A Multi-Agent Framework for an Architecting Process Environment: A Position Paper

Rodion M. Podorozhny
UT Advanced Research In Software Engineering (UT ARISE)
University of Texas
Austin, Texas 78712
podorozh@mail.utexas.edu

Dewayne E. Perry
UT Advanced Research In Software Engineering (UT ARISE)
University of Texas
Austin, Texas 78712
perry@ece.utexas.edu

ABSTRACT

This paper describes experience in extending and generalizing the design of the multi-agent aspect of an existing software process environment [1]. The software processes used for gaining experience with the system focus on recovering the software architecture of an existing software system. These software processes are considered to be architecting processes as they manipulate and produce software architecture [2] artifacts.

The main goal of the framework is to simplify specification and development of software processes in an agent-based software process execution system. The described multi-agent framework suggests a general approach to modeling agents needed for software process specification, allows for greater software reuse, and accelerates agent specification.

Although a rigorous and complete specification of the model and system is beyond the scope of this paper, the descriptions provided suffice to support an explanation of the experience we have had in applying our multi-agent framework.

The result was greater clarity about the architecture of agents in SPEs, a more comprehensive (and comprehensible) process specification, provisions for rigorous software process analysis, and an acceleration of process specification activity itself.

A range of future work in this area is indicated at the conclusion of the paper.

Keywords

Multi-agent systems, software architecture, software process, process programming

1. INTRODUCTION

Software processes environments (SPEs) are a type of software systems that benefits from agent-oriented approaches. This is due primarily to a reasonable mapping of the environment for software process execution onto an agent-based framework. The software process specifications can be viewed as templates, whose decomposition units (steps) are executed by humans or automated tools (agents) of some concrete environment. Autonomy is one of the main properties of the notion of an agent. Such entities as humans or automated tools used in software processes quite often exhibit a great deal of autonomy, so an agent-oriented approach seems to be better suited for modeling software process environments than other approaches, for example, an object-oriented approach. Software development processes can (and in many cases do) have several thousands of agents executing.

Some SPEs (Little-JIL/Juliette [13], APEL [3], DYNAMITE [6]) describe entities similar to agents, even though not always in exactly the same sense as in the AI multi-agent community. Most SPEs focus on the process specification formalisms rather than the details of modeling the environment in which those processes would execute. Consequently little attention has been given to description of agents, their place, and their structure in SPEs. Filling this void is beneficial to both the multi-agent and the software process communities.

This work resulted from the need to accelerate software process specifications and their enaction in agent-based software process environments. Agent-based SPEs require the specification of agents as part of the specification of the software processes. The SPEs (e.g. Little-JIL/Juliette [8], [13]), as well as AI multi-agent frameworks (e.g. JAF [12]), provide for substantial flexibility in agent specifications. The downside of such freedom is an almost complete lack of guidelines

[1] The notion of software process environment is understood as a system for supporting the development, execution, evaluation, and evolution of a broad range of software development processes encoded in a rigorous software process language.

[2] The notion of software architecture is understood as it is defined in ([9]), namely: a set of a software system's elements, their interactions, and the constraints on those elements and their interactions necessary to provide a framework in which to satisfy the requirements for the software system and serve as a basis for the design.

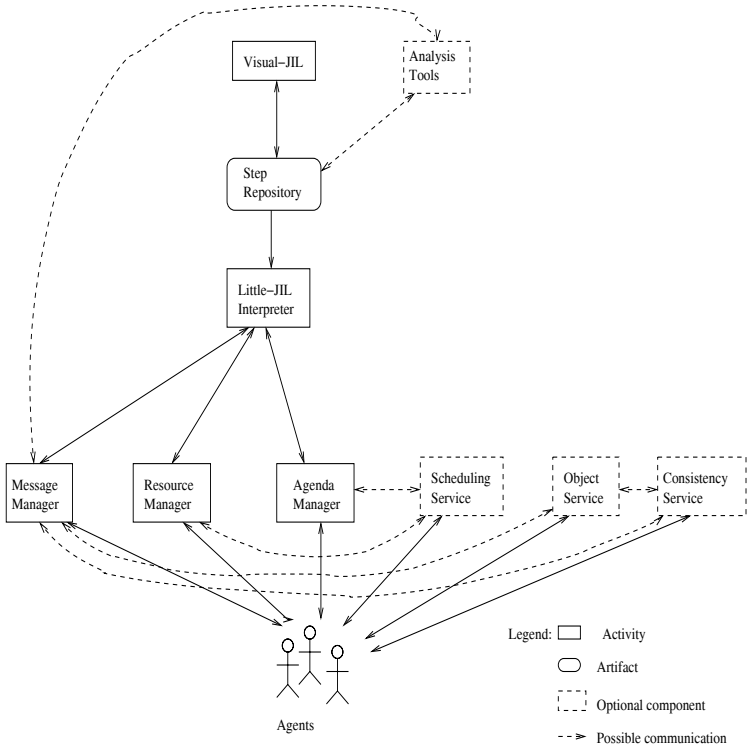

Figure 1: Little-JIL environment architecture

for the problem-solving part of agent specifications. Thus the agent specification becomes one of the major bottlenecks in process specifications, as every new process, even in the same domain, requires agents to be specified almost from scratch.

The aim of this work is to simplify the specification and development of software processes in agent-based SPEs by constraining the agent structure, making it more specific, making it more immediately suitable for the needs of software processes. Our framework presents an initial attempt at extending an existing, agent-based SPE (Little-JIL/Juliette [13]) with an agent framework for execution of processes. We illustrate our approach by using processes dealing with software architectures. We show that by restricting the structure of an agent and providing agent templates we gain a great deal of leverage in defining agent-based processes.

2. OVERVIEW

The Little-JIL/Juliette SPE was developed at the Laboratory for Advanced Software Engineering at UMass, Amherst. Juliette is a Software Process Environment that supports the development and execution of software processes represented in a software process language, Little-JIL. The high-level design of the Little-JIL/Juliette SPE is shown in Fig. 1

Little-JIL is a visual language intended for the specification of software processes. Programs in Little-JIL describe the coordination and communication between agents that enable them to perform a process.

Little-JIL agents are intelligent, autonomous entities that are experts in some part of the process described by a Little-JIL program. Agents may be human or automata. In either case, they are assigned work and required to report back the success or failure of that work when they are done.

Steps are specifications of units of work assigned to agents. Each step contains information and resources that are required (e.g., a detailed design for a programming task), preconditions that must be satisfied before an agent can begin the work, the decomposition of the work into smaller steps, and post-conditions to check that the work was completed correctly.

Parents and children in the sub-step hierarchy of a Little-JIL program can transmit objects between each other. The objects are stored in a persistent repository managed by an object manager.

Different agents are assigned to some step and its substeps. Thus the assignment of an agent to a step means that the agent has the responsibility for accomplishing that step rather than executing all of the substeps of that step.

A step is assigned to an agent by posting it on an agenda. Each agent has one or more agendas that the agent can examine to determine the work assigned to it. An agent can choose different alternatives of accomplishing an agenda item corresponding to some step. The alternatives can be either elaborated in the process represented by Little-JIL, both in pro-active and reactive control, or they can be "hidden" in agent's implementation. The current state of the

environment (for example, resource model) can influence the agent's decision making. There is a provision for agents to exchange messages directly, rather than transmit them along the sub-step hierarchy. It is mainly these features that enable us to claim that agents in Little-JIL are autonomous. Currently, the Juliette SPE lacks a scheduling component which somewhat constrains the degree of variance of alternatives. Most AI multi-agent systems (e.g. JAF [12], DECAF [5]) provide its agents with complex schedulers that help arriving at alternative ways to accomplish a certain goal, producing both the sets of predefined atomic operations and their schedules. Negotiation mechanisms in such AI multi-agent systems increase the overall utility of the actions of a set of agents by helping them conduct a distributed search over their local schedules. The agent-oriented SPEs are moving in that direction, while placing a greater emphasis on the fact that software processes almost always involve humans as agents and are tending to define alternatives explicitly in the process specification, both via pro-active and reactive control constructs. Nevertheless, we feel that SPEs, Little-JIL/Juliette in particular, provide enough autonomy for its agents to be viewed as multi-agent systems.

An agenda item corresponding to some concrete step goes through several states according to predefined finite-state machine (among others, they include STARTED, COMPLETED, RETRACTED, CANCELED). The Juliette SPE only requires that agents be able to listen to agenda item events. No guidelines are given as to the structure of the agent and its problem solving component. Neither are there any guidelines about the representation of artifacts. The software process designer using the Little-JIL is left to make his/her own decisions regarding the structure of the agents and regarding the format of the communication between them. After specifying several software processes and building agent structures from scratch a software process designer is likely to recognize that it might be beneficial to introduce more constrained frameworks that provide a predifined structure for the agents that spells out how to go about constructing an agent and enables the reuse of the agents originally written for other processes in the same problem domain. It is understandable that the freedom the stock Little-JIL provides is needed because it is impossible to envision and generalize all areas of application. On the other hand, once such areas of application are recognized, it is beneficial to introduce extensions to the SPE specific to that application area.

The topic of this work is description of an agent framework which is part of such an extension for processes dealing with software architectures. To a great extent, it is the nature of the process artifacts that define the agent's problem solving component and its functionality in our framework. We believe that the architecting processes serve as a surrogate for the rest of the development processes. Consequently, we believe that our framework would apply to other development processes equally well.

In case of architecting processes, the artifacts are software architecture and design specifications represented with architecture description languages. The steps of architecting processes manipulate these specifications by adding, deleting or reorganizing their components. Thus the activities of agents that execute such steps can be thought of as productions. The notion of production is used because such activities most often transform an expression of the language used to represent the input artifact into the language of the output artifact (which can be a different subset of the same language). The set of productions used by some agent can vary and it depends on the needs of the executed process. The design and architecture specifications themselves can be represented in a meta-language such as XML (e.g. XArch project [4]) to generalize some common manipulation activities that do not depend on peculiarities of specific architecture description languages. The use of XML for artifact representation also makes it easier to integrate XML-aware external tools. Because the software processes often require humans as agents it is important to make provisions for step-specific or artifact specific GUIs to help them.

This reasoning led to development of the agent framework for architecting processes shown in Fig. 2. The architecting agent class obligates its subclasses to have a certain common functionality set and a certain common structure specifically suited for architecting process agents. Part of that common functionality provides for communication with the external environment, in this case - with other components of the Juliette SPE (such as Little-JIL language interpreter).

The process specific agent class has some knowledge of the steps that might appear in a certain process. This class essentially matches the steps and activities that the agent would execute to accomplish those steps. Depending on the degree of desired automation the agents can be:

- Human assistants (such an agent would invoke a step-specific GUI and try to assist the human in accomplishing the step)
- Human-modeling (such an agent would attempt to model the duties of a human for process simulation or guidance purposes)
- Automated (such an agent would accomplish steps amenable to complete automation)

It is because of this that we get a very large set of agents in existence at any one time: large numbers of human agents are executing the software process concurrently and may use one or more of these agents for its own purposes for interaction, guidance or automation.

Any software process is likely to require step specific GUIs. This need is recognized in the agent framework by defining a set of step specific GUIs that is checked by the agent every time it receives a new agenda item. The table that puts GUI classes and process steps in correspondence is defined as part of the process specification.

3. EXPERIENCES WITH OUR APPROACH

The agent framework described above has been used to model several simple architecture recovery processes. The framework made it much easier to implement new processes. Only the description of the set of steps recognized by the agent and the productions it would use to accomplish them had

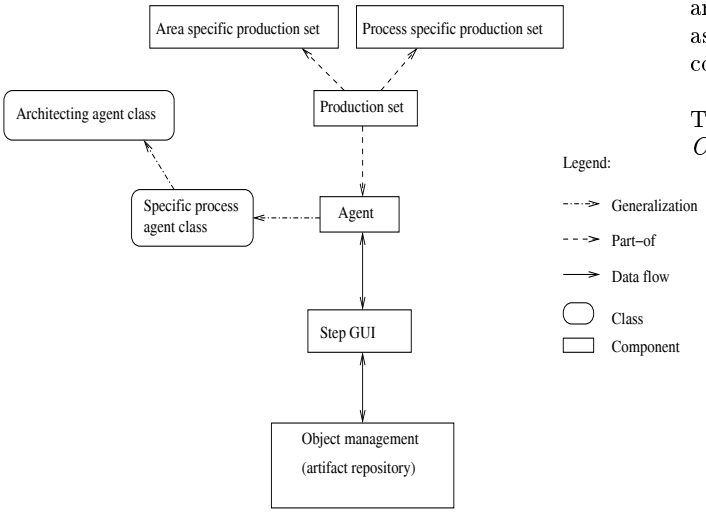

Figure 2: Agent structure

to be changed for the agent to work with a new architecting process defined in Little-JIL.

One such process is MapOnArchitecture, it takes source code of some software system as input and produces a set of the software system's elements and relationships between them which can be viewed as an architectural description. The process initially obtains the class diagram from the source code and then transforms the class diagram into an architectural description. This process' root step, *MapOnArchitecture*, is decomposed into four sequential steps: *IdClasses* - identification of classes, *IdRelationships* identification of relationships, *IdClassProperties* identification of class properties, and *CreateArchitecture* - mapping of the class diagram onto an architectural template, thus shifting from a class diagram to an architecture. A human assistant agent was assigned to *MapOnArchitecture* and *CreateArchitecture* steps, while automated agents performed *IdClasses*, *IdRelationships*, *IdClassProperties* steps.

The process programmer has to make a decision about the level of elaboration of a process. It is a trade-off between interpreted behavior and built-in precompiled behavior specification. In this case it is broken up between Little-JIL specifications and production specifications. Every automated step has a corresponding production that constructs output artifacts from the input ones. The human assistant agent provides the GUIs to assist the user with supervising the process execution and with the architecture mapping activity.

The activity that the human assistant agent performs on reception of an *CreateArchitecture* agenda item can serve as an example of a production. The human assistant agent at first executes that production and then opens a GUI that allows viewing the input artifact to the step (a class diagram showing the set of classes and relationships between them) and the first attempt at the output artifact (an architecture showing a run-time snapshot of the software system). This production specifies just one possible way of generating an architectural view from the class diagram. It is used more as a test case for the agent framework than a full-fledged, comprehensive architecture recovery process.

The production for one of the agents that can perform the *CreateArchitecture* step specifies that:

- every leaf class in the input artifact (class diagram) generates an instance of that class in the output artifact (architecture).
- every non-leaf class that is meant to be instantiated generates an instance of that class in the architecture (such a non-leaf class is likely to have other than generalization relationships to other classes in the class diagram, so this condition is used to guess which classes are to be instantiated).
- every non-generalization relationship is transferred from the classes in the class diagram to the corresponding instances in the architecture
- duplicate relationships are removed from the architecture

Other agents can have different productions for the same step. The user (human agent) responsible for the *CreateArchitecture* agenda item can edit the output of this production in the GUI provided by the human assistant agent.

4. RELATED WORK

Few SPEs pay any attention to agent specifications. Those SPEs that mention agents (or entities in agent roles: software development team member roles, tools) usually describe the skill sets needed from them and the activities they need to perform (e.g. APEL [3], DYNAMITE [6]). The AI multi-agent systems do focus on the agent frameworks for particular problem domains (e.g. [7], [10]). However, AI multi-agent systems do not pay attention to the needs of software process execution, primarily to the fact that a lot of agents in SPEs play the role of human assistants. There is also very little support for human agents (for instance, artifact-specific GUIs or provisions for development of human assistant agents) in the frameworks and environments for development of multi-agent systems ([1], [2], [5], [11], [12]). We believe that attending to the needs of human agents in multi-agent systems will enable a more effective use of the multi-agent approach for development of agent-based SPEs.

5. EVALUATION AND FUTURE WORK

The architecting process agent framework enables greater software reuse and accelerated specification of architecting processes. The use of productions made the process and agent description more comprehensive and more amenable to analysis. The use of XML for artifact representation enabled greater inter-operability with external tools and made their integration into the agents and process execution easier.

One of the future directions of the framework's improvement involves the generalization of artifact GUIs (as process-specific tools). There is a need for an automatic GUI generator based on artifact formats; it would greatly reduce the specification of new processes. It would be beneficial for the

GUI's to reflect the peculiarities of the languages used for artifact specification. Another direction is further generalization of production specifications, such as an XML-based language defining the transformation or generation of artifact elements.

The framework will be tested on more complex processes, requiring greater a level of communication between the agents (not just along the substep hierarchy or via resource manager). Part of this effort involves encoding agent communication processes in Little-JIL.

6. ACKNOWLEDGMENTS

The authors wish to thank Prof. Lee Osterweil and Sandy Wise for making available the Little-JIL/Juliette software process environment and providing further help with the environment.

7. REFERENCES

[1] K. S. Barber, A. Goel, D. C. Han, J. Kim, D. N. Lam, T. H. Liu, M. T. MacMahon, R. M. McKay, and C. E. Martin. Infrastructure for Design, Deployment and Experimentation of Distributed Agent-based Systems: The Requirements, the Technologies, and an Example. In *Autonomous Agents and Multi-Agent Systems (AAMAS-2001)*, 2001.

[2] F. Bellifemine, A. Poggi, and G. Rimassa. JADE - A FIPA-compliant agent framework. In *Proceedings of PAAM'99*, pages 97–108, London, April 1999.

[3] S. Dami, J. Estublier, and M. Amiuor. APEL: a Graphical Yet Executable Formalism for Process Modeling. Technical report, LSR, Actimart Bat 8, av. de Vignate, 38610 Gieres France, 1997.

[4] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. An Infrastructure for the Rapid Development of XML-based Architecture Description Languages. In *International Conference on Software Engineering 2002*, 2002.

[5] J. Graham and K. Decker. Towards a Distributed, Environment-Centered Agent Framework. In *The 1999 Intl. Workshop on Agent Theories, Architectures, and Languages [ATAL-99]*, Orlando, FL, July 1999.

[6] P. Heimann, G. Joeris, C.-A. Krapp, and B. Westfechtel. DYNAMITE: Dynamic task nets for software process management. In *The 18th International Conference on Software Engineering*, pages 331–341, March 1996.

[7] B. Horling, R. Vincent, R. Mailler, J. Shen, R. Becker, K. Rawlins, and V. Lesser. Distributed Sensor Network for Real Time Tracking. In *5th International Conference on Autonomous Agents*, pages 417–424, Montreal, June 2001.

[8] B. S. Lerner, L. J. Osterweil, J. Stanley M. Sutton, and A. Wise. Programming process coordination in Little-JIL. In V. Gruhn, editor, *Proceedings of the 6th European Workshop on Software Process Technology (EWSPT '98)*, number 1487 in Lecture Notes in Computer Science, pages 127–131, Weybridge, UK, September 1998. Springer-Verlag.

[9] D. E. Perry and A. L. Wolf. Foundations for the Study of Software Architecture. In *ACM SIGSOFT Software Engineering Notes*, pages 40–52, October 1992.

[10] V.Gorodetski, O.Karsaev, A.Khabalov, I.Kotenko, L.Popyack, and V.Skormin. Agent-based model of Computer Network Security System: A Case Study. In *International Workshop on Mathematical Methods, Models and Architectures for Computer Network Security*, volume 2052, pages 39–50. Lecture Notes in Computer Science, Springer Verlag, 2001.

[11] V.Gorodetski, O.Karsaev, I.Kotenko, and A.Khabalov. Software Development Kit for Multi-agent Systems Design and Implementation. *International Workshop of Central and Eastern Europe on Multi-agent Systems (CEEMAS-2001)*, September 2001.

[12] R. Vincent, B. Horling, and V. Lesser. An Agent Infrastructure to Build and Evaluate Multi-Agent Systems: The Java Agent Framework and Multi-Agent System Simulator. *Lecture Notes in Artificial Intelligence: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, 1887:102–127, 2001.

[13] A. Wise. Little-JIL 1.0 Language Report. Technical report 98-24, Department of Computer Science, University of Massachusetts at Amherst, 1998.