Evolutionary Modelling of Software Systems: its Application to Agent-Based and Hypermedia Systems[1]

M.J. Rodríguez-Fortiz*, P.Paderewski-Rodríguez*, L.García-Cabrera**, J.Parets-Llorca*
*Dpto.Lenguajes y Sistemas Informáticos
Granada University, Spain
mjfortiz, patricia, jparets@ugr.es
**Dpto.Lenguajes y Sistemas Informáticos
Jaén University, Spain
lina@ujaen.es

ABSTRACT
Evolution of software systems can be conceived from the Theory of Systems as a maturation process in which the developer have an active participation. This paper presents a two levels architecture: system and Meta-system. The developer interacts with the high level, Meta-system, for evolving the structure of the software system defined in the first level in which the user works. This architecture can be used to model the structure, functioning and evolution of any kind of software systems. In concrete, as the paper describes, it is applied to agents-based systems and hypermedia systems in a satisfactory way.

General Terms
Design, Theory.

Keywords
Software Evolution. Agent-Based Systems. Hypermedia Systems. Software Specification.

1. INTRODUCTION
Evolution is a crucial problem in software development. Very different views of this subject have been presented in the existent literature, in which, both the definition of the evolution concept and the approach to manage the evolutionary aspects are considered.

In order to model evolution we propose an architecture based on the Theory of Systems [20]. From this perspective, a *Software System* (SS) consists of a structure which maturate over time. A development team is in charge of modifying that structure in order to modify its functioning.

Our proposal includes two abstraction levels: the first level specifies the structure of a Software System; the second level, called *Meta-system* (MS), contains operations to modify the structure of the first level. Figure 1 shows the user working in the first level, running the structure of the software system, and the interaction of the developer with the Meta-system level.

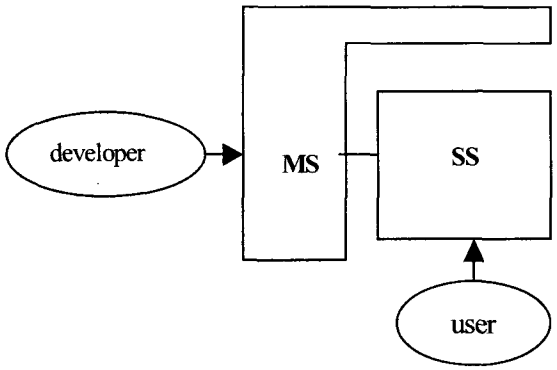


Figure 1. Software System and Meta-System levels

In previous works [2][26] we have presented formalisms useful in the specification and evolution of the structure of a software system. In this paper, our aim is the application of the approach in two kinds of systems: information systems based on agents and hypermedia systems.

In the second section, a taxonomy of evolution approaches in the literature will be shown. Third section will present the two levels architecture. Application to agent-based systems and hypermedia systems will be explained in the fourth, fifth and sixth sections. Finally, we will finish with our conclusions and future works.

2. PERSPECTIVES OF SOFTWARE EVOLUTION
The software evolution problem has been studied by different authors. Their works can be grouped taking into account different perspectives and objectives. The following six tendencies can be identified.

[1] This research is supported by a project –MEIGAS- by the Spanish CICYT (TIC2000-1673-C06-04) which is a subproject of the DOLMEN project (TIC2000-1673-C06).

1) Belady and Lehman [6] consider software evolution as the dynamic of the program evolution, establishing quantitative rules over the behaviour of a Software System and its development process. In recent works, Lehman and Ramil [21] focus their work in the what and why instead of the how of software evolution. They propose the use of formal theories in order to model the dynamics of systems.

2) During the eighties, new life cycle models tried to incorporate iteration in the software development process. The prototyping model, the spiral model of Boehm [8], and the Henderson model [18] are part of that group. All of them establish some basis useful in automating the software process, managing the workflow and controlling versions.

3) Pattern design which allow an evolutionary design of systems have been developed. Examples of it are the works of Aoyama [3], Niertrasz et al [24], and Amano et al [1]. The components of the architecture of a system can be modified. In order to guarantee a safe evolution, change impact and constraint management during evolution are studied.

4) Another considered perspective models the succession of different states in a system and the actions which produce these changes. The aim is to provide a model of the functioning of the system. State transition diagrams of UML [9] are an example of this approach.

5) Programs can be transformed in order to generate new versions which consider automatically the modifications in the requirements specifications, Kozaczynski [19], or preserve the original meaning but improve aspects as the efficiency of the code, Berzins et al [7]. Many authors, as Said [31], carry out automatic transactions based on quality factors.

6) The last perspective conceives that each Software System has a structure which could evolve and maturate. Evolution imply transitions of the structure by means of the activity of a development team. In order to study evolution, models which consider these structural transitions are elaborated. Banerjee [5] and Casais [10] incorporate concepts as the history of the evolution of an structure to describe the sequence of evolution transitions. In the same sense, Heckel [17], Wermenlinger [35] and Mens [23] propose to use graph rewriting as a formal tool used in a meta-model level to modify the structure of the programs, solving evolution problems.

Our work is centred in the last perspective because, as the previous authors, we conceive the evolution process like a maturation process. This maturation process begin when a system is conceived and it is carried out during the whole life of a system. We consider that evolution is: "A transformation of the structure over time, produced by the developer". Following this definition, the evolution of the Software System is the main functional capacity of the developer because it cannot evolve by itself. In order to execute this task, the developer must work in a higher level, that we call Meta-System level, using formalisms and tools to model the Software System of a lower level.

Next section will explain the proposed two levels architecture in a more detailed way.

3. TWO LEVEL ARCHITECTURE

In order to manage evolution, a two level architecture can be considered:

1. The first level specifies the structure of a Software System. A Software System runs over time. It has an structure which may be able to change, evolving and maturating over time and during its use. Based on the Theory of Systems, the functioning of a System depends on its structure at each moment. If the structure changes, the functioning changes.
2. The second level specifies the Meta-system. It is in charge of changing the structure, and therefore the functioning, of the Software System. The developer interacts at this level creating and modifying a Software System, specifying and designing it.

When an user decides to modify the functioning of a Software System, the developer will be informed about this and will change the structure of the Software System by means of the Meta-system.

The structure of a system determines which are its requirements. A Software System consists of *components* like agents or other systems which carry out *actions*. Actions have associated *restrictions* (pre or post conditions) which determine when they can be carried out. Restrictions are based in the previous state of the system. The objective of verifying the restrictions is maintaining the integrity of the system at each moment, taking into account its requirements.

Meta-system is also a software system but it does not changes its structure and its functioning is always the same. The developer uses it as a CASE tool which allows *evolutionary actions* creating and modifying the structure of any Software System.

It is not possible to carry out all of the desired changes in the Software System because, as stated before, integrity must be maintained during its evolution. In order to do this, a list of *invariants* must be established for the systems and a set of *meta-restrictions* must be specified and associated to the evolutionary actions. In the next paragraphs evolutionary actions, restrictions and meta-restrictions are better explained.

3.1 Evolutionary Actions

The objective of the evolutionary actions is the modification of the structure of a Software System (components, actions and restrictions). The set of evolutionary actions which can be carried out is:

- Adding, deleting or modifying the components of the structure.
- Adding, deleting or modifying the functioning actions that the Software System can carry out.
- Adding, deleting or modifying the restrictions associated to the functional actions.

Evolutionary actions must satisfy the invariants of the Software System *a priori* established. A list of invariants must be determined. This invariants depend on the general specification of the system and the desired characteristics. For example, an invariant could be: *"the names of the components of the structure must be different"*.

Two mechanisms are in charge of preserving these invariants:

- The meta-restrictions.
- The change propagation mechanism.

Meta-restrictions are pre or post conditions associated to the evolutionary actions. Each meta-restriction guarantees one or more invariants of the Software System definition. For example, the evolutionary action "*adding a new component*" of the Meta-system could have associated a meta-restriction, precondition, which proves that "*a component with the same name had not been added previously*".

Sometimes, an evolutionary action can modify a component of a system related to another component. This second component can be affected by the change in such way that another evolutionary action have to be carried out to adapt it to the change. The modification must be propagated to ensure the integrity of the whole system. For example, an invariant of the system could be: "*each component must belong to a cluster*". In this case, the evolutionary action "*deleting a component*", which deletes one of the components of a cluster of two components, should propagate the evolutionary action "*associating with a component*". This last action associates the remaining component to another component of a cluster in order to maintain the invariant.

3.2 Restrictions and Meta-Restrictions

Actions of a Software System and evolutionary actions of a Meta-system have associated restrictions and meta-restrictions, respectively.

A system must be always in an state of integrity, satisfying its invariants. The restrictions of the actions must be verified during its functioning. The meta-restrictions of the evolutionary actions must be verified during its evolution, i.e. before and after changes.

The state of a system at each moment depends on its previous functioning and structure, that is to say, on the previous actions and changes carried out. Consequently, the restrictions and meta-restrictions must hold on the previous actions and previous evolutionary actions, respectively. They must be modeled by means of temporal formalisms which allow to specify and verify the previous occurrences of actions and changes, and the order relationships between them. In previous works [15][29] we have proposed the use of Temporal Logic and Petri Nets to model restrictions, and Predicate Temporal Logic and Coloured Petri Nets to model meta-restrictions.

Figure 2 shows the steps followed to carry out a change in a Software System.

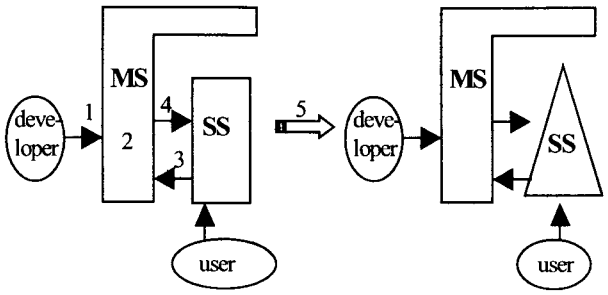
4. APPLICATIONS

Our intention in this paper is applying the proposed architecture of two levels to model the evolution of two kinds of systems: Information Systems based on agents and hypermedia systems.

The structure of an Information System can be composed by agents which interact carrying out actions. Sometimes agents must co-operate in carrying out more complex actions or transactions. The hierarchical relationship established between agents can determinate the way of co-operation.

Agent-based systems can be applied in a lot of fields like workflow, groupware and group decision support systems. The hierarchy of agents, the co-operation patterns and the description of transactions can be described with the proposed architecture. Besides, new evolution actions, and their pre and post conditions, have to be considered to establish how to modify the structure, respecting the integrity of the system at each moment.

The preparation of hyperdocuments includes a lot of changes, additions and updates. Hypermedia systems are dynamic or evolutionary by nature Following the same systemic perspective, their structure and functioning have to be specified and tools to be changed must be provided. Semantics of the information that they offer and the routes of navigation through this information are part of their structure, determine their functioning and can be changed. As other kinds of systems, invariants or restrictions must be verified during evolution and changes must be propagated.



1. Developer wants to change something
2. MS searches the associated meta-restriction
3. Previous structure and functioning of SS are queried
4. The meta-restriction hold, the change can be carried out
5. The SS structure have changed, and so its funcioning

Figure 2. Evolution process with meta-restrictions

5. AGENT-BASED SYSTEMS

A Software System can be conceived as a set of cooperating components wich carry out actions, we will call these components agents. An agent [12] is autonomous, independent and always is able to work if the environemnt is fevourable. Because, these agents should cooperate in order to carry out more complex tasks, a model of coordination between them should be stablished.

During the life of a Software System a lot of circumstances exists which produce changes. The environment of execution, the user conditions or the functionality can change. This possibility require an evolution of the system i.e, the addition of a new agent, the deletion of an agent, a change in the cooperation mechanisms,....

[16] propose a separation of the individual functioning of an agent and the cooperation mechanisms in order to increase the autonomy of the agents. This separation of concerns can be achieved using the previous two level architecture.

5.1 Structure of an Agent-Based System

An Agent-Based system is composed of a set of concurrent agents which carry out actions. The set of actions determine the functionality of the system.

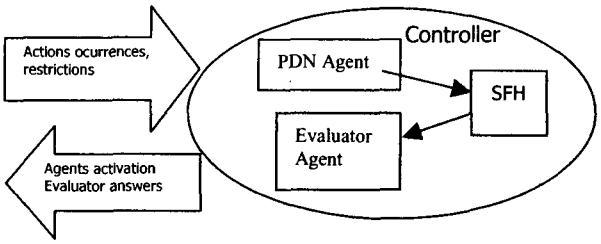


Figure 3. Structure of the Controller Agent

These actions can be classified as:

- Simple actions: actions carried out by one agent.
- Complex actions or transactions: a set of ordered actions carried out by a set of agents. A transaction should be atomic, i.e. from the environment point of view a transaction is a unique action.

Each action has a restriction (precondition) which determines the conditions of activation. This restriction is expressed as a condition over the state of the system, i.e. it holds when a sequence of actions have previously been produced in the system.

In order to coordinate the activity of the agents, a *blackboard* architecture provides a good design pattern. The *blackboard* architecture [33] consists of a central data structure which represents the state of the system and a collection of agents which operate in this structure. The state of the system, i.e. the sequence of previously executed actions, acts as a trigger for new actions. Because the blackboard stores the occurrences of actions carried out in the systems we will call it System Functional History (SFH). Each agent can query the SFH in order to know if the restrictions of its actions hold.

This architecture implies that the agents have no explicit knowledge about other agents. This independence between agents facilitates the evolution of the system.

5.1.1 A special agent: The Controller Agent

The previous structure presents three main problems:

1. The concurrent access for reading and writing to a central structure, the SFH.
2. The activation of the agents. Because the agents have to test the restrictions of their actions, they will be always testing the restrictions. This implies a loss of efficiency
3. The evaluation of restrictions should be carried out without interference of new actions. When a restriction is being tested the SFH can be modified.

In order to solve these problems an special agent, called *Controller*, is introduced. The *Controller* follows the design pattern called *PDN-Precondition Dynamic Notifier* [27]. This agent is in charge of providing services to the agents: evaluation of restrictions and reception of new action occurrences. In addition the *Controller* guarantees the consistency of the SFH and provides an order to the agent requests.

The second problem stated above is specially important in the +evolution of the system, i.e. when the restrictions of an action changes. This problem is solved preventing that an agent tests its restrictions when there are no possibilities of success. In order to accomplish this task, the *Controller* is in charge of activating the test of a restriction when certain possibility of success exists. This implies that the *Controller* has knowledge about the agents, their actions and restrictions, knowledge which change dynamically when the system evolves.

Figure 3 shows the structure of the *Controller*, which functionally is divided in the *PDN* and *Evaluator* agents. It contains also the SFH which stores the occurrences of actions. *PDN* agent stores occurrences of actions and activate agents. The *Evaluator* agent check action restrictions and provides an answer to the agents.

5.1.2 Agent nesting: complex actions (transactions)

A complex action is composed of simple actions. When an agent carry out a complex action it will be composed of simple agents which carry out the individual actions. The temporal results of a

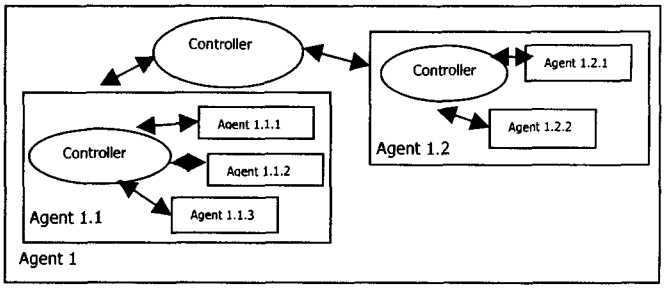


Figure 4. Transactions and Agent Nesting

complex action are temporally stored in the Controller of the nesting agent. Only when the transaction ends successfully the final result will be stored in the SFH of the main agent. Figure 4 shows an example of hierarchy of agents.

5.2 Evolution of Agent-Based Systems

A system evolves when its structural elements change. As previously stated, the Meta-system will allow us to change dynamically the modeled system. During the evolution of the System it stops its activity, but after the changes it will continue its work using the new structure and functionality.

The developer defines the set of evolution actions and the meta-restrictions needed for each type of system. The meta-restrictions guarantees the set of the required invariants for the system. In addition change propagation have to be defined for each evolution action.

The application of the two levels architecture to these systems is shown in the figure 5.

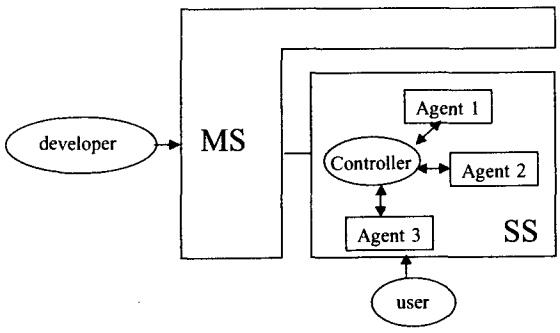


Figure 5. The two levels architecture for Agents-Based System

In agent-based systems, evolution actions can be carried out for the different elements of the system:

- Adding, deleting, nesting an agent.
- Adding, deleting actions. Changing the definition of a complex action (transaction).
- Adding, deleting, modifying a restriction of an action.

When an evolution action is carried out, the meta-restriction should be checked and the change propagation should be conducted. For instance, when an action is removed the meta-restriction is that "*the action is not a part of a transaction*". The invariant which guarantees this meta-restrictions is: "*each transaction is always composed by actions previously defined in the system*". If the meta-restriction holds, the Meta-system will propagate the changes in order to maintain the consistency of the system. In the example of removing an action, the propagation of change will consist in eliminating the references to this action in the restrictions of other actions.

In order to guarantees coherence and consistency, the evolution actions are communicated to the *Controller* which updates the information that it maintains about the components of the system: agents, actions and restrictions.

Obviously because the *Controller* is also an agent, its structure can also be changed, establishing the adequate evolution actions and meta-restrictions.

6. HYPERMEDIA SYSTEMS

All traditional models proposed for designing hypermedia systems try to model the process of the final edition of hyperdocuments and, sometimes, the process of the navigation performed by the reader. Nevertheless, the design, construction and evolution processes –the whole life-cycle- of hypermedia systems is not sufficiently considered [32]. However, this development process is very important because it implies a structuring process that is implicit, diluted and unaffordable inside the documents [25].

In our opinion, Hypermedia Systems are a special kind of Information Systems constructed over a conceptual domain. It represents some aspects and relationships of a conceptual domain explained by a set of authors. Because they include the knowledge captured by their authors, they are continuously changing. Changes can be carried out in the concepts offered by them, in the relationships between concepts, in the way of presenting or viewing the information and in the documents -information items- which explain the concepts.

From this discussion, a more extensive view of the development of hypermedia systems must be adopted:

- The development process of the hypermedia systems includes the memorisation and structuring of information domain, the possibility of offering different views of the same information domain and, finally, the possibility of offering different routes of navigation. Therefore, a hypermedia system can be conceived as a set of interacting systems in continuous evolution: It's necessary a *Systemic* perspective.
- In order to control this development process, the following elements should be provided: mechanisms for representing the information system; a representation of the conceptual domain or ontology [34] that information belongs to; useful ways of browsing and remembering the memorized knowledge. As a result, it is necessary to represent the *Semantic* of the process of construction –a cognitive model [14]-.
- Information systems, conceptual domains and navigation routes are exposed to continuous changes and updates which should be integrated in the development process: Hypermedia Systems are *Evolutionary*.

To sump up, a hypermedia model must integrate different systems which could be represented and evolved.

6.1 The structure of a hypermedia system

In order to specify the structure of a hypermedia system, aproposal for a Systemic, SEMantic, Evolutionary Model for Hypermedia Systems, SEM-HP, which maps the previous objectives is presented in this section. One such approach allows the construction, maintenance and navigation of information systems in continuous evolution and makes these activities more feasible, understandable and flexible.

A Hypermedia System can be conceived as being made up by two systems: The *Knowledge System* and the *Navigation System*.

Knowledge System is in charge of storage, structuring and maintenance of the different pieces of information. It memorises the acquired knowledge about the information system that is represented. This knowledge will guide the design and structuring processes of the information system. It will determine the possibilities for transformation and change of this structure throughout its evolution. It is made up by a *Memorisation Subsystem* and a *Presentation Subsystem*.

Navigation System helps the reader in his/her interaction with the information system following navigation routes throw the documents.

Consecutively, the characteristics of those systems are explained

6.1.1 The Memorisation Subsystem

The Memorisation Subsystem allows the storage of selected knowledge for each Information Domain –pages or documents-. It memorises information concerning the whole *Conceptual Domain*, which is managed in a particular information system. The conceptual domain is represented by means of a directed graph, in which, nodes and links are labeled with semantic meanings –a semantic net-. The graph represents the concepts and

relationships between concepts of the information system, named *Conceptual Structure* (CS). The different information items – documents- can be associated –labeled- with one or more concepts of the CS. These items are also nodes of the CS. Figure 6 shows an abstract example where *MS* is an artificial node which is the root of the represented information systems. Two conceptual structures are included (CA and CK).

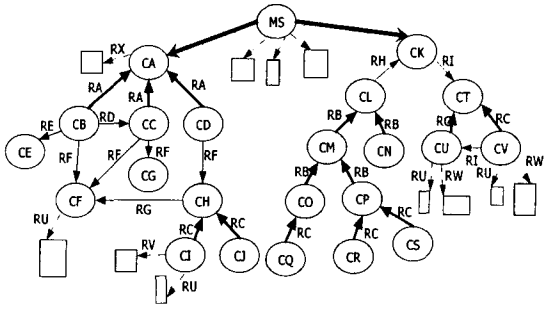


Figure 6. Examples of CSs of the Memorisation System.

The author can also include additional restrictions which determine what associations between concepts are possible. In order to represent these restrictions, formulas in temporal logic are used. This formalism also allows to check if the CS is valid at any moment. Some examples are:

- *"Concept-A can be connected with concept-B by means of the relationship-A".*
- *"The relationship-B must be acyclic".*
- *"Concept-C can be connected with concept-G if concept-C is reached from concept-B".*

6.1.2 The Presentation Subsystem

The Presentation Subsystem determines the set of possible views of a concrete Conceptual and Information Domain. To some extent it establishes the possible views of the hypermedia documents which can be built with the items of the Memorisation Subsystem. The Presentation Subsystem, using as basis the Conceptual Structure of the Memorisation System, allows a selection of a subset of the concepts and associations included in Conceptual Structure, creating a new graph CS_p.

6.1.3 The Navigation System

The Navigation System helps the reader in his/her interaction with the information system. Using the knowledge base and the reader's activity over time in a dynamic way, this system determines –firstly– the accessible information and –secondly– its interaction possibilities. The Navigation System, using as basis the CS_p of the Presentation Subsystem, allows adding some navigation restrictions in order to follow more restricted routes in the subgraph. These restrictions or navigation rules are expressed using temporal logic. Considering the CS_p and temporal restrictions, a Petri net is automatically constructed. Petri net is used to follow the navigation routes.

6.2 The evolution in a hypermedia system.

In order to manage evolution, the SEM-HP follows the general architecture described in section 3. Each one of the Systems of the approach has the two levels of abstraction: a) the Meta-system (MS) which is in charge of constructing and changing the structure of its SS and b) the Software-system (SS) which offer some special functionality.

The author (developer) creates or modify the SS, carrying out evolutionary actions, so he/she interacts with the Meta-System. Besides, at the same time, he/she can use the SS defined by each of these Systems in order to consult or check the structure. Therefore, the author interacts with all the Systems and can adopt the role of developer or user.

The reader only interacts with the Navigation System and makes only functional actions; he/she is a user of one of the Systems of SEM-HP (figure 7).

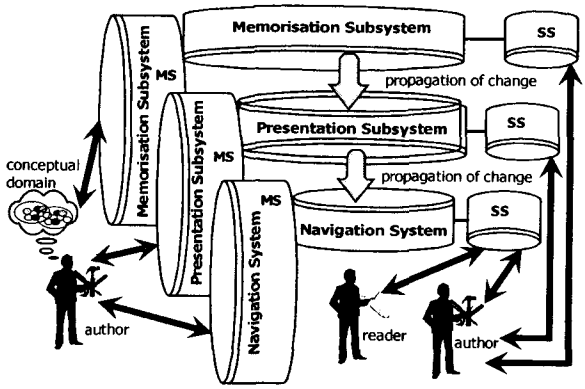


Figure 7. Architecture of SEM-HP

In order to drive and control the evolving construction process of the hypermedia system, each of the Systems of the SEM-HP support:

1. one or more components which represent the particular vision of the conceptual and information domain,
2. a set of restrictions,
3. a set of evolutionary actions that allow to make and propagate changes in the different Systems of the hypermedia system and,
4. meta-restrictions for controlling the construction and guaranteeing the consistency of the components and systems.

Three first types of elements –1,2 and 3- represent the structure of the different systems. This structure determines its functionality and can be modified by means of the evolutionary actions –4-. These actions must verify the meta-restrictions which can also be changed by the author.

When an evolutionary action is carried out by the autor, three types of tasks must be done by each of the systems in the Meta-system level:

1. Checking if the changed structure conserves integrity. Meta-restrictions, which maintains the invariants of the System, must be verified in order to guarantee the consistency.
2. Propagating the changes to the rest of the components of the system. In this case, the System propagates the change inside the System itself and its consistency is guaranteed.
3. Propagating the changes in System outside the System, i.e., to the other Systems of the SEM-HP. In this way, the integrity of the three Systems and, therefore of the hypermedia system, is guaranteed.

The evolution of each of the systems of the SEM-HP is explained in [15] and in next paragraphs.

6.2.1 The Memorisation Subsystem

During the development process, two aspects of this system can be changed, the CS –the graph- and the restrictions defined by the author. The Memorisation Subsystem always must guarantee the consistency of these changes –in the modified component, inside of the Subsystem and outside of the Subsystem-. Graph Theory is used to represent the evolutionary actions of the graph and their associated meta-restrictions.

The author –the developer-, an expert in a domain, designs the structure of the Subsystem. Its Meta-system provides the necessary evolutionary actions to modify its components: adding a concept, deleting an association, modifying an association, adding an item, etc.

The evolutionary actions must verify a set of meta-restrictions in order to maintain the invariants of the CS. Some examples of these invariants are: *"each association of the CS must connect two concepts or a concept and an item", "each arc and node of the CS must be labelled","two nodes in a CS cannot have the same label".*

Otherwise, changes in restrictions defined by the author (adding, deleting or modifying restrictions) must be defined by means of meta-restrictions written in temporal logic to refer to the previous state of the system.

6.2.2 The Presentation Subsystem

Using this system, the author can select a particular subgraph, CS_p, from one Conceptual Structure. In a similar way to the Memorisation Subsystem, the consistency must be guaranteed during the evolution of the Presentation Subsystem. In this system changes can be produced in the subgraph selected, CS_p. When the CS_p is changed –the author select another set of concepts and associations- the Meta-system level must check:

1. The CS_p verifies the restrictions defined by the system and the associations satisfy the set of restrictions defined by the author. These restrictions are the same of the Memorisation Subsystem because the Presentation Subsystem inherits them.
2. A new view or presentation is defined. In this case, the author must define again the navigation restrictions.
3. Changes must be propagated outside of the Navigation System.

6.2.3 The Navigation System

In this System, the structure is formed by a concrete presentation offered by the Presentation System, a set of navigation restrictions defined by the author and, finally, a Petri net constructed based on the two earlier components.

The Meta-system level of the Navigation System must guarantee the consistency again. When the author redefines –add, delete or modify- a navigation restriction, the system must check:

1. The set of restrictions that establish the order of navigation is consistent. Predicate temporal logic is used to specify the evolution operations over the restrictions, and their associated meta-Restrictions.
2. The navigation restrictions have changed. Changes in a restriction can imply the modification of other restrictions. The Petri net based on the navigation restrictions must evolve, generating it again -internal propagation of changes-.

7. CONCLUSIONS AND FUTURE WORKS

A two level architecture to model evolution have been presented. The higher abstraction level, the Meta-system, is used to specify the invariants of a Software System and to carry out changes in it. This approach allows to conceive the development and use of a system as a maturation process. The advantage of this general approach is that many different formalisms can be used in order to describe and modify different kinds of systems.

We have applied it to model the evolution of to kind of different systems: Information Systems based on agents and hypermedia systems. We have shown how their structure can be specified, the actions that they can carry out and the restrictions associated to them. Some formalisms have been presented to describe and evolve this structure. Besides we have proposed temporal formalisms to specify the structure, evolutionary actions and meta-restrictions of the Meta-system.

This architecture, which was implemented in a tool called HEDES [28] for agent-based systems, is being extended now for hypermedia systems following the same philosophy: evolutionary actions and meta-restrictions which verify before carrying out changes. The meta-restrictions are modelled using different formalisms which allows us to model structure and time (Graphs, Temporal logic and Petri Nets).

8. REFERENCES

[1] Amano, M; Watanabe, T: An Approach for constructing Component-base Software Systems with Dynamic Adaptability using LEAD++. Principles of Software Evolution. ISPSE 2000. IEEE Computer Society00) 118-127

[2] Anaya, A; Rodríguez, M.J.; Parets J.: Representation and management of memory and decision in evolving software systemsComputer Aided Systems Theory- EUROCAST'97. LNCS 1333 Berlin: Springer-Verlag (1997). 71-82.

[3] Aoyama, M. : Evolutionary Patterns of Design and Design Patterns. Principles of Software Evolution. ISPSE 2000. IEEE Computer Society (2000) 110-117

[4] Bacon, J.: Concurrent Systems. Operating Systems, Database and Distributed Systems: An Integrated Approach. Addison Wesley, 1998.

[5] Banerjee, J., Kim, W., Kim, H.K., Korth, H.F.: Semantics and implementation of schema evolution in object-oriented databases, In Proc. Of ACM-SIGMOD International Conference on Management of Data, San Francisco, 1987

[6] Belady L.A, Lehman, M.M.: A Model of Large Program Development, IBM SYST.J. Vol. 15.3, 225-252. (1976).

[7] Berzins , V., Luqi, Yehudai, A:. Using Transformations in Specification-Based Prototyping, IEEE Trans.S.E. Vol. 19.5, 436-452 (1993.)

[8] Boehm, B.W.: A Spiral Model of Software Development and Enhancement, ACM SIGSOFT S.E.NOTES. AUGUST Vol. 11, 14-24 (1986)

[9] Booch, G. et al.: The Unified Modeling Language. Reference Manual. Addison-Wesley (1999).

[10] Casais, E.: Managing Class Evolution in Object-Oriented Systems. In: Object Management. Tsichritzis, D. GENEVE. Centre Universitaire d'Informatique. 133-195 (1990)

[11] Charlton, P.: Self-Configurable Software Agents. Advances in Object-Oriented Metalevel Architectures and Reflection. Chris Zimmermann (ed.). CRC Press. (1996) 103-127.

[12] Franklin, S; Graesser, A: Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. Proceedings of the Third International Worksshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

[13] Gabbay, D.M.; Hodkinson, M., Reynolds, I.: Temporal logic. Mathematical Foundations and Computational Aspects, Volume 1. Oxford Science Publications. Oxford Logic Guides:28, 1995.

[14] García-Cabrera, L.; Parets-Llorca, J.: A Cognitive Model for Adaptive Hypermedia Systems. The 1st International Conference on WISE, Workshop on World Wide Web Semantincs. Hong-Kong, China, June 2000, 29-33.

[15] García-Cabrera, L.; Rodríguez-Fórtiz, M. J; Parets-Llorca, J.: Formal Foundations for the Evolution of Hypermedia Systems. 5th European Conference on Software Maintenance and Reengineering, Workshop on FFSE. IEEE Press. Lisbon, Portugal, March (2001) 5-12.

[16] Gelernter, D.; Carreiro, N.: Coordination Languages and their Significance. Communication of ACM, vol. 35, no.2, 1992.

[17] Heckel, R; Engels, G: Graph Transformation as a Meta Language for Dynamic Modelling and Model Evolution. Formal Foundations for the Evolution of Hypermedia Systems. 5th European Conference on Software Maintenance and Reengineering, Workshop on FFSE. IEEE Press. Lisbon, Portugal, March (2001) 42-47.

[18] Henderson-Sellers, B.; Edwards, J.L.: The Object-Oriented Systems Life Cycle, CACM 33.9. p.143-159. (1990)

[19] Kozaczynsky, W., Ning, J., Engberts, A.: Program Concept Recognition and Transformation, IEEE Trans.S.E. 18,12. p.1065-1075 (1992)

[20] Le Moigne, J-L: La théorie du système général. Théorie de la modélisation, PARIS, Presses Universitaires de France (1977-1990)

[21] Lehman, M; Ramil, J: Towards a Theory of Software Evolution- And its Practical Impact. Principles of Software Evolution. ISPSE 2000. IEEE Computer Society (2000) 2-13

[22] Lin, C.; Chaudhury, A.; Whinston, A. B.; Marinescu, D. C. "Logical Inference of Horn Clauses in Petri Net Models". IEEE Transactions on Knowledge and Data Engineering, vol 5,3. pp: 416-425. 1993.

[23] Mens, T: Transformational Software Evolution by Assertions. Formal Foundations for the Evolution of Hypermedia Systems. 5th European Conference on Software Maintenance and Reengineering, Workshop on FFSE. IEEE Press. Lisbon, Portugal, March (2001) 67-74

[24] Niertrasz, O; Achermann , F Supporting Compositional Styles for Software Evolution. Principles of Software Evolution. ISPSE 2000. IEEE Computer Society 14-22

[25] Nürnberg, P.J.; Leggett, J.J.; Schneider, E.R. As We Should Have Thought, Hypertext'97 Proc., ACM Press: 96-101.

[26] Paderewski, P.; J. Parets-Llorca, Anaya A., Rodriguez M.J., G. Sanchez, J. Torres, M.V. Hurtado: A Software Development Tool for Evolutionary Prototyping of Information Systems. Computers and Computational Engineering in Control. Electric and Computer Engineering Series. World Scientific and Engineering Society Press (1999) 347-352.

[27] Paderewski, P.; Parets, J.: Un patrón de activación de objetos activos. Jornadas de Ingeniería del Softwaere y Bases de Datos, JISBD'99 (1999) 257-268

[28] Rodriguez, M.J.; Parets, J.;Paderewski, P.; Anaya, A.; Hurtado, M.V.: HEDES: A System Theory based tool to support evolutionary Software Systems. Lectures Notes in Computer Science, Vol. 1798, pp.450-464. Springer Verlag (2000)

[29] Rodriguez-Fortiz, M.J, Parets Llorca, J. Using Predicate Temporal Logic and Coloured Petri Nets to specifying integrity restrictions in the structural evolution of temporal active systems. Principles of Software Evolution. ISPSE 2000. IEEE Computer Society (2000) 83-89

[30] Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall, 1998.

[31] Said, J; Steegmans, E: Transformations of Binary relations into Associations and Nested Classes. Formal Foundations for the Evolution of Hypermedia Systems. 5th European Conference on Software Maintenance and Reengineering, Workshop on FFSE. IEEE Press. Lisbon, Portugal, March (2001) 75-82

[32] Schnase, J.L., Leggett, J.J, Hicks, D.L., Szabo, R.L. Semantic Data Modelling of Hypermedia Associations, ACM Trans. Information Systems, 11(1):27-50, January 1993.

[33] Shaw, M., Garlan, D.: Software Architecture. Perspectives and emergencing discipline. Prentice Hall. 1996.

[34] Uschold, M.: Ontologies: Principles, Methods and Applications. Knowledge Engineering Review, vol. 11, n. 2, June (1996).

[35] Wermelinger, M; Lopes, A; Fiadeiro, J. L.: A Graph Transformation Approach to Architectural Run-Time Reconfiguration. Formal Foundations for the Evolution of Hypermedia Systems. 5th European Conference on Software Maintenance and Reengineering, Workshop on FFSE. IEEE Press. Lisbon, Portugal, March (2001) 59-66.