, the interface of every object is self-contained, and thus no forwarding is needed.

There are different views on inheritance in object-oriented programming. One is that inheritance can only take place between classes. Object-oriented systems are built on classes. A class holds the similarities among a group of objects, dictating the structure and behavior of its instances, whereas instances hold the local data representing the state of the object. To add a new kind of object to a class-based system, a class describing the properties of that object type must be defined first.

Another alternative for traditional class inheritance is prototype inheritance, or object inheritance [3]. A prototype represents the default behavior for some concepts. There are no classes in prototype-based systems. Therefore, there is no notion of instantiation either. New object types are formed directly by constructing concrete, full-fledged objects (prototypes or exemplars). The distinction between the two types of inheritance reflects the philosophical dispute concerning the representation of abstractions. Some object-oriented languages like Smalltalk use classes to represent similarity among collections of objects. Prototype-based approaches view the world by exploiting alikeness rather than classification.

A characteristic of inheritance is sharing. There are two fundamental forms of sharing: *life-time sharing* and *creation-time sharing* [5]. Life-time sharing refers to physical sharing between parents and children. Once the sharing relationship between parent and child has been established, the child remains sharing the properties of its parent until the relationship is removed. Any change to the parent is implicitly reflected to the child. The creation-time sharing implies that sharing occurs only while the receiving process is in progress. It is characterized by the independent evolution of the parent and the child. Delegation discussed above results in life-time sharing, whereas concatenation results in creation-time sharing.

3 Soft-Device World

A soft-device is a network software virtual mechanism that can be configured for different purpose, can actively provide with services by automatically seeking

requirements, and can adapt to change [16]. The configurable feature develops early special-purpose computer as general-purpose computer that can run software of different types. The process of configuring a soft-device is similar to installing software in computer, which contains multi-step human-computer interactions.

The soft-device takes the advantages of the active and intelligent features of the intelligent agents, the semantic-based features of the semantic web, the advantages of the Knowledge Grid, and the configurable feature of general-purpose computers. Soft-devices hide versatile forms of resources like text, image, services etc, and could actively provide services according to the content of the resources specified inside and the configure information related to the resources.

A soft-device world consists of the self-organized soft-device society and the requirement space, and two roles: producer and consumer. The producer could add the resources to a soft-device and then configuring it. The consumers could enjoy the service provided by any soft-device in two ways: push and subscribe. Soft-devices could actively push services to the consumers, and can also accept subscription from consumers to provide long-term services. People could either play the role of producer to produce soft-devices by putting in the contents of relevant resources, or could play the role of consumer to enjoy the proper services by just posting their requirement or selecting and using an operable browser (soft-device) to get the required services. A soft-device could play the roles of both the producer and the consumer. A soft-device could accept content definition from multiple providers and provide services for multiple consumers with the consideration of a certain economic cost.

There are no central controls in the soft-device world, so the soft-device world is completely peer-to-peer [1]. Any hard device accessible across the Internet is regarded as a soft-device that provides the same service. The operation of a hard device is implemented by a corresponding soft-device.

A soft-device consists of the following components:

- *The detector* detects the requirements in the requirement space, processes the informal requirements and transforms them into formal, and then seeks solutions in resource spaces.
- *The interface* supports interaction between soft-devices and between components. It also supports the producer in defining resources' content and offers services to others by appropriate means. A browser is a kind of interface for conveying the output of the explainer to the consumer.
- *The explainer* explains requirements, information, knowledge and services.
- *The multiple built-in workflows* enable the soft-device to work according to requirements. The workflow should be time sensitive and adaptive. A control flow operates the soft-device by coordinating the execution of components.
- The reasoning mechanism inferences according to the knowledge specified in the knowledge space, a specialization of the resource space. The explainer interprets the reasoning result and submits it to the interface.
- *The coordinator* coordinates and adapts above components.

– *The Resource Space*. It specifies versatile resources represented in a machine-understandable way, which could be realized by markup languages and ontology mechanisms [18]. *The knowledge space* supports the operation of the explainer, the detector, and the coordinator. It also contains meta-knowledge for adapting the processes according to the changing situation. The meta-knowledge is specific to the type of soft-device.

The resource space in a soft-device (i.e., local resource space) has two parts: a private resource space and a sharable resource space where resources are accessible to other soft-devices. All the sharable resource space constitutes a peer-to-peer global resource space. The local sharable resources can be shared in the following two ways:

– Merge or join relevant local resource spaces to get the entire view where resources are uniformly classified.
– Establish semantic links between resources, and then locate the required resources by using peer-to-peer locating algorithms.

Fig. 1 shows the reference structure of soft-devices.

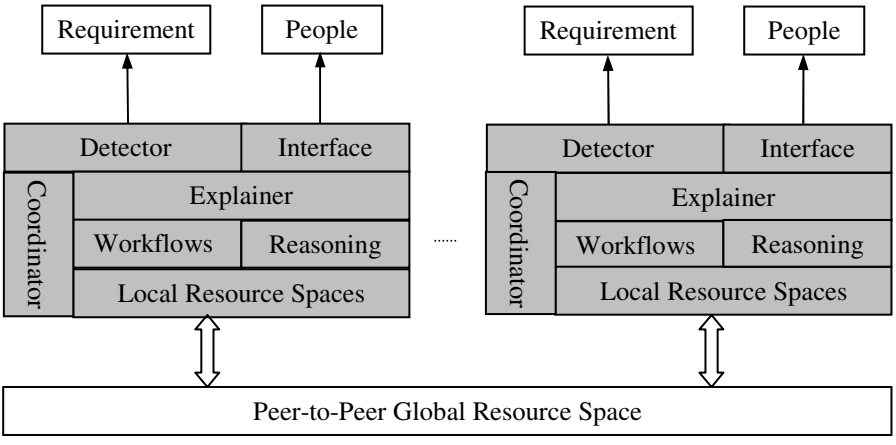

Fig. 1. The reference structure of soft-device

At the abstraction level, a soft-device can be defined as the encapsulation of processes (programs) on resource spaces, represented as *Soft-Device=<Resource-Space*, *Processes*, *Constraints>* and as *Soft-Device=<Resource-Space*, *Operations*, *Detector*, *Interface*, *Explainer*, *Workflows*, *Reasoning*, *Coordinator*, *Constraints>* in detail. The operations manage resources in the resource space under integrity constraints. The processes realize the functions of the soft-device by operating the detector, explainer, interface, coordinator, workflows, and reasoning mechanism. The resource spaces specify and organize information, knowledge and other resources in a normalized form [18]. Database and knowledge base are the instance of the resource

space. The processes defined in the soft-device should satisfy the constraints to ensure the process integrity [18]: a process output its result either to another process or to the resource space, and ensure that the input of a process is either from another resource or the resource space.

The soft-device world is a society that contains a market mechanism for pricing, exchanging, evaluating and rewarding the services that the soft-devices provided. Soft-devices organize themselves according to their relationships and interactions during service process. The society evolves with the addition of new soft-devices and the removal of the soft-devices that have never served for others. Soft-devices compete each other in the market to contribute service and to obtain reward (increase its rank) through the service evaluation and reward mechanism. Our experiment shows that the distribution of the ranks of soft-devices in the society is close to the power-law [2, 18], but the highest ranks will be blocked and average out under the limitation of social regulations.

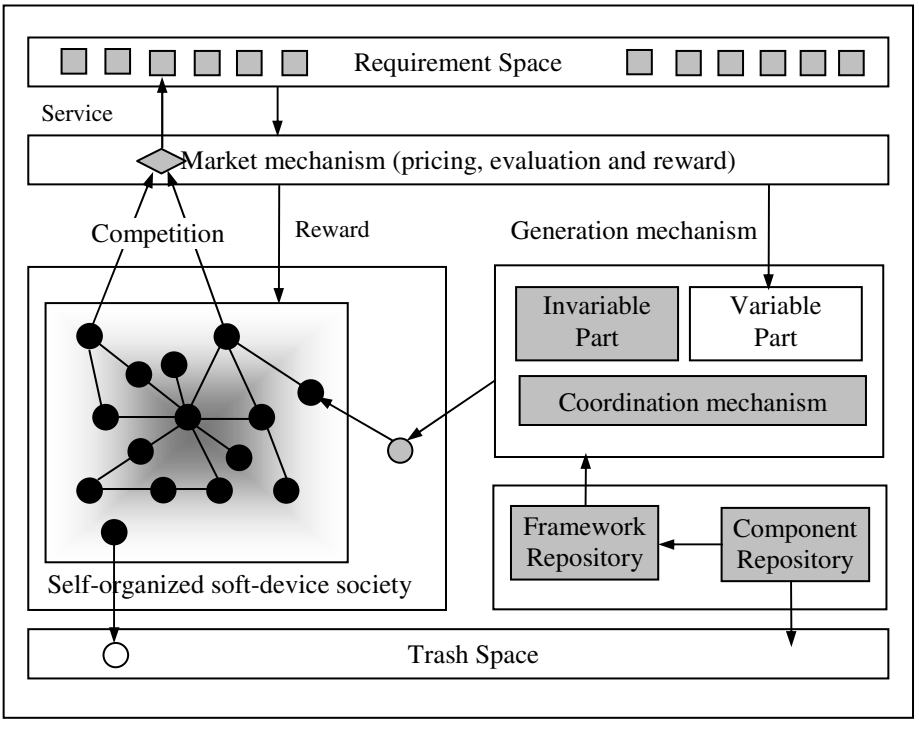

Fig. 2. The soft-device world

The soft-device world has a generation mechanism. New soft-devices are generated by getting a framework from the soft-device framework repository and then adding the variable part according to the request from the market mechanism.

4 Soft-Device Fusion and Inheritance

Two soft-devices can be loosely and tightly fused into a new one for a definite purpose by operations ⊕ and ⊗ as follows:

Soft-Device⊕*Soft-Device*'=<*Resource-Space*⊗*Resource-Space*', *Operation* ∪*Operation*', *Processes*∪*Processes*', *Constraints*∨*Constraints*'>, where ⊗ is the join or merge operations of two resource spaces [18]; ∪ is the union of two operation sets; ∪ also represents the merge of two processes such that *processes*(ξ)=*processes*∪*processes*'(ξ) for any input ξ in the input field of the *process*; and, *Constraints*∨*Constraints*' is the logic "or" of two constraints.

Similarly, *Soft-Device*⊗*Soft-Device*'=<*Resource-Space*⊗*Resource-Space*', *Operation* ∩*Operation*', *Processes*∩*Processes*', *Constraints*∧*Constraints*'>.

A soft-device can be minimized under certain conditions, represented as *Soft-Device*/*Condition*=<*Resource-Space*/*Condition*, *Operation*/*Condition*, *Processes*/*Condition*, *Constraints*∧*Condition*>, where *Resource-Space*/*Condition*, *Operation*/*Condition* and *Processes*/*Condition* are the minimization of the *Resource-Space*, *Operation* and *Processes* under the *Condition*.

Inheritance is an important way to generate new soft-devices in the soft-device world. Different from previous inheritance mechanisms, soft-device inheritance is dynamic in nature due to continuous evolution of soft-devices. The children inherited from the same soft-device at different time could be different and they will keep evolving after generated.

A new soft-device is generated by inheriting from an existing soft-device denoted as: New.Soft-Device→Soft-Device.

New.Soft-Device= <*Resource-Space*⊗Δ*Resource-Space*, *Operation*∪Δ*Operation*, *Detector*∪Δ*Detector*, *Interface*∪Δ*Interface*, *Explainer*∪Δ*Explainer*, *Workflows*∪ Δ*Workflows*, *Reasoning*∪Δ*Reasoning*, *Coordinator*∪Δ*Coordinator, Constraints* ∧*Constraints*'>.

The above inheritance relationship is transitive, i.e., New.New.Soft-Device→New.Soft-Device→Soft-Device⇒ New.New.Soft-Device→Soft-Device.

A new soft-device can also inherit from two or more soft-devices denoted as: *New.*(*Soft-Device*'⊗*Soft-Device*'') → *Soft-Device*'⊗*Soft-Device*''.

New.(*Soft-Device*'⊗*Soft-Device*'')= <*Resource-Space*'⊗*Resource-Space*''⊗ Δ*Resource-Space*, *Operation*'∪*Operation*''∪Δ*Operation*, *Detector*'∪*Detector*'' ∪ Δ*Detector*, *Interface*'∪*Interface*'' ∪ Δ*Interface*, *Explainer*'∪ *Explainer*'' ∪ Δ*Explainer*, *Workflows*' ∪*Workflows*'' ∪ Δ*Workflows*, *Reasoning*'∪ *Reasoning*''∪ Δ*Reasoning*, *Coordinator*' ∪*Coordinator*'' ∪ Δ*Coordinator*, *Constraints*'∧*Constraints*'' ∧ Δ*Constraints*>.

A soft-device evolves with three dimensions: *time*, *inheritance*, and *variety*. A soft-device can have a hierarchy of varies descendants, and changes itself with time due to the change of requirement and its status. The change of a soft-device automatically induces the upgrading of its children. The changed soft-device could also have new children. Fig 3 depicts the evolution of a soft-device in the three dimensional space.

Soft-device inheritance can take different forms:

1. *Time-sensitive inheritance*. *New.Soft-Device*(t+Δt)= <*Resource-Space*(t) ⊗Δ*Resource-Space*(t+Δt), *Detector*(t)∪ Δ*Detector*(t+Δt), *Interface*(t)∪

Δ*Interface*(t+Δt), *Explainer*(t)$\cup$ Δ*Explainer*(t+Δt), *Workflows*(t)$\cup$ Δ*Workflows* (t+Δt), *Reasoning*(t)$\cup$ Δ*Reasoning*(t+Δt), *Coordinator*(t)$\cup$ Δ*Coordinator*(t+Δt), *Constraints*(t)$\wedge$*Constraint*(t+Δt)>.

2. *Partial inheritance*. The ancestor can only allow its descendent to inherit a part of it, or the descendent only need to select a part of its ancestor to inherit, by using the minimization operation with given conditions. Inheritance rules can be derived to support advanced features based on the definition of the partial inheritance [14]. The constraint here is the selected part and the newly added part should satisfy the integrity constraints within a soft-device.

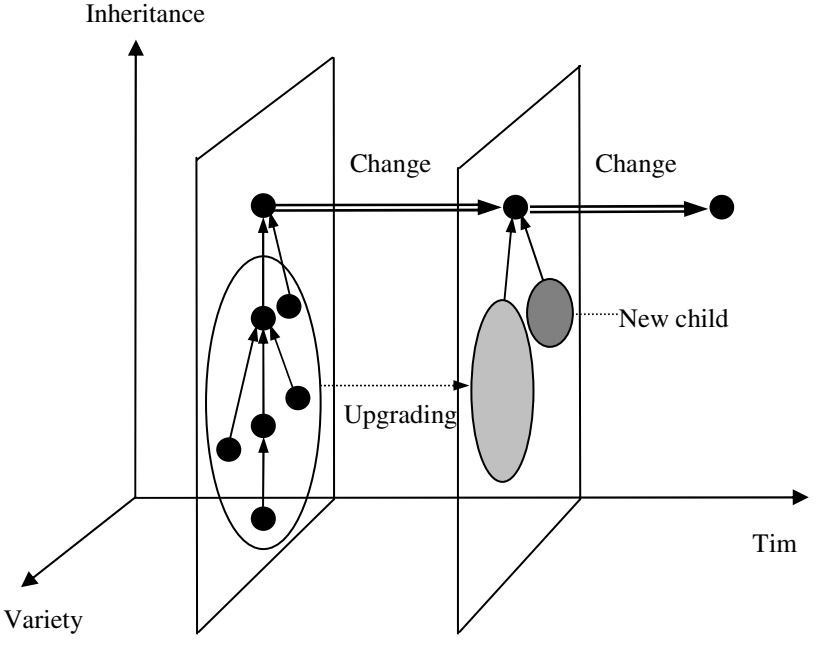

Fig. 3. Evolution of soft-devices

A soft-device society evolves with its social and market rules, which can be made with the reference to human society, as they need to harmoniously co-exist and evolve [18]. In the market mechanism, soft-device inheritance has the following social features:

- An evolving soft-device always tends to raise its rank (or reputation) in the society.
- The higher rank soft-devices tend to own higher rank descendents.
- A soft-device always intents to maximize its profit from the market if the soft-device society selects the free competition mechanism to organize themselves.

5 Self-organization and Self-adaptation of Soft-Devices

Self-organization has been investigated in Web structure, Web Services, and agent coalition formation [2, 8]. Soft-devices are self-organized through the dynamically evolved service spaces and the requirement space as well as the inheritance, knowledge and information flows as shown in Fig.4. The arrows in the soft-device space represent information and knowledge flows [15], the dashed arrows represent the operations for posting requirements, and the bi-arrows represent the matching between requirements and services. Applications initialize and add requirements in the requirement space. The person who posts a requirement needs to pay in the market and determines the initial price of the requirement. The fulfilled requirements will be removed from the requirement space. The requirement space evolves with the addition of new resources and removal of old requirements.

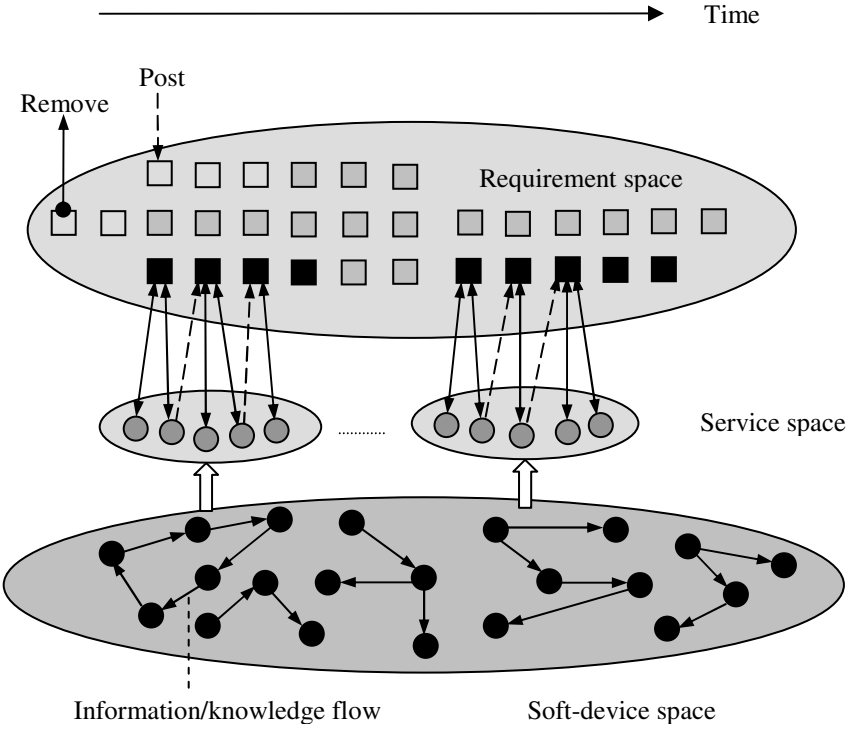

Fig. 4. Self-organization of soft-devices

Soft-devices will actively compete for providing services for the detected requirements and then get the reward. To ensure the justification in competition, a market mechanism selects soft-devices and logically organizes them together to fulfill the requirement. A selected soft-device can also post requirements in the requirement space to acquire services from other soft-devices.

Information and knowledge flow through soft-devices to enhance behaviors [15]. Competition in a large-scale soft-device world leads to unequal rank distribution among soft-devices [2]. Knowing the distribution law helps make routing strategy to enhance the efficiency of information and knowledge flows. The driving force of information and knowledge flows is the common interest in the involved soft-devices. The impact of the soft-devices in information and knowledge flow networks provide a reference evidence for them to make decisions to participate the market competition.

A soft-device will make decisions to survive during its life cycle under the regulations in the soft-device world. Decisions should benefit itself and the community it involved. To maximize the benefit, a soft-device should adapt its functions to meet the change of the society. It needs to consider the following factors when making decisions:

- *Adaptation cost*. Adaptation takes time for providing service to get profit. To own some necessary functions, the soft-device needs the cost for posting requirement and rewarding the services provided by others.
- *Market share and trend*. According to the market information and the competition, the soft-device will make the market share and trend analysis to determine the time for adaptation.
- *Profit*. The soft-device makes profit estimation according to the possible market shares and the adaptation cost.

The outcome of the decision is the adaptation target—the specification of new function. The coordinator mechanism will re-organize its functions according to the adaptation target.

6 Implementation

Since soft-device inheritance would carry out in a dynamic and distributed environment, using the delegation approach can facilitate the implementation because

- the parent and child can be loosely coupled (some object-oriented languages adopt the concatenation inheritance, but it is bounded so tight that any change on parent and child is not allowed), and,
- dynamic change of the parent can be facilitated.

A real example of the delegation approach is the proxy, which is a server that works between a client application and a real server. It intercepts all requests to the real server to see if it can fulfill the requests before it forwards the request to the real server. Proxy servers can filter requests and improve performance for groups of users because it saves the results of all the requests for a certain amount of time. The proxy server is often on the same network as with the user, so it can offer a much faster operation. Real proxy servers support hundreds or thousands of users. Some major online services such as America Online uses an array of proxy servers.

Since the services of a soft-device are offered through the interface, we can realize the proxy for soft-device interface. The interface services can be realized by using the

Remote Method Invocation (RMI), which allows the definition and the implementation to remain separately and to run on separate Java virtual machine. It processes as follows:

- The RMI server contains the implementation and interface of the remote soft-device.
- The RMI registers the remote soft-device.
- The RMI client finds the appropriate remote soft-device.
- The interface of the remote soft-device dynamically loads into local machine as a proxy.
- The RMI client uses the remote soft-device like local soft-device through the proxy.

The soft-device inheritance carries out as follows:

- The owner posts the static description of his/her soft-device in the requirement space.
- The child searches in the requirement space, and finds the required soft-device.
- According to the static description of the parent, add incremental part, and then form the static description of the child.
- The static description of the child is parsed to form a dynamic instance. The parent and child establish the message pipe in the meanwhile.

The changing of parent can be realized by changing the proxy with the consideration of two situations: the updated parent soft-device informs children, and the child changes its parent actively. In either situation, the child just needs to repeat the former two steps for updating itself. All operations can be done dynamically without recompiling and redeploying. But the children need to check the compatibility because some former services would still be used by its child even if they do not exist at present.

7 Application in Culture Heritage Exhibition

7.1 Using Soft-Devices to Animate Artifacts

The proposed approach is applying to animate Dunhuang cave culture—the precious world culture heritage. The artifact soft-device is a framework that can be configured to host programs for animating artifacts. A new artifact soft-device can be defined by inheriting from existing one. Fig.5 shows the effect of animating a wall-painting in Dunhuang cave (the up-left-hand image is the original). Fig. 6 shows another example of animating another Dunhuang wall-painting. The angels in the painting are animated by soft-devices, which can fly with different postures to form different meaningful patterns and interact with users upon pointed and clicked. The advantages of using soft-device technology include the following three aspects:

- The roles (e.g., angels) in the painting can actively and intelligently find clues among different roles in the same cave and in different caves made in different dynasty by querying each other and the sharable resource spaces. The advanced

features of the resource space model support the effectiveness of information and knowledge shared among soft-devices.
- The roles can be generated by inheriting from one predefined artifact soft-device. Such dynamic inheritance relationship structures the soft-device society, establishes the basis for culture evolution, and can raise the efficiency of animation.
- The artifact soft-devices provide applications and designers with components to create the animation. For example, the implementation of the posture of the same types of roles can adopt similar operations.

In the soft-device society, we have used the market selection mechanism (i.e., the utility) to measure the fitness in the competition environment. The problem is how to interpret the fitness in culture selection during the evolution of culture. Considering the features of the artifacts in the distributed caves, we use the amount of occurrence of culture features with the development of dynasties as the selection criteria based on the following two basic assumptions of evolutionary game theory (http://plato.stanford.edu/entries/game-evolutionary/):

- *Individuals always want more rather than less*, and
- *Interpersonal comparisons are meaningful*.

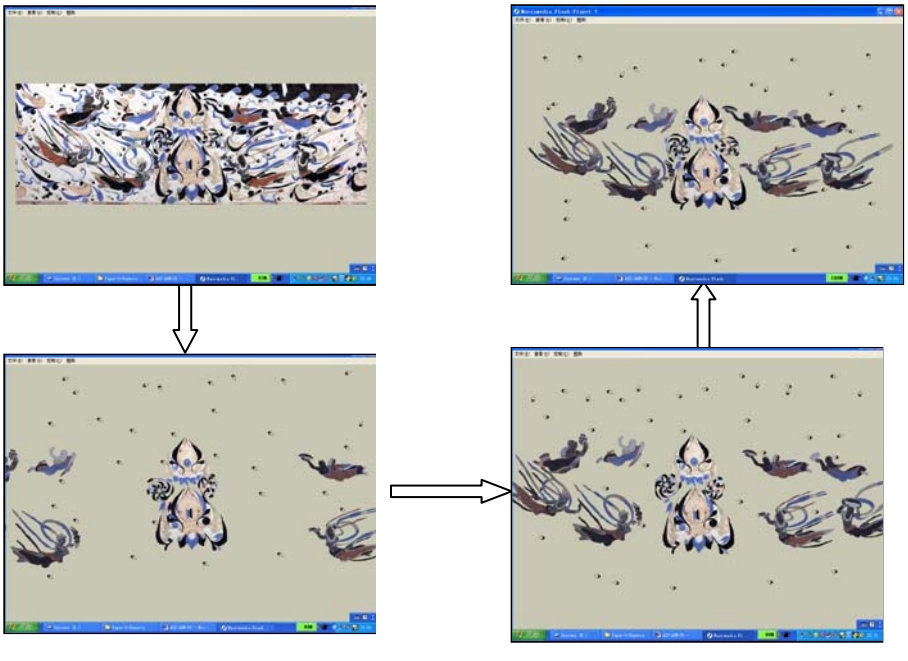

Fig. 5. Demo 1: animation of wall-paintings of Dunhuang

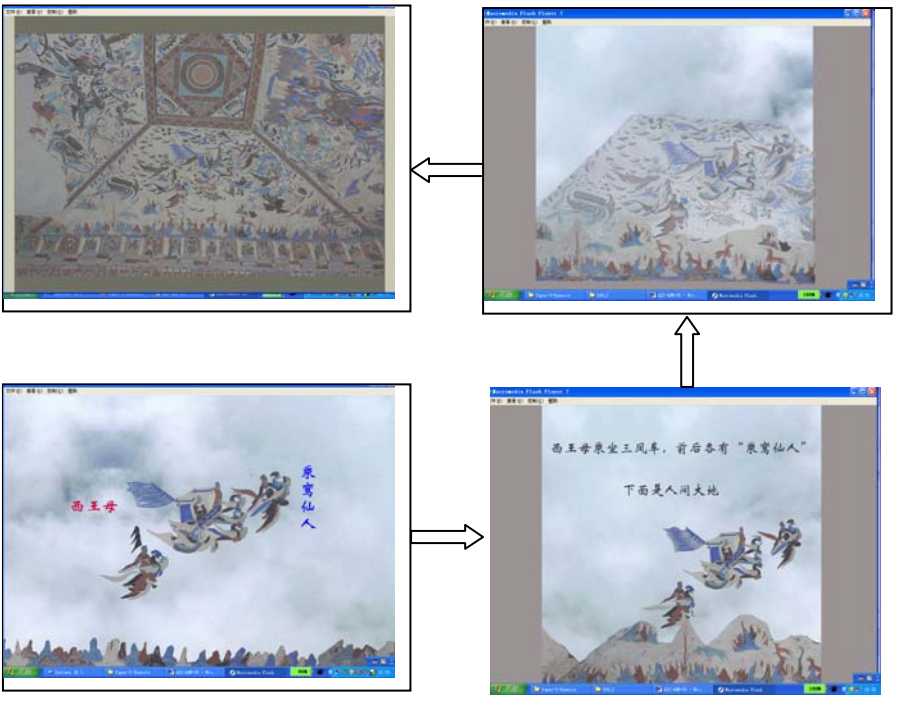

Fig. 6. Demo2: animation of wall-painting of Dunhuang

7.2 Culture Representation and Inheritance

Paintings tell stories and reflect artists' idea, feeling, and emotion. The roles in wall paintings can be made as soft-devices encapsulating the semantics of authors. The soft-devices can then actively communicate with each other and with human users when activated. Fig.7 shows an example of the artifact soft-devices, where the roles in the painting can perform like actors and explain themselves upon being clicked by users.

The scene of a wall painting can be also made as a soft-device composed by multiple soft-devices. By discovering the soft-device inheritance relationship between scenes, we can find the inheritance relationship between cultures of different dynasties.

A scene soft-device can be described by a semantic space and a set of operations as follows: *Scene-soft-device*=<*Semantic-Space*, *Operations*>. The semantic space here consists of the orthogonal multi-dimensional semantic space and the semantic links as shown in Fig.8 [17]: *Semantic-Space*=<{<*Author*, *Dynasty*, *Actors*, *Content*, *Cave*, *Resource-type*, *Artifact-type*>, <*Link-to-scenes*, *Link-to-resources*, *link-to-story*, *link-to-reality*, *link-to-knowledge*, *Semantic-relationship*>}, where the resource type refers to the text, image and sound. The artifact type refers to the building, color statue, and wall painting. The operations include {<*Move*(from <*x*, *y*> to <*x*', *y*'>)>,

Explanation(resource), *rotate-at*(<x, y>), *zoom-in*(<x, y>), *zoom-out*(<x, y>), *focus*(<*x*, *y*, *Radius*>)}>.

Fig. 7. Demo 3: Artifact soft-devices

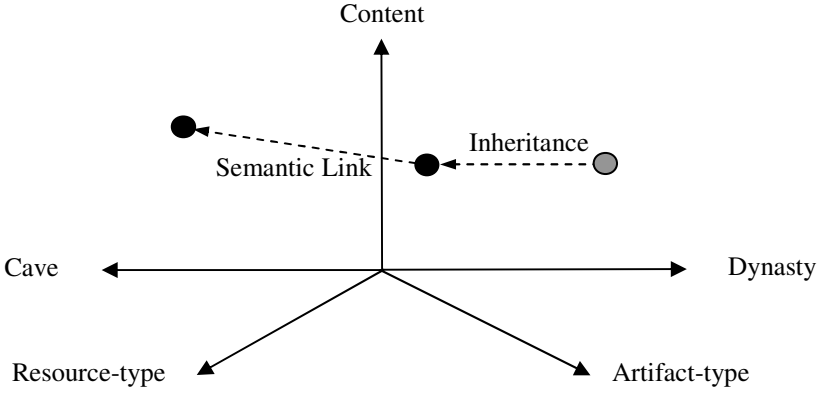

Fig. 8. Semantic Resource Space for Dunhuang Cave

Culture is represented by features like color, layout and architecture. Culture inheritance focuses on these features.

Cultural Inheritance. For two sets of features that describe scenes *A* and *A*': *A*=<{*Color*, *Style*, *Layout*, *Architecture*}, *Time*> and *A*'=<{*Color*', *Style*', *Layout*', *Architecture*'}, *Time*'>. If the *Color*, *Style*, *Layout*, and *Architecture* is similar to *Color*', *Style*', *Layout*', and *Architecture*' respectively, and *A*' is created after *A* (i.e.,

Time'>*Time*), then we can say that *A*' is cultural inheritance from *A*. Fig. 9 shows an example of cultural inheritance.

The key to determine the culture inheritance relationship is to find similarities between corresponding features. The following are some examples that help determination:

- Color *X* appears in *A* for describing *Y* also appears in *B* for describing *Y*.
- Shape *X* is similar to *Y* if they are geometrically similar to each other.
- Structure *X* is similar to *Y* if there exists an isomorphism between them.
- Layout *X* is similar to *Y* if there exists an isomorphism between them.

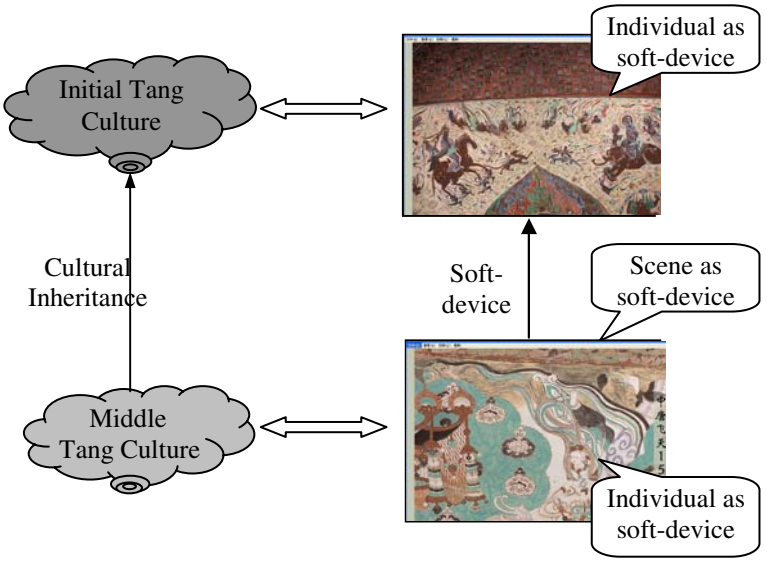

Fig. 9. From soft-device inheritance to culture inheritance

8 Conclusion

Soft-devices are configurable and adaptive, and can organize themselves to effectively perform tasks. Different from passive objects and active agents, soft-devices represent both passive and active resources in network environments. They can be as hard as hardware, as soft as software, and can even be a knowledge model that can actively detect problems and can fuse with and inherit from other knowledge to solve problems. The soft-device world obeys social and economical principles to ensure its simplicity, diversity and effectiveness. Soft-device inheritance is dynamic in nature. It enables the generation of new soft-devices in an evolving environment.

Research on soft-device will challenge current software structure and methodology. Incorporating domain features, soft-devices can be used to model any content-rich object. The cultural inheritance relationship can be automatically

discovered by finding the inheritance relationship between artifact soft-devices. This keynote also demonstrates the real application of the proposed soft-device approach in the animation of Dunhuang cave culture.

Acknowledgement

This work is supported by National Basic Research and Development Program (973, No. 2003CB317001) and National Science Foundation (Grant No. 60273020 and 70271007) of China. The author also thanks all the team members of China Knowledge Grid Research Group (http://kg.ict.ac.cn) for their help and cooperation, especially the members of the Knowledge Grid Center and the Dunhuang Knowledge Grid Laboratory (http://www.knowledgegrid.net).

References

1. Balakrishnan, H. et al.: Looking Up Data in P2P Systems. Communications of the ACM, 46 (2) (2003) 43–48
2. Barabási, A.L., Albert, R.: Emergence of Scaling in Random Networks. Science, 286, (1999) 509–512
3. Blaschek, G.: Object-Oriented Programming with Prototypes. Springer-Verlag (1994)
4. Booch, G.: Object Oriented Design with Applications. Redwood City, Calif.: Benjamin/Cummings Pub. Co. (1991)
5. Dony, C., Malenfant, J., Cointe, P.: Prototype-based Languages: From a New Taxonomy to Constructive Proposals and Their Validation. ACM SIGPLAN Not. 27, 10 (1992) 201–217
6. Foster, I.: Internet Computing and the Emerging Grid. Nature, 408, 6815 (2000) www.nature.com/nature/webmatters/grid/grid.html
7. Horty, J.F., Thomason, R.H., Touretzky, D.S.: A Skeptical Theory of Inheritance in Nonmonotonic Semantic Networks. Artificial Intelligence, 42 (1990) 311–348
8. Iamnitchi, A., Ripeanu, M., and Foster, I.: Small-world file-sharing communities. In Proceedings of the IEEE Infocom 2004, Hong Kong
9. Taivalsaari, A.: On the Notion of Inheritance. ACM Computing Surveys, 28, 3 (1996) 438–479
10. Tamma, V.A.M. and Bench-Capon, T.J.M.: Supporting Inheritance Mechanisms in Ontology Representation, EKAW 2000, LNAI, 1937 (2000) 140–155
11. Morgenstern, L.: Inheritance Comes of Age: Applying Non-monotonic Techniques to Problems in Industry. Artificial Intelligence. 103 (1998) 1–34
12. Nillson, N.: Principles of Artificial Intelligence. Tioga, Palo Alto (1980)
13. Wegner, P., Zdonik, S.B.: Inheritance as an Incremental Modification Mechanism or What Like Is and Isn't Like. ECOOP '88 (European Conference on Object-Oriented Programming)
14. Zhuge, H.: Inheritance Rules for Flexible Model Retrieval. Decision Support Systems, 22 (4) (1998) 379–390
15. Zhuge, H.: A Knowledge Flow Model for Peer-to-Peer Team Knowledge Sharing and Management. Expert Systems with Applications, 23, 1 (2002) 23–30

16. Zhuge, H. Clustering Soft-Devices in Semantic Grid. IEEE Computing in Science and Engineering, 4 (6) (2002) 60–62
17. Zhuge, H.: The Knowledge Grid. World Scientific Publishing Co. Singapore (2004)
18. Zhuge, H.: Toward the Eco-Grid: A Harmoniously Evolved Interconnection Environment. Communications of the ACM, 47, 9 (2004) 79–83