Gaia-PL: A Product Line Engineering Approach for Efficiently Designing Multi-Agent Systems

JOSH DEHLINGER
Towson University
and
ROBYN R. LUTZ
Iowa State University and Jet Propulsion Laboratory / Caltech

Agent-oriented software engineering (AOSE) has provided powerful and natural, high-level abstractions in which software developers can understand, model and develop complex, distributed systems. Yet, the realization of AOSE partially depends upon whether agent-based software systems can achieve reductions in development time and cost similar to other reuse-conscious development methods. Specifically, AOSE does not adequately address requirements specifications as reusable assets. Software product line engineering is a reuse technology that supports the systematic development of a set of similar software systems through understanding, controlling and managing their common, core characteristics and their differing variation points. In this article, we present an extension to the Gaia AOSE methodology, named Gaia-PL (Gaia – Product Line), for agent-based distributed software systems that enables requirements specifications to be easily reused. We show how our methodology uses a product line perspective to promote reuse in agent-based, software systems early in the development lifecycle so that software assets can be reused throughout system development and evolution. We also present results from an application to show how Gaia-PL provided reuse that reduced the design and development effort for a large, multi-agent system.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications - *Methodologies*; D.2.10 [**Software Engineering**]: Design - *Methodologies*; D.2.13 [**Software Engineering**]: Reusable Software – *Domain Engineering, Reuse Models*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Multiagent Systems*

General Terms: Design, Documentation

Additional Key Words and Phrases: Agent-oriented software engineering, software product line engineering

1. INTRODUCTION

Software reuse technologies have been a driving force in significantly reducing both the time and cost of software requirements specification, development, maintenance and evolution [Clements 2002; Clements and Northrop 2002; Pohl et al. 2005; Schmid and Verlage 2002; Weiss and Lai 1999]. Industry's ongoing demand for shorter software

This article is a revised, extended version of works presented at the 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS), St. Louis, MO, May 2005; Software Engineering for Multi-Agent Systems IV, Lecture Notes in Computer Science 3914, 2006; and the 24th International Conference on Software Maintenance (ICSM), Beijing, China, September 2008. This research was supported by National Science Foundation grants 0204139, 0205588 and 0541163; and by an Iowa Space Consortium grant.
Authors' addresses: Josh Dehlinger, Department of Computer and Information Sciences, Towson University, Towson, MD 21252; email: jdehlinger@towson.edu; Robyn Lutz, Department of Computer Science, Iowa State University, Ames, IA 50011; email: rlutz@cs.iastate.edu.

development cycles and lower software costs encourages software development methodologies to exploit software reuse principles whenever possible.

Software product line engineering (SPLE) is a mechanism for reuse [Jacobson et al. 1997] that supports the systematic development of a set of similar software systems through understanding, controlling and managing their common, core characteristics and their differing variation points [Clements and Northrop 2002; Pohl et al. 2005; Weiss and Lai 1999]. In a software product line, the common, managed sets of features shared by all members are the commonalities. The members of a product line may differ from each other via a set of allowed features not necessarily found in other members of the product line (i.e., the variabilities). The benefits of SPLE come from the reuse of the common features of the product line in the development of a new product line member. As a reuse technology, SPLE has the advantages of a lifecycle approach to reusing not just code but also requirements, architecture and test suites; broad usage in companies developing a variety of applications; and an active research community that supports transfer of innovations into practice.

Agent-oriented software engineering (AOSE) has provided powerful and natural, high-level abstractions in which software developers can understand, model and develop complex, distributed systems [Wooldridge et al. 2000; Zambonelli et al. 2003]. Yet, the realization of AOSE partially depends upon whether agent-based systems can achieve reductions in development time and cost similar to other reuse-conscious software development methods such as object-oriented design, service-oriented architectures and component based systems [Chan and Sterling 2003]. The work described here seeks to obtain the benefits of reuse for AOSE by incorporating SPLE techniques. In recent years, several AOSE methodologies have been proposed for various agent-based application domains. The Gaia methodology [Wooldridge et al. 2000; Zambonelli et al. 2003], in particular, offers a comprehensive analysis and design framework based on organizational abstractions by supplying schemas and models to capture the requirements of a multi-agent system (MAS).

In this article, we advocate for the inclusion of SPLE into AOSE to develop multi-agent system product lines (MAS-PL). To support this, this article presents an extension to the Gaia AOSE methodology, Gaia-PL (Gaia – Product Line), for agent-based, distributed software systems to capture requirements specifications that can be easily reused during the initial requirements phase as well as later if the software needs to be updated. The Gaia-PL methodology advances the state of the art in AOSE by providing a requirements specification pattern to capture the dynamically changing design configurations of agents and the potential reuse of the requirements specifications for future similar systems. The ability of the requirements specifications to accommodate the dynamically changing design configurations of an agent is important because an agent may need to adapt and reconfigure itself based on external conditions (e.g., environmental conditions, state of the MAS, changing goals, failures, etc.). This is achieved by adopting a SPLE approach for AOSE. Requirements specifications reuse is the ability to easily use previously defined requirements specifications from an earlier system and apply them to a new, slightly different system. This can significantly reduce the development time and cost of building an agent-based system.

Specifically, this article contributes the following results:

- Incorporates SPLE principles into the development of a MAS to build a MAS-PL. The notion of variation points in an agent of a MAS enables us to capture the differing protocols, activities, permissions and responsibilities specific to an agent's role.
- Extends the Gaia AOSE methodology to support the design of MAS-PL using aspects of Gaia, an established AOSE methodology, and FAST [Weiss and Lai 1999], an established SPLE methodology.
- Illustrates how our methodology, Gaia-PL, is amenable to the development of reusable software engineering assets during the design of a MAS-PL and how the reusable assets can be used to develop the systems (i.e., agents) of a MAS-PL. Application to a large MAS-PL demonstrated the ability to capture and reuse 36% of the requirements for 160 unique agents.
- Evaluates our Gaia-PL methodology's ability to reduce the design cost of a MAS via a case study and comparison to the Gaia methodology. A comparison of our methodology to the Gaia methodology showed an elimination of many redundant, scattered requirements and a 48% reduction in design and documentation time.

The work presented in this article builds on [Dehlinger and Lutz 2005; Dehlinger and Lutz 2006; Dehlinger and Lutz 2008] but lays out for the first time the technical approach in sufficient detail that it can be both used and evaluated by others. Sections 4.3 and 4.4 present new material regarding the information that needs to be captured in the Gaia-PL schemas and reuse of the schemas in defining the configuration of an agent, respectively. Section 5.2

presents new material to make the case for the advantages of Gaia-PL, and Section 5.3 for the first time evaluates the broader validity of the results.

The remainder of this article develops along the following lines. Section 2 discusses the related work in SPLE and AOSE. Section 3 gives an overview of the Prospecting Asteroid Mission system used throughout this article to illustrate and evaluate our methodology. Section 4 details our Gaia-PL methodology. Section 5 provides an evaluation of our methodology and compares the results of its application to the Gaia methodology. Finally, Section 6 offers some concluding remarks.

2. RELATED WORK

The Gaia-PL (Gaia-Product Line) methodology presented in this article provides a design and development methodology for efficiently building multi-agent systems that leverages the reuse inherent in a product line engineering approach. This section discusses the key concepts, techniques, methodologies and tools that are related to this work.

2.1 Software Product Line Engineering

Software product line engineering (SPLE) is a systematic approach for the design and development of software applications to create an array of similar products [Clements and Northrop 2002; Pohl et al. 2005]. SPLE creates a family of products based on an analysis of the commonalities and variabilities of the members of the family prior to the design or development of any software engineering artifacts [van Ommering 2005]. The goal of SPLE is to support the systematic development of a set of similar software systems through understanding, controlling and managing their common characteristics and their differing variation points [Clements and Northrop 2002; Pohl et al. 2005].

The benefits of SPLE come in the reuse of the common requirements of the product line during the development of a new product line member. The assets gained from the initial engineering of the product line (e.g., the underlying requirements, specifications, design and architecture) can be at least partially applied to any new product line member. In this sense, SPLE allows for the amortization of costs in startup development and analysis of the initial product line members over the development of the entire product line. In fact, studies suggest that the use of SPLE can reduce the development and production time as well as the overall cost and increase the product quality by a factor of 10 times or more [Schmid and Verlage 2002].

Requirements engineering in SPLE consists of identifying, refining and documenting the common and variable requirements of a proposed product line [Clements and Northrop 2002; Pohl et al. 2005]. Within SPLE, several requirements engineering approaches have been proposed including [Clements and Northrop 2002; Doerr 2002; Kang et al. 2002]. Similarly, agent-oriented software engineering work has provided mechanisms to assist in the requirements engineering process including [Castro et al. 2002; DeLoach 2004]. In this work we utilize a Commonality and Variability Analysis (CVA) [Ardis and Weiss 1997; Weiss and Lai 1999] as well as a feature model [Svanberg et al. 2005], as detailed in Section 4.2, as the instruments to aid in the requirements engineering process. To assist in this process, we utilize the DECIMAL tool [Dehlinger et al. 2007; Padmanabhan and Lutz 2005] to document and manage the requirements of the product line. Other existing SPLE requirements engineering tools that could have been used include those surveyed in [Beauche et al. 2007].

In this work, we follow Weiss and Lai's Family-Oriented Abstraction, Specification and Translation (FAST) approach for building product lines. The FAST approach is based on investing resources proactively [Weiss and Lai 1999]. The FAST approach partitions the design and development of a product line into two phases: *domain engineering* and *application engineering*.

Domain engineering is the phase in which the product line is defined by its commonalities, variabilities and dependencies [Clements and Northrop 2002; Pohl et al. 2005; Weiss and Lai 1999]. This is the investment phase that allows practitioners to, during the application engineering phase, realize a wide variety of products within the product line for a competitive advantage. A *commonality* is a feature that is the same in each member of a product line and contributes to the development of the core assets of the product line. A *variability* captures optional or alternative features not contained in every member of the product line and should describe the anticipated variations of the product line member over the "foreseeable lifetime of the product line" [Clements and Northrop 2002]. Finally, a *dependency* (i.e., constraint) restricts and/or dictates some combinations of variability subsets from being viable products in the form of "mutual exclusion" or "requires" variability dependencies [Doerr 2002]. Alternative approaches include the goal-oriented [Castro et al. 2002] and feature-oriented [Kang et al. 2002] approaches.

The *application engineering* phase is where members of the product line are developed by reusing the domain engineering assets and exploiting the product line's variability. The goal of the application engineering phase is to build individual product line members from the product line requirements specified during the domain engineering phase [Weiss and Lai 1999].

2.2 Agent-Oriented Software Engineering

Agent-oriented software engineering (AOSE) provides viable, high-level abstractions, models and approaches for designing and developing the autonomous agents of a multi-agent system to solve a problem [Zambonelli et al. 2003]. Wooldridge defines an *agent* as "an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives" [Wooldridge 1997]. Thus, a *multi-agent system* (MAS) is defined as a system "designed and developed in terms of autonomous software entities that can flexibly achieve their objectives by interacting with one another in terms of high-level protocols and terms" [Zambonelli et al. 2003]. A full discussion of the unique characteristics of agents and MAS is out of the scope of this article and can be found in [Jennings and Wooldridge 2000; Wooldridge 1997; Wooldridge et al. 2000; Zambonelli et al. 2003].

AOSE methodologies aim to provide tools and techniques for abstracting, modeling, analyzing and designing MAS early in the development lifecycle. A number of methodologies, such as Gaia [Cernuzzi et al. 2004; Wooldridge et al. 2000; Zambonelli et al. 2003], Tropos [Bresciani et al. 2004; Castro et al. 2002; Giorgini et al. 2004] and MaSE [DeLoach 2004], use different abstractions and models for MAS development.

2.2.1 The Gaia Methodology. In the work described in this article, the Gaia methodology was selected for the inclusion of software product line engineering (SPLE) concepts in building MAS for several reasons. First, it is a broad AOSE methodology spanning development from the initial analysis phase (that formulates the specifications for the collection of agents in a MAS) to the detailed design phase (that, based on the results of the analysis phase, focuses on the design and instantiation of individual agents of the MAS). Second, the Gaia methodology's development process and models are highly conducive to the design and development phases of SPLE. For example, Gaia's analysis and design phases have comparable objectives to SPLE's domain engineering phase when designing and developing a product line; similarly, Gaia's detailed design phase nicely corresponds to SPLE's application engineering phase. This is more fully described in Section 4. Third, Gaia is generic enough that the adaptation of SPLE for building MAS in it could be similarly included in other AOSE methodologies.

Exemplifying this is the development of the MaCMAS methodology [Peña et al. 2006a; Peña et al. 2006b], described in Section 2.2.3, the extension of PASSI to include SPLE [Nunes et al. 2009b], described in Section 2.2.4, and a domain engineering process for designing and developing MAS with SPLE [Nunes et al. 2009a]. Finally, Gaia is an established, well-documented and widely accepted methodology in the AOSE community.

The Gaia methodology centers on defining an agent based upon the role(s) that it can assume during its lifetime [Cernuzzi et al. 2004; Wooldridge et al. 2000; Zambonelli et al. 2003]. Each role's requirements specification is defined by its protocols (i.e., how agents interact), activities (i.e., the computations associated with the role that can be executed without interacting with other agents), permissions (i.e., the information resources that the role can read, change and generate) and responsibilities (i.e., the liveness and safety properties the role must ensure). The Gaia methodology adopts a computational organizational metaphor where each agent within a MAS may play a variety of roles and where the agents cooperate with each other to accomplish a common, organization-wide goal. The analysis phase of the development of a MAS concentrates on specifying the requirements for the roles in which an agent may participate during its lifetime in a set of Role Schemas. It is primarily this phase of the Gaia methodology that we extend in this article to include a SPLE approach.

The extension of Gaia described here, called Gaia-PL (Gaia – Product Line) differs from Gaia in that we integrate SPLE into the Gaia methodology, thus enabling AOSE to capture the reuse potential of a MAS's software engineering assets to build multi-agent system product lines (MAS-PL). Further, Gaia-PL focuses on capturing the reusability of the software engineering assets developed during the design and development of a MAS-PL using a SPLE approach so that future agents of a MAS-PL can be built more quickly and cheaply.

The Gaia methodology has three limitations that we address. First, although Gaia allows the role of an agent to change dynamically, it is unclear how to document agent requirements specifications when an agent must be updated to include new functionality during the analysis and design phases [Dehlinger and Lutz 2005]. Second, the design of an agent in Gaia is not hierarchical [Juan and Sterling 2002]. That is, the roles of an agent are coarsely defined, allowing little flexibility for similar, yet slightly different behavior in the same role in different agents. This limits the opportunity for reuse. Third, the Gaia methodology fails to provide a mechanism by which the requirements specification templates developed during the analysis phase can be reused and incorporated into the

Table I. Overview of Key Differences between the Gaia and Gaia-PL methodologies

Gaia	**Gaia-PL**
Constructs single-use assets	Constructs reusable assets for MAS-PL
No distinction between common and variable requirements, leading to more duplication	Identifies common requirements and reuses those schemas
Each variation point requires a new role, leading to more role schemas	Represents variations in reusable variation point schemas, leading to fewer schemas overall
Duplication of functionality in roles	Avoids unnecessary duplication
No support for an agent's ability to change from one set of functionalities of a role to another	Supports agent's ability to change from one set of functionalities of a role to another
Non-hierarchical approach does not provide association among related roles	Hierarchical approach links related roles
Less efficient design with no opportunity for reuse in MAS-PL case study	More efficient design with a high degree of reuse in MAS-PL case study

current system or to build a new, similar but slightly different system [Dehlinger and Lutz 2006; Peña et al. 2006]. Since Gaia does not provide a way to describe variations, even very similar roles have to be totally re-created.

To address these limitations, Gaia-PL introduces variation points into the design and development of MAS. Variation points are used in SPLE to capture the allowed differences amongst members belonging to the same product family. For Gaia-PL, we define the variation points for a specific role of an agent as the *differing* protocols, activities, permissions and responsibilities available to that role. Variation points typically derive from the grouping of the product line variabilities defined in a Commonality and Variability Analysis. The introduction of variation points in Gaia-PL addresses the limitations of Gaia by allowing the software engineer to define a role with greater flexibility and partition some functionality to better respond to the agent and system's current configuration. The key differences between the Gaia methodology and our Gaia-PL methodology are presented in Table 1.

2.2.2 Reuse-Oriented Methodologies. From its earliest days, one of the goals of AOSE has been to provide methodologies for reusing and maintaining agent-based software systems [Tveit 2000]. Despite this goal, AOSE methodologies have failed to adequately capture the reuse potential since many of the developed methodologies center on the development of specific software applications. A few authors have proposed reuse in an agent-oriented development environment. For example, in [Giorgini et al. 2004], asset reuse occurs during the design phase of a MAS. Likewise, [Hara et al. 2000] reuses components from a previously developed agent-based component repository. However, these previous works delay consideration of reuse until design or implementation rather than applying it at an early design stage, as is done here. Further, the work described here differs from previous work in that we present an approach, based on software product line engineering, to capture the reuse potential of distributed, agent-based software systems in the requirements analysis, design and specification stage.

2.2.3 The MaCMAS Methodology. Peña et al. described a methodology for analyzing Complex Multiagent Systems (MaCMAS) using an SPLE approach to build MAS-PL [Peña et al. 2006a; Peña et al. 2006b]. MaCMAS uses the UML to model a MAS-PL and focuses on handling the complexity of MAS-PL and building its core architecture.

The MaCMAS methodology, like the work described in this article, utilizes a feature model to document the commonalities and variabilities of the MAS-PL. Unlike the work reported here, however, the MaCMAS methodology uses an algorithm to analyze the features (i.e., commonality and variability requirements) of a MAS-PL in order to partition the requirements into either commonalities or variabilities based on the probability that a feature will appear in a product. This information is then used to determine which features should be included, using their approach, in the MAS-PL's core architecture [Peña et al. 2006b].

The MaCMAS methodology uses and extends our incorporation of SPLE techniques into AOSE, originally reported in [Dehlinger and Lutz 2005; Dehlinger and Lutz 2006]. Unlike the MaCMAS methodology, the work presented here extends an established, well-known AOSE methodology, Gaia [Wooldridge et al. 2000; Zambonelli et al. 2003], by introducing SPLE concepts from an established, well-known SPLE approach, FAST [Weiss and Lai 1999]. Further, the work presented in this article differs from that of MaCMAS in that we focus on the reusability of the MAS-PL's requirements and requirements specifications rather than on the architecture.

2.2.4 Other MAS-PL Methodologies. More recently, Nunes et al. have explored additional approaches to developing and evolving MAS-PLs. In [Nunes et al. 2008], a feature model was used in an SPLE manner to assist in organizing the introduction of new variable agency features (i.e., product line variabilities) and guiding the refactoring of a web-based system's architecture. Unlike the work presented in this article, Nunes et al. viewed an agent-based, evolutionary system as a MAS-PL and focused on the development and evolution of the system rather than on the initial design, analysis and specification, as is done here.

In [Nunes et al. 2009], a MAS-PL methodology was proposed extending the AOSE PASSI methodology [Cossentino 2005] to include support for incorporating product line variabilities. The PASSI methodology, unlike Gaia and Gaia-PL, provided models that cover the entire development process from requirements to implementation. In Nunes et al.'s work, however, the focus was on the domain analysis phase (i.e., developing and analyzing the requirements and specifications to produce a System Requirements Model) and the extension of PASSI's UML models via stereotypes to include SPLE concepts [Nunes et al. 2009] rather than on the application engineering of MAS-PLs.

3. THE PROSPECTING ASTEROID MISSION

To illustrate, motivate and evaluate the work presented in this article, we use requirements based on the Prospecting Asteroid Mission (PAM). PAM is a proposed NASA concept mission lasting 5-10 years based on the Autonomous Nano-Technology Swarm (ANTS) technology to explore the asteroid belt between Mars and Jupiter [Peña et al. 2006b; Rouff et al. 2005; Sterritt et al. 2005; Truszkowski et al. 2004; Truszkowski et al. 2006].

The ANTS technology is a system architecture for scalable, robust and highly distributed systems and has been proposed to be used in a *family* of flight-based and ground-based, NASA-proposed missions (each with differing objectives and goals) to explore our solar system. The ANTS architecture will be based on autonomous, self-addressable, self-configuring components that will consist of common subsystems that all spacecraft must have (e.g., inter-spacecraft communication components, guidance and navigation components, etc.) as well as specialized components for a small subset of spacecraft. Further, the autonomy required by ANTS-based spacecraft will require each spacecraft to have the ability to be self-configuring, self-healing, self-optimizing and self protecting. While, these behaviors will have many similarities (i.e., common requirements) across all ANTS-based missions (e.g., all ANTS-based spacecraft will be self-protecting by avoiding collisions with other spacecraft), each property will require specialized requirements depending on the specific mission [Sterritt et al. 2005]. Thus, the reuse reported in this work specific to the PAM mission could be expanded to the entire ANTS-based family of missions, and the ANTS-based spacecraft themselves could be explored as a multi-agent system product line (MAS-PL), as was done in [Peña et al. 2006b].

The proposed PAM consists of up to 1,000 pico-spacecraft (spacecraft weighing less than one kilogram) that can autonomously form subswarms to investigate asteroids of interest in the asteroid belt. Except for a spacecraft's scientific instrumentation specialties, each PAM spacecraft has identical hardware.

Each PAM spacecraft is designated as a *leader*, a *messenger* or a *worker* [Rouff et al. 2005; Sterritt et al. 2005; Truszkowski et al. 2004; Truszkowski et al. 2006]. A spacecraft designated as having a *Leader* role determines the types of asteroids and data the mission should pursue and coordinates the efforts of *Worker* spacecraft to investigate asteroids in order to satisfy mission objectives. A spacecraft designated as having a *Messenger* role coordinates communication between *Worker* spacecraft, the *Leader* spacecraft and the Earth. *Worker* spacecraft each contain a single scientific instrument and perform scientific investigation using its specialized equipment (e.g., spectrometers, altimeters, magnetometers, infrared radiometers, etc.).

Of the 1,000 PAM spacecraft, approximately 80% are *Worker* spacecraft with the remaining 20% equally divided between *Leader* and *Messenger* spacecraft. Thus, for each type of spacecraft there is significant redundancy since NASA projects that 60%-70% of the PAM spacecraft could be lost over the duration of the mission due to failures, collisions, etc. To preserve mission-critical requirements (e.g., the swarm's ability to pursue scientific goals and report their findings), additional capabilities are given to some spacecraft to achieve redundancy at the swarm level. For example, some spacecraft may be able to switch from a *Leader* role to a *Messenger* role or vice versa in response to the loss or failure of some spacecraft.

We claim that viewing the PAM spacecraft as a MAS-PL has several advantages. From a product line engineering perspective, the similarities in requirements among every spacecraft of the PAM swarm (e.g., the navigation and guidance capabilities, collision avoidance, etc.) are product line commonality requirements. Similarly, the differences amongst the spacecraft of the PAM swarm (e.g., the differing requirements between *Leader*, *Messenger* and *Worker* spacecraft, the ability of some *Messenger* spacecraft to be upgraded to a *Leader* spacecraft, etc.) are usefully viewed as product line variability requirements.

The characteristics of the proposed PAM swarm present significant challenges to existing multi-agent system design and development including:

- *Size of the design space*. The high degree of allowed variability (64 high-level variability requirements in our PAM study) together with the core functionality (35 high-level commonality requirements in our PAM study) presents a large design space in which a large number of unique PAM spacecraft configurations are viable (160 unique configurations in our PAM study).

- *High degree of reuse*. A significant portion of the features of the PAM spacecraft (approximately one-third of the total requirements) are commonalities that will be reused on all spacecraft
- *Hierarchical and dynamically changing roles.* The hierarchical nature of the roles and variation points, described in Section 4, depend on context (e.g., environment, failures, current goal, etc.) and must autonomously address MAS-PL goals, evolution and maintenance needs.

These challenges motivated the work presented here. The remainder of this article shows how Gaia-PL can handle these challenges and uses the PAM MAS-PL to illustrate and evaluate our proposed methodology. Future software applications will increasingly require specialized, autonomous software agents to be designed and developed quickly and inexpensively. Gaia-PL offers an approach to accommodate these needs for highly redundant multi-agent systems.

4. THE GAIA-PL METHODOOGY

The Gaia-PL methodology provides a requirements specification pattern to capture the dynamically changing design configurations of agents and reuse the requirements specifications for future similar systems. This is achieved by representing the dynamically changing design configurations of agents as product line variation points and reusing them as new systems are built. We first describe the use of variation points as a mechanism to capture and reuse the variations in the behavior of an agent's role and facilitate reuse.

4.1 Using Variation Points in Agent-Oriented Software Engineering

In previous work [Dehlinger and Lutz 2005; Dehlinger and Lutz 2006; Dehlinger and Lutz 2008], we have shown that an important way to classify variation points for an agent of a multi-agent system product line (MAS-PL) is based on the varying *intelligence* levels for a specific role. In another NASA-proposed satellite constellation [Chien et al. 2002; Schetter et al. 2000], variation points for a role were ordered in terms of increasing intelligence levels and defined as follows:

- **I4**: the role is able to receive and execute commands

- **I3**: the role is able to participate in local planning activities pertinent to the role as well as receive and execute commands
- **I2**: the role is able participate in local planning and interaction activities pertinent to the role, contains partial cluster-knowledge related to the role's objective as well as receive and execute commands
- **I1**: the role is able participate in cluster-level planning and interaction activities pertinent to the role, contains full cluster-knowledge related to the role's objective as well as receive and execute commands

In this example, as a role in an agent is promoted to a higher intelligence level (e.g., from I3 to I2) the configuration of the agent dynamically changes by incorporating additional protocols, activities, permissions and/or responsibilities. The opposite occurs when a role is demoted from a higher intelligence level to a lower intelligence level (e.g., from I2 to I3). Using this construct, *an agent's role may have one or more variation points*. Variation points are particular to each application and, indeed, particular to each role. However, the intelligence level variation point of this example will not be universal to all MAS-PL. Other variation points found in other multi-agent systems (MAS) include active, passive; hot-spare, warm-spare, cold-spare; etc.

The variation points of an agent are initially fixed upon deployment of the MAS-PL based upon the software and hardware facilities available to the agent as well as the role's goal. At deployment, a default variation point for each role is set. During execution, a role may dynamically change a variation point based upon its internal state or commands from external sources. However, it is not likely that the same set of variation points will be included in any given role throughout the entire MAS [Dehlinger and Lutz 2005; Dehlinger and Lutz 2006]. Thus, from a software product line engineering (SPLE) perspective, we can view the set of roles containing different role/variation point combinations as a product line. The set of roles and dynamic variation points that an agent has defines its *configuration*.

4.2 Identifying Features, Roles and Variation Points of a MAS-PL

Figure 1 shows the process and agent-oriented software engineering (AOSE) artifacts generated in each phase. This figure illustrates the Gaia-PL methodology in the context of the phases of Gaia [Wooldridge et al. 2000; Zambonelli et al. 2003] (i.e., Collection of Requirements, Analysis and Design, and the Detailed Design) and the Family-Oriented Abstraction, Specification and Translation (FAST) [Weiss and Lai 1999] SPLE approach.

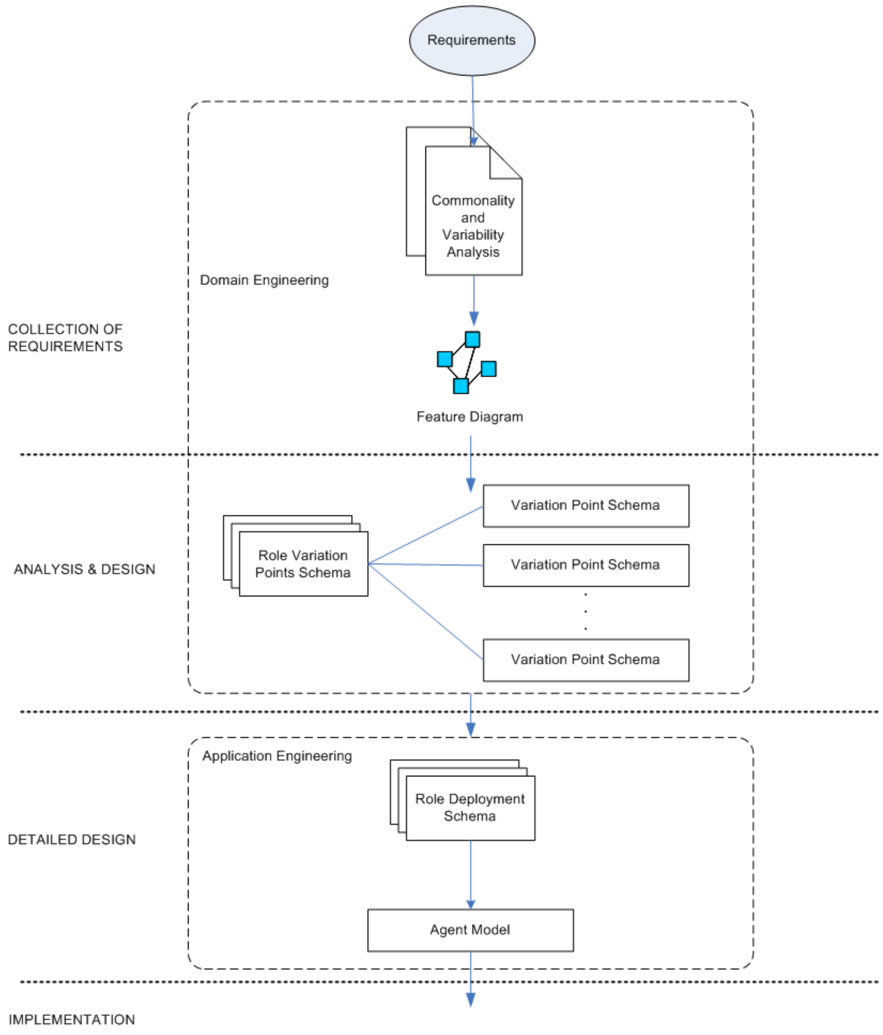


Fig. 1. An Overview of the Software Engineering Artifacts of Gaia-PL.

4.2.1 Documenting MAS-PL Requirements. The Collection of Requirements phase involves identifying and documenting the MAS-PL's commonality and variability requirements in a Commonality and Variability Analysis (CVA). This phase corresponds with the domain engineering activities of SPLE.

In the Prospecting Asteroid Mission (PAM) study, our CVA identified and documented a total of 35 high-level commonality requirements and 62 variability requirements. From the CVA's variabilities, a Parameters of Variation table was derived to better define the variabilities listed in the CVA [Ardis and Weiss 1997; Weiss and Lai 1999]. The Parameters of Variation tables lists the parameter's name, the associated variability requirement (for traceability), a description of the parameter, the domain of the possible values of the parameter, and the binding time at which the configuration of the parameter must be selected. Additionally, 48 parameters of variation were defined from the 62 variability requirements. Note that several product line variabilities can constitute a single parameter of variation.

Included within the CVA are the dependencies (i.e., constraints) among the variabilities arising from design decisions, safety implications, resource constraints, weight considerations, etc. For a large MAS-PL, such as the PAM swarm, with many constraints, automated CVA tool-support is important to ensure efficient management and verification in the selection of variability requirements for an agent of the MAS-PL to enable and support reuse. To support large, complex MAS-PL and facilitate requirements reuse, we utilize DECIMAL [Dehlinger et al. 2007; Padmanabhan and Lutz 2005], a product line requirements engineering and management and verification tool.

The CVA and the requirements documented in DECIMAL guide the definition of the roles and of the variation points possible in each role. DECIMAL can also facilitate creation of a feature model to identify roles and a role's variation points and aids in structuring the requirements specifications schemas (Section 4.3) that will assist in requirement reuse (Section 4.5).

4.2.2 Identifying MAS-PL Roles and Variation Points. A feature model hierarchically defines the mandatory, optional and alternative features of a product line by breaking down a single, high-level feature into its subfeatures [Svanberg et al. 2005]. A new product line member consists of the mandatory (i.e., common) features, a selection amongst the alternative features and the desired, optional features. A child feature in the feature model can only be present if its parent feature is also present.

Figure 2 shows a portion of the PAM feature model constructed from the MAS-PL requirements documented in the CVA using DECIMAL. The feature model can greatly facilitate the identification of the roles of the MAS-PL as well as the variation points. The following heuristics were found to support identification of the roles and variation points:

- Any feature that has mandatory and optional subfeatures is a candidate role with its subfeatures as candidate variation points (e.g., in Figure 2, the feature "Warn of Solar Storms" includes the requirements of a candidate role)
- Any feature that has an "Only One" or "At Least One" cardinality is a candidate role with its subfeatures as candidate variation points

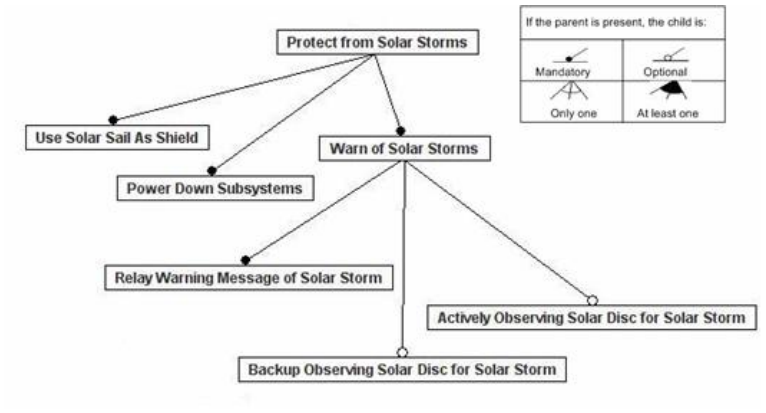

If the parent is present, the child is:
Mandatory
Optional
Only one
At least one
Protect from Solar Storms
Use Solar Sail As Shield
Power Down Subsystems
Warn of Solar Storms
Relay Warning Message of Solar Storm
Actively Observing Solar Disc for Solar Storm
Backup Observing Solar Disc for Solar Storm

Fig. 2. Partial Feature Model for the Prospecting Asteroid MAS-PL Study.

- Any mandatory feature that has no children but has a sibling with children that match one of the above rules is either a candidate role, with no variation points, or should include its functionality within one of its siblings (e.g., in Figure 2, the functionality of the feature "Use Solar Sail as Shield" could be consolidated within the "Warn of Solar Storms" feature as a role or become its own role)
- For any feature labeled as a candidate role using the above rules, consider consolidating its functionality with its parent feature's functionality to constitute a role, along with the already identified candidate variation points

The heuristics are informal, step-by-step rules that guide the identification of candidate roles and variation points from the feature model. They were developed during our experience with the large application. They are general in that they can be used to hierarchically derive candidate roles and variation points from any feature model. These heuristics were found to provide sufficient guidance to identify the roles and variation points for the PAM swarm and structure the requirements specifications schemas for the MAS-PL in a hierarchical manner.

In addition to defining the variation points of a role for a PAM spacecraft based on the type of spacecraft (described in Section 3), Gaia-PL identified other variation points to be defined for other roles. For example, a *Leader* spacecraft of the PAM swarm will have a role called *LeaderPlanner* that is tasked with managing, planning and coordinating the spacecraft of a PAM subswarm. For this role, the variation points include:

- **Passive:** Acts as a backup to verify/double-check the commands and calculations of a spacecraft with a *LeaderPlanner* role acting with the "active" variation point

- **Active:** Able to command the spacecraft; can request from "passive" *LeaderPlanner*s verification/agreement on its calculated strategy

A *Leader* spacecraft's *LeaderPlanner* role will be configured as either passive only or as both *passive* and *act*ive. A *LeaderPlanner* role configured with both the passive and active variation points may only assume one of the variation points at a time. However, not every role that can be defined for an agent will necessarily have variation points. For example, the *Navigator* role, tasked with the functionality to maneuver itself in space using its solar sail has no variation points so its behavior is identical regardless of the type of spacecraft.

For every variation point identified, a binding time is associated to it which defines the time at which the variation point could be assumed by a role. Potential binding times include design-time, specification-time, configuration-time and run-time. In the case of the PAM, most of the binding times were at design-time. For the *LeaderPlanner* role, however, the binding time is determined in two stages. The decision for whether a spacecraft with the *LeaderPlanner* role should have only the *passive* variation point or both the *passive* and *active* variation point must be done at design time. Subsequently, for those *LeaderPlanner* roles that have both the *passive* and *active* variation points, the switch from *passive* to *active* or vice versa, based on its own decision or on a command received, is done at runtime.

In our application of the Gaia-PL methodology to the PAM requirements, we found several roles where the binding time of a role's variation point occurred in two stages, as in the *LeaderPlanner* example described above. This is an important characteristic and a challenge to developers for the many MAS that need to be autonomous and adapt to the changing situation and environment [Luck et al. 2004]. Dynamic (i.e., runtime) reconfiguration is essential for agents to accommodate situatedness and proactively adapt to their environment [Luck et al. 2004; Zambonelli et al. 2003]. This necessitates a solution that allows the possible configurations of the role to be specified at design time and the agent's role to change its variation point(s) during execution. For agents that have roles that may dynamically change functionality during their lifetime, the ability to partition a role's varying functionality via its variation points allows the designer to specify the possible configurations of the role at an early binding time. Then, that role can autonomously assume a particular variation point of the role during runtime. Thus, the variation points provide a mechanism to capture the functionality of a role within an agent that may dynamically change during execution.

By partitioning the role of an agent into its common and variable parts in this manner, Gaia-PL provides the ability to define a role hierarchically. Using this approach, the common functionality of a role is first captured and then the variable functionality is captured as variation points at a level lower.

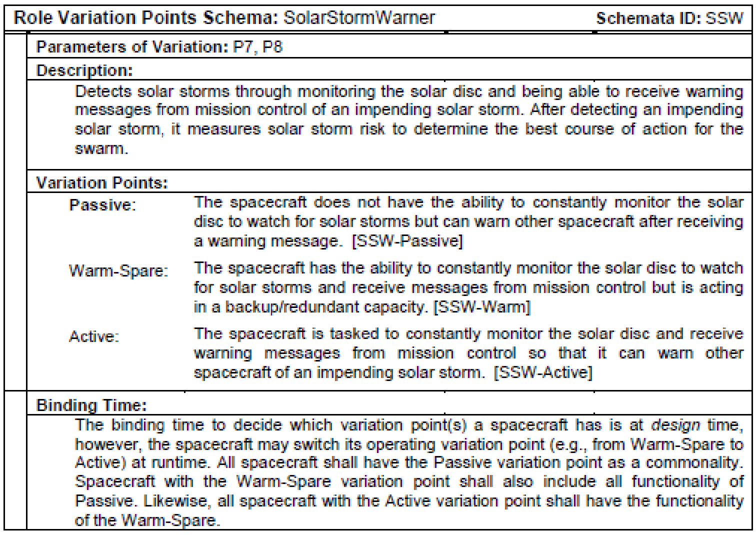
4.3 Documenting the Requirements Specifications of a MAS-PL

The Analysis and Design phase of Gaia-PL, shown in Figure 1, takes the requirements and features documented in the Collection of Requirements phase and develops and documents the MAS-PL's requirements specifications in schemas. The schemas, adapted from Gaia [Wooldridge et al. 2000; Zambonelli et al. 2003], provide a structured requirements specification pattern to document MAS-PL requirements and allow for reuse. In Gaia-PL, requirements specifications are documented using three schemas: The Role Schema, The Role Variation Points Schema and The Variation Point Schema. This section describes the development and documentation of the roles and variation points for the PAM from the requirements and features discussed in the previous section. Note that the complete set of schemas documenting PAM mission's requirements specifications can be found in [Dehlinger 2007].

4.3.1 Role Schema. For those roles that have been identified as having no variation points (i.e., the role will have identical functionality in all agents that have the role), Gaia-PL uses a slightly modified version of Gaia's Role Schema [Wooldridge et al. 2000; Zambonelli et al. 2003]. For example, the *Navigator* role of the PAM mission, mentioned in Section 4.2.2, has no variation points and thus can be documented in a Role Schema.

The only difference is that Gaia-PL includes the following additional information in the requirements specifications schemas:

- Identification numbers for all schemas for traceability, organization and management purposes. A row is added to indicate specifically which variation point the requirements specification is detailing.
- An "Inherits" field describes which schemas must be included with the schema for a particular variation point, further described in Sections 4.3.2 and 4.3.3.
- Parameters of Variation and Requirements fields related to the schema are included to better enable MAS-PL traceability, organization and management.



Role Variation Points Schema: SolarStormWarner **Schemata ID:** SSW

Parameters of Variation: P7, P8

Description:

Detects solar storms through monitoring the solar disc and being able to receive warning messages from mission control of an impending solar storm. After detecting an impending solar storm, it measures solar storm risk to determine the best course of action for the swarm.

Variation Points:

Passive:	The spacecraft does not have the ability to constantly monitor the solar disc to watch for solar storms but can warn other spacecraft after receiving a warning message. [SSW-Passive]
Warm-Spare:	The spacecraft has the ability to constantly monitor the solar disc to watch for solar storms and receive messages from mission control but is acting in a backup/redundant capacity. [SSW-Warm]
Active:	The spacecraft is tasked to constantly monitor the solar disc and receive warning messages from mission control so that it can warn other spacecraft of an impending solar storm. [SSW-Active]

Binding Time:

The binding time to decide which variation point(s) a spacecraft has is at *design* time, however, the spacecraft may switch its operating variation point (e.g., from Warm-Spare to Active) at runtime. All spacecraft shall have the Passive variation point as a commonality. Spacecraft with the Warm-Spare variation point shall also include all functionality of Passive. Likewise, all spacecraft with the Active variation point shall have the functionality of the Warm-Spare.

Fig. 3. Role Variation Points Schema in Gaia-PL for the *SolarStormWarner* Role.

4.3.2 Role Variation Points Schema. The Role Variation Points Schema defines a role and the variation points that the role can assume during its lifetime. Figure 3 shows the Role Variation Points Schema for the *SolarStormWarner* (SSW) role discussed in Section 4.2.2 and shown in a feature model in Figure 2. The Role Variation Points Schema describes the role, the role's variation points and the binding time for the variation points. This is an improvement on Gaia's approach [Wooldridge et al. 2000; Zambonelli et al. 2003] in that it allows for the partitioning of a role's differing functionality to better enable reuse. The variation points are described for the role and provide the identification tags (e.g., SSW-Passive, SSW-Warm, SSW-Active) for the Variation Points Schema, discussed in the next section. For most roles, one of the variation points listed in the Role Variation Points Schema will contain the common functionality of the role, denoted by being in bold. This variation point will be included for all agents containing the role. The common functionality defined by a variation point is further refined by the variable variation points. The hierarchical nature of the functionality in a role as modeled by the feature model, shown in Figure 2, is traced forward to the Role Variation Points Schema.

4.3.3 Variation Point Schema. The Variation Point Schema, shown in Figures 4 - 6 for the variation points of the *SolarStormWarner* role, captures the required capabilities of a role variation point's functional behavior. The Variation Point Schema and the Role Schema are identical; however, the Variation Point Schema will always have a Role Variation Points Schema associated with it (shown in the Schema-ID using the

Variation Point Schema: SolarStormWarner **Schema ID:** SSW-Passive

Variation Point: Passive

Inherits: SP-Core

Parameters of Variation: P7=Passive; P8=False

Requirements: C_G1, C_SH4, C_SP5, C_SP8, V_SP1, V_SP2

Description:
Receives warnings from other spacecraft about impending solar storms and calculates the risk factor to itself from solar radiation damage. Notifies other nearby spacecraft of the impending solar storm.

Activities and Protocols:
CalculateStormRisk, UpgradeToWarm, AcceptUpgrade, AcceptWarnMsg, RecieveHeartbeat, ReplyHeartBeat, SendSolarStormWarnMsg

Permissions:

Reads -

position	// current spacecraft position
velocityIncrement	// current spacecraft velocity increment
curScienceGoalFactor	// current spacecraft scientific goal factor
subswarmVector	// vector of nearby spacecraft to warn
supplied *stormType*	// type of storm supplied by warning
supplied *stormIntensity*	// storm intensity supplied by warning
supplied *stormVector*	// storm vector supplied by warning

Changes -

riskForSystemFactor	// current risk to spacecraft

Generates -

stormRiskValue	// new value of the risk to the spacecraft of // the solar storm

Responsibilities:

Liveness -
Optimize the ability to satisfy scientific goals while minimizing the risk factor.

Safety -
Prevent other spacecraft from being damaged by notifying others.

Fig. 4. Variation Point Schema in Gaia-PL for the *SolarStormWarner* Role's Passive Variation Point.

Variation Point Schema: SolarStormWarner **Schema ID:** SSW-Warm

Variation Point: Warm-Spare

Inherits: SSW-Passive

Parameters of Variation: P7=Warm-Spare; P8=False

Requirements: V_SP1, V_SP2

Description:
Acts as a redundant backup to those spacecraft that are actively monitoring the solar disc and warning other spacecraft of impending solar storms that may damage their onboard equipment. With actively monitoring spacecraft, verifies measurements and other solar storm measurements.

Activities and Protocols:
CalculateStormDataAccuracy, CompareVerifyStromData, DetectStormData, DowngradeToPassive, ObserveSolarDisc, UpgradeToActive, AcceptStormData, AcceptDowngrade, AcceptUpgrade, SendHeartbeat, SendStormData, VoteStormDataAccuracy

Permissions:

Reads -

supplied *prelimStormType*	// preliminary type of storm supplied by // active spacecraft to be verified
supplied *prelimstormIntensity*	// preliminary intensity of storm supplied by // active spacecraft to be verified
supplied prelim*stormVector*	// preliminary storm vector supplied by // active spacecraft to be verified

Changes -

stormDataAccuracyValue	// current value of the accuracy of the // supplied data compared to detected data
stormRiskValue	// current risk value of the storm to the // spacecraft

Generates -

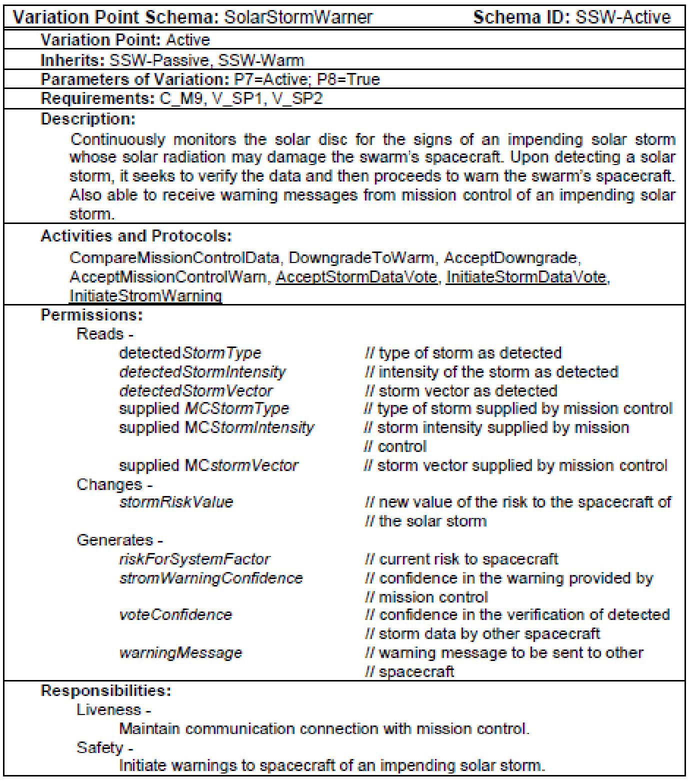
detected*StormType*	// type of storm as detected
detectedStormIntensity	// intensity of the storm as detected
detectedStormVector	// storm vector as detected

Responsibilities:

Liveness -
Maintain heartbeat with other spacecraft monitoring the solar disc.

Safety -
Prevent dissemination of false solar storm warnings.

Fig. 5. Variation Point Schema in Gaia-PL for the *SolarStormWarner* Role's Warm-Spare Variation Point.



Variation Point Schema: SolarStormWarner **Schema ID:** SSW-Active

Variation Point: Active

Inherits: SSW-Passive, SSW-Warm

Parameters of Variation: P7=Active; P8=True

Requirements: C_M9, V_SP1, V_SP2

Description:

Continuously monitors the solar disc for the signs of an impending solar storm whose solar radiation may damage the swarm's spacecraft. Upon detecting a solar storm, it seeks to verify the data and then proceeds to warn the swarm's spacecraft. Also able to receive warning messages from mission control of an impending solar storm.

Activities and Protocols:

CompareMissionControlData, DowngradeToWarm, AcceptDowngrade, AcceptMissionControlWarn, AcceptStormDataVote, InitiateStormDataVote, InitiateStromWarning

Permissions:

Reads -

detected*StormType*	// type of storm as detected
detectedStormIntensity	// intensity of the storm as detected
detectedStormVector	// storm vector as detected
supplied *MCStormType*	// type of storm supplied by mission control
supplied MC*StormIntensity*	// storm intensity supplied by mission // control
supplied MC*stormVector*	// storm vector supplied by mission control

Changes -

stormRiskValue	// new value of the risk to the spacecraft of // the solar storm

Generates -

riskForSystemFactor	// current risk to spacecraft
stromWarningConfidence	// confidence in the warning provided by // mission control
voteConfidence	// confidence in the verification of detected // storm data by other spacecraft
warningMessage	// warning message to be sent to other // spacecraft

Responsibilities:

Liveness -

Maintain communication connection with mission control.

Safety -

Initiate warnings to spacecraft of an impending solar storm.

Fig. 6. Variation Point Schema in Gaia-PL for the *SolarStormWarner* Role.

convention of *Role Variation Points Schema ID – Variation Point ID*). Some variation points will inherit other variation points, as illustrated in the Inherits row. For example, the Variation Point Schema in Figure 5 denotes that it inherits the SSW-Passive variation point, shown in Figure 4, since the SO-Passive Variation Point Schema provides the common functionality of the *SolarStormWarner* role.

4.3.4 Documenting Roles and Variation Points. To capture the requirements specifications of the roles and variation points of a MAS-PL and document them in the two schemas, we use the following procedure:

1. Identify the roles within the system, discussed in Section 4.2.2. Each role will constitute a new Role Variation Points Schema. If the role has no variation points (see Step 3), then create a new Role Schema and follow Steps 4a – 4c.
2. For each role, provide the role's name, a unique identification, a listing of the associated parameters of variation, a brief description of the role and the variation points' binding time in the appropriate fields of the Role Variation Points Schema. We follow the numbering scheme of [Schetter et al. 2000] as shown in Figure 3.

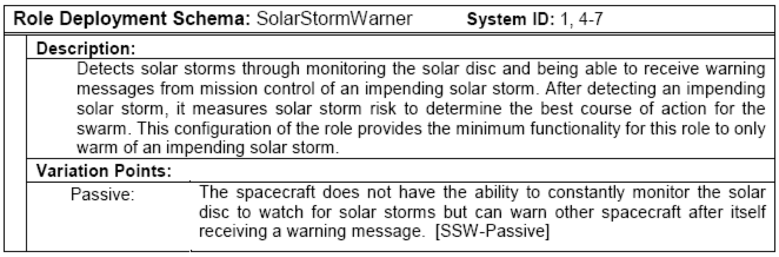
3. For each role, identify and define the differing variation points that the role can adopt during all envisioned execution scenarios of the system as described in Section 4.2.2. For each variation point, fill in the Variation Points section of the Role Variation Points Schema by including the name, a brief description of the variation point and a reference identification number to the Role Variation Points Schema that gives the detailed requirements of the variation point (see Step 4a).
4. For each identified variation point (Step 3), create a new Variation Point Schema. For each Variation Point Schema:
   a. Document the name of the role to which the variation point corresponds as well as the name of the variation points in the appropriate sections of the Variation Point Schema. Indicate the variation point identification tag (corresponding to the variation point identification of Step 3) in the appropriate field in the Role Variation Points Schema. Further, provide the identification tags of the associated product line requirements and parameters of variation as well as an identification tag to any Variation Point Schema(s) or Role Schema that the variation point inherits, if any.
   b. Identify the protocols, activities, permissions and responsibilities that are particular to only that variation point.
   c. Document and define the identified protocols, activities, permissions and responsibilities in the appropriate sections of the Role Variation Points Schema. (Note, in accordance with the Gaia conventions, activities are distinguished from protocols by being underlined in Gaia-PL).

These steps result in a set of Role Variation Points Schema that have an associated set of Variation Point Schemas.

4.4 Developing and Documenting the Configuration of a MAS-PL Agent

The Detailed Design phase designs and documents the agents of a MAS-PL by reusing the requirements specification previously developed. To use the derived requirements specifications during the initial deployment of the agents, we exploit the fact that the prior steps have specified all the possible variation points of the roles, to instantiate each new MAS-PL member (i.e., agent) to be added to the MAS-PL by specifying it in a Role Deployment Schema.

The process to design, document and validate an agent of a MAS-PL in our Gaia-PL methodology is as follows:



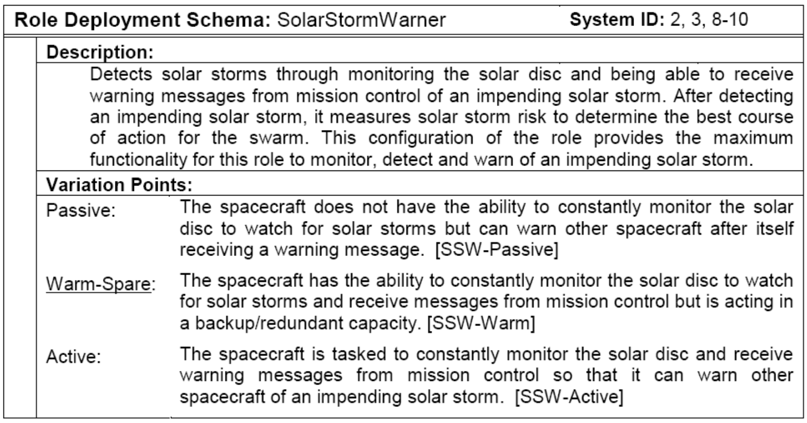
Role Deployment Schema: SolarStormWarner **System ID:** 1, 4-7

Description:
Detects solar storms through monitoring the solar disc and being able to receive warning messages from mission control of an impending solar storm. After detecting an impending solar storm, it measures solar storm risk to determine the best course of action for the swarm. This configuration of the role provides the minimum functionality for this role to only warm of an impending solar storm.

Variation Points:

Passive:	The spacecraft does not have the ability to constantly monitor the solar disc to watch for solar storms but can warn other spacecraft after itself receiving a warning message. [SSW-Passive]

Fig. 7. Role Deployment Schema in Gaia-PL for the *SolarStormWarner* Role with a single variation point.



Role Deployment Schema: SolarStormWarner **System ID:** 2, 3, 8-10

Description:
Detects solar storms through monitoring the solar disc and being able to receive warning messages from mission control of an impending solar storm. After detecting an impending solar storm, it measures solar storm risk to determine the best course of action for the swarm. This configuration of the role provides the maximum functionality for this role to monitor, detect and warn of an impending solar storm.

Variation Points:

Passive:	The spacecraft does not have the ability to constantly monitor the solar disc to watch for solar storms but can warn other spacecraft after itself receiving a warning message. [SSW-Passive]
Warm-Spare:	The spacecraft has the ability to constantly monitor the solar disc to watch for solar storms and receive messages from mission control but is acting in a backup/redundant capacity. [SSW-Warm]
Active:	The spacecraft is tasked to constantly monitor the solar disc and receive warning messages from mission control so that it can warn other spacecraft of an impending solar storm. [SSW-Active]

Fig. 8. Role Deployment Schema in Gaia-PL for the *SolarStormWarner* Role with all variation points.

1. Identify the roles that will constitute the agent to be deployed.
2. For each role identified, create a new Role Deployment Schema and:
   a. Provide the role's name, unique system(s) identification and a brief description of the role specific to this deployment in the appropriate fields of the Role Deployment Schema. The agent(s) System ID identifies the specific member(s) of the distributed system to be deployed that has the role configuration described in the particular Role Deployment Schema. For example, if agents with identification numbers 1, 4-7 are to employ the *SolarStormWarner* role in which only variation point Passive is possible, we denote this in the System(s) ID field of the Role Deployment Schema, as shown in Figure 7. This avoids repetitive manual overhead when designing new members to be deployed in the distributed system and supports traceability, organization and management activities.

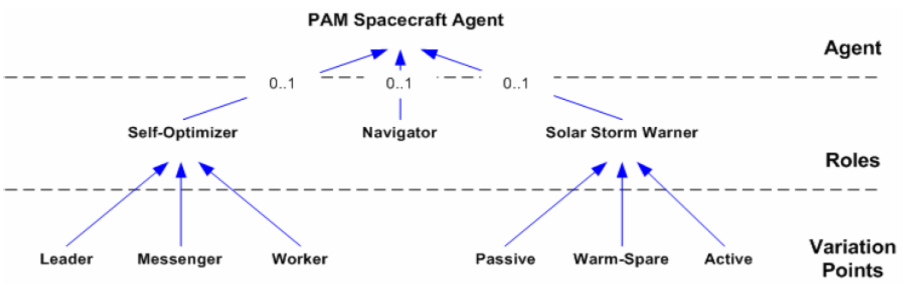


Fig. 9. Excerpt Agent Model in Gaia-PL.

b. Identify all possible variation points that the role can assume during its lifetime. The set of variation points was previously established when the original Role Variation Points Schema was developed for the particular role.
c. Identify the variation point in which the role will be deployed and denote it in the Role Deployment Schema by underlining it. This variation point represents the default variation point at which the agent will most commonly operate during normal operations. For example, Figure 8 denotes the agents that have the *SolarStormWarner* role in which the variation points Passive, Warm-Spare and Active are possible but where the agent is initially configured to operate at the Warm-Spare variation point level.
d. Verify that the selected roles and variation points for the agent conform to the MAS-PL constraint/dependency requirements documented in the Commonality and Variability Analysis, described in Section 4.2.1.

These steps in Gaia-PL are repeated for all agents that are to be deployed in the MAS-PL. These steps produce a set of completed Role Deployment Schemas describing how different agents of the MAS-PL are to be deployed and how they are initially configured.

The use of DECIMAL [Dehlinger et al. 2007; Padmanabhan and Lutz 2005] to document and manage the Commonality and Variability Analysis for the MAS-PL, as described in Section 4.2.1, allows the selection of roles and variation points (Steps 2b and 2c) for a new agent of a MAS-PL and efficiently *verifies* the selection against the MAS-PL dependencies (124 in the PAM study described in Section 3). To make this approach scalable, we utilized DECIMAL to automatically verify that the proposed new agent's set of roles and variation points do not violate the defined dependencies. If any violations are discovered, DECIMAL flags them so that the developer can rectify the problem. This is important because it ensures that the design decisions and constraints are maintained for the agents of the MAS-PL.

An Agent Model, extended from the Agent Model of Gaia [Zambonelli et al. 2003] and shown in Figure 9, can then be developed to graphically illustrate the assignment of roles to agents as well as variation points to roles, similar to that of a feature model. The cardinality relationship between an agent and a role is indicated and all possible variation points are listed for each role. At runtime, the designer annotates the actual cardinality and the specific possible variation points of an agent instance, typically a one-to-one relationship.

In Gaia, the Agent Model defines for each agent the roles that will map to it. Gaia-PL extends this model to additionally map for each role the variation points that may map to it. For example, the partial Agent Model shown in Figure 9 illustrates the *Self-Optimizer*, *Navigator* and *SolarStormWarner* roles and their associated variation points.

4.5 Reusing Requirements Specifications in Gaia-PL

The Gaia-PL methodology takes advantage of how the requirements specifications for an agent's role are partitioned and documented in the Role Variation Points Schema and Variation Point Schema based on their variation points to aid in reuse.

4.5.1 Reuse During Initial System Development. The agents of a MAS-PL often will be heterogeneous in their functional capabilities yet mostly similar in structure. Heterogeneity may also arise when resources (e.g., weight restrictions, memory size, etc.) are limited and different agents of a MAS-PL must assume different roles. Agents of a MAS-PL may also be heterogeneous in terms of their functional capabilities, intelligence levels or other possible variation points.

Requirements specification reuse can be exploited during the initial development and deployment of the agents of a MAS-PL in Gaia-PL using the Role Deployment Schema, shown in Figure 7 and Figure 8. Rather than repeatedly defining the requirements of a role for any given agent (as would be necessary in Gaia), the Role Deployment Schema enables software engineers to define the intelligence levels it can assume. This reuse is possible because the requirements specifications for each of the differing variation points were documented in the Variation Point Schemas, and because the agents of a distributed system will be similar. Thus, to document a particular role for several different heterogeneous members of a MAS-PL we must only indicate which variation points each can assume and give the reference number(s) to the Role Variation Point Schemas as was described in Section 4.4.

4.5.2 Reuse During System Evolution. Change is inevitable. For example, hardware failures or altered mission goals in a deployed system typically necessitate software updates to one or more members. In addition, technology or mission goals routinely evolve after the initial deployment of a distributed system in such a way that future deployments of members joining the distributed system will require additional functionality (i.e., new features).

A deployed MAS-PL can evolve in three ways relevant to this work: 1. new agents may be added to the MAS-PL; 2. new roles with new functionality may be created that future agents can employ; and, 3. new variation points may be added to existing roles that future agents can employ. The following paragraphs discuss how these types of evolution in a MAS-PL are accommodated in the Gaia-PL methodology.

In the first case (new agent), the agent is added as part of the MAS-PL evolution, e.g., to replace a destroyed or failing agent. If this update includes functionality previously defined in the requirements specifications, it suffices to modify the Role Deployment Schema and, possibly, the Agent Model to reflect the update as described in Section 4.4.

In the second case (new role with new functionality), the MAS-PL's requirements specifications must be updated. The addition of a new role during the evolution of a MAS-PL is analogous to the inclusion of a role during initial development, as described in Section 4.3. Briefly, we create a new Role Variation Points Schema and a Variation Point Schema(s) just as during the initial development of a MAS-PL. The process in Gaia-PL's Detailed Design phase, described in Section 4.3.4, is then used to instantiate a new agent with the new role.

In the third case (new variation point added to an existing role), the evolution requires modifying the existing Role Variation Points Schema documentation and creating a new Variation Point Schema. For example, after the deployment of the PAM swarm, mission engineers may decide to include an additional *scout* type of spacecraft (i.e., a new role), that would be tasked to quickly survey asteroids, assess their relevance to the mission goals and decide which asteroids should be further explored. The new *scout* role will include some of the existing functionality of the *leader* and *worker* roles but will additionally contribute new functionality. This addition of new requirements to the PAM MAS-PL will also require updating portions of the feature model, requirements specifications and Agent Model but should not affect existing, already deployed agents. Specifically, to accommodate a new variation point in an existing role for the use in future deployments of the MAS-PL the process in Section 4.3.4, Step 4, is followed for the new variation point. The Agent Model is then updated (as described in Section 4.4) to reflect the inclusion of the new variation point for the role. These steps will produce a new variation point for a role and the accompanying Variation Point Schema for use by future agents of the MAS-PL. In this evaluation, we utilized DECIMAL [Dehlinger et al. 2007; Padmanabhan and Lutz 2005] as the product line requirements engineering management tool to enforce the evolution of new requirements and dependencies/constraints into the detailed design phase of Gaia-PL.

Note that an application project may choose not to incorporate application-specific changes into domain-engineered requirements artifacts. In this case, those changes “are realized just as in single-system engineering” [Pohl et al. 2005]. However, most changes as a product line evolves involve new functionalities and choices (e.g., making a commonality into a variability) that will also be made available to other, future customers. In those cases, updating the MAS-PL’s requirements specifications is a worthwhile investment to ease the traceability and management of the changes.

5. EVALUATION AND DISCUSSION

In this section we evaluate the Gaia-PL methodology in the context of its application to the Prospecting Asteroid Mission (PAM) case study. We also provide a comparison of the Gaia-PL and Gaia methodologies in the context of the PAM case study and a brief discussion of the results and threats to validity.

5.1 Application to the Prospecting Asteroid Mission

The application of the Gaia-PL methodology to the PAM during the Collection of Requirements phase (described in Section 4.2) documented 97 high-level multi-agent system product line (MAS-PL) requirements in the Commonality and Variability Analysis. The 97 high-level MAS-PL requirements included 35 commonality requirements and 62 variability requirements. Thus, we found that approximately one-third of the requirements were shared by all spacecraft regardless of their specialized role (i.e., *Leader*, *Messenger* or *Worker* spacecraft). Further, the product line requirements of the PAM case study were partitioned into 47 high-level features for the feature model, discussed in Section 4.2.

In the Analysis and Design phase of Gaia-PL (described in Section 4.3), we identified 13 unique roles for the PAM case study that were documented in 2 Role Schemas, 11 Role Variation Points Schemas and 39 Variation Point Schemas, as discussed in Section

4.3. These requirements specifications schemas can be used to design and develop **160 unique** PAM spacecraft (80 different *Worker* spacecraft, 48 different *Leader* spacecraft and 32 different *Messenger* spacecraft). Thus, the reuse of the 52 schemas developed in Gaia-PL's Analysis and Design phase was able to accommodate the development of a wide range of PAM spacecraft.

To measure the impact and ability of the inclusion of variation points into the roles of an agent in a MAS, we measured: (1) the number of variation points defined for each role; (2) the number of parameters of variation; and, (3) the number of requirements implemented in each variation point. These measurements provided insight into the extent of the variable behavior of an agent that can be defined for a role and illustrates the advantage of the inclusion of product line engineering into the development of a MAS-PL in Gaia-PL.

The Role Variation Points Schemas developed for the PAM case study during the Analysis and Design phase of Gaia-PL had an average of 3.9 variation points with the minimum number of variation points identified for a role being 2, and the maximum 10. Further, the Variation Point Schemas implemented an average of 4.1 high-level requirements from the Commonality and Variability Analysis with the minimum number of requirements implemented in a variation point being 1, and the maximum 14. Note that many of the high-level requirements were implemented in several roles (i.e., cross-cut more than one role). For example, the requirement "Every spacecraft shall be able to know its current position" is needed in multiple roles.

Of the 11 Role Variation Points Schemas identified for the PAM case study, 8 contained a variation point that must be included if the role is included in the agent. For example, the *Messenger* role (i.e., a role that not every agent will contain) contains two variation points, one of which is required. For the *Messenger* role, the required variation point captures 6 of the 8 requirements that are associated with the functionality possible in the *Messenger* role. That is, 6 of the 8 requirements were common to all agents containing the *Messenger* role while only 2 of the 8 requirements were optional.

The *SolarStormWarner* role, discussed throughout Section 4, similarly captured a large portion of the role's common requirements in its required variation point. However, unlike the *Messenger* role, the *SolarStormWarner* role is required for all PAM spacecraft. Nevertheless, the required variation point for the *SolarStormWarner* role captured 54.5% of the common requirements in its Variation Point Schema.

Among the 8 Role Variation Points Schemas of the PAM case study that contained a variation point that must be included if the role is included in the agent, an average of 41% of the requirements were found to be common to the required variation point of the role. The minimum amount of common requirements for a role was 13% for the *Worker* role and the maximum was 75% for the *Messenger* role. Thus, using the Role Variation Points Schema in Gaia-PL captures, at least in the case of the PAM case study, a sizeable portion of the requirements common to all agents with a particular role and can be reused to develop agents with any allowable combination of the role's variation points. Further, the ability to separately capture the common requirements of a role in a variation point avoids the need to have the common requirements repeated in several role schemas for each of the variation points. The design and documentation of the 39 Variation Point Schemas for the PAM case study took approximately 30 minutes each for a total of 19.5 hours. Thus, for each requirement implemented in a Role Variation Point Schema, an average of 7.3 minutes was needed here to document the requirement's specification in a Variation Point Schema.

5.2 Comparison to the Gaia Methodology

The main contribution of the Gaia-PL methodology detailed in this article is to provide a way to develop software engineering assets that can be readily reused to build the agents of a MAS-PL. The mechanism to provide the reusable assets in the Gaia-PL methodology centers on the identification and separation of the commonalities of the agents and the agent's roles and the refinement of the variabilities of the agents and the agent's roles in separate software engineering artifacts. The ability to separately capture the common requirements of a role in a variation point avoids the need to have the common requirements repeated in several role schemas for each of the variation points.

Our application of the Gaia methodology [Wooldridge et al. 2000; Zambonelli et al. 2003] to the requirements for the PAM study listed in the Commonality and Variability Analysis yielded 48 Gaia Role Schemas (similar to Gaia-PL's Variation Point Schemas) for the 160 unique agents. To accommodate the requirements of the PAM case study, Gaia would need to implement a role for each of our variation points where the variable variation points (i.e., non-required) additionally included the required variation point functionality. For example, in the *SolarStormWarner* role, a new Role Schema in Gaia has to be created for the Passive, Warm-Spare and Active variation points. Further, the new roles for the Warm-Spare and Active roles must include the functionality of the

Passive variation point. Thus, the Gaia Role Schema for the Warm-Spare and Active *SolarStormWarner* roles repeat the functionality of the Passive variation point.

While Gaia can accommodate the functionality of the PAM case study, it does not clearly document an agent's ability to change from one set of functionalities of a role to another (e.g., from the Warm-Spare to the Active functionality of the *SolarStormWarner* role). Rather, Gaia has to combine the functionality from the variation points into a single role. The disadvantage is that, unlike Gaia-PL, this does not keep the modularity of the differing types of functionality in a role so may confuse developers during coding. Additionally, unlike Gaia-PL, the non-hierarchical nature of Gaia cannot provide any linking relationships between related roles (e.g., from the Warm-Spare to the Passive functionality of the *SolarStormWarner* role). Lastly, some functionality is unnecessarily repeated in Gaia (e.g., the Passive functionality also must be included in the Warm-Spare and Active roles of a SolarStormWarner).

Application of the Gaia methodology to the PAM case study increased the number of schemas needed compared to our Gaia-PL approach. We found that an average of 41% of the requirements implemented in the required variation points of 8 of the 11 Role Variation Points schemas were common to all variation points of the role. Since the redundant requirements need to be documented for each variation point to create a new role in Gaia, 66.5% of the Role Schemas had already been documented in another role. Of these 8 roles identified in Gaia-PL (with 35 variation points), Gaia created 41 roles that contained 33 redundant requirements. Due to the high number of redundant requirements, the 33 Role Schemas created in Gaia documented 222 requirements (of which 66.5% or 147 requirements are redundant). Assuming that it continues to take an average of 7.3 minutes to specify a requirement, the Gaia approach incurred an *additional* 17.8 hours to derive and document specifications compared to the Gaia-PL approach. Thus, Gaia-PL showed a 48% reduction in the design and documentation time over Gaia in the case study.

5.3 Validity

The evaluation of our Gaia-PL methodology using the PAM study measures Gaia-PL's ability to capture the common parts of a MAS-PL so that they can be reused along with the variable parts to design and develop agents. It was shown in the previous section that compared to Gaia, an agent-oriented software engineering methodology that does not explicitly partition the common and variable parts of a MAS-PL, Gaia-PL's ability to reuse the common parts of an agent's role reduces the work and time required to design and develop an agent. However, the evaluation of our Gaia-PL methodology does not come without caveats. In this section, we discuss some of the threats to validity of our evaluation.

5.3.1 Internal Validity. The internal validity (i.e., whether the measured reuse of requirements specifications accurately reflects the ability of the methodology to reduce the documentation and time required) faced several challenges.

The first set of threats involves the analysts and the application of the Gaia-PL and Gaia methodologies to the PAM requirements. The evaluation and comparison of our Gaia-PL methodology with the Gaia methodology was performed serially by a single analyst. The design and documentation of the PAM specifications using the Gaia methodology may have been influenced by that of the prior application of the Gaia-PL methodology to the PAM case study to produce its specifications. However, additional familiarity with PAM gained prior to applying the Gaia methodology to it could only make the Gaia representation easier to perform, so would not bias the evaluation against Gaia. Furthermore, this evaluation did not consider design alternatives in the application of Gaia-PL to the PAM case study. That is, we did not design and evaluate different ways of defining a role's variation points nor did we design and evaluate different ways of defining the roles possible in an agent. Thus, the results obtained from our evaluation might differ if we had used a different, non-Gaia-based design approach for the PAM study.

A second set of threats includes the availability and accuracy of the data (i.e., requirements) used to construct the PAM case study. The primary threats to validity are:

1. *Availability of requirements*. The original requirements for the proposed PAM spacecraft were not obtainable. Instead, the requirements used in this study were derived from published descriptions of the system including [Rouff et al. 2005; Sterritt et al. 2005; Truszkowski et al. 2004; Truszkowski et al. 2006]. The consistency of the descriptions of PAM across multiple sources gives some indication that they accurately reflected the original requirements.
2. *Incomplete descriptions*. The existing descriptions of the PAM requirements were at a high level, necessitating refinement into detailed specifications. We used domain knowledge from the detailed specifications of similar systems, including

[Chien et al. 2002; Schetter et al. 2000], as well as our own experience on spacecraft to derive the lower-level requirements.

A final threat to the internal validity is the underlying assumption that the measurement of reuse of requirements and specifications will appropriately reflect the benefits of the methodology reduction in cost. In this study, we claimed reuse based on the number of common and reused requirements for several spacecraft in the PAM family. The metric of requirement/specification reuse is common within software product line engineering and is the basis for several techniques including the FAST [Weiss and Lai 1999], goal-oriented [Castro et al. 2002] and feature-oriented [Kang et al. 2002] approaches. Increasing the amount of requirements specification reuse for any given product is expected to reduce the production time and cost of the software system [Clements and Northup 2002; Pohl et al. 2005]. However, the assumption that increasing requirements reuse reduces development costs in the product line context might be challenged by further empirical investigations.

5.3.2 External Validity. The external validity (i.e., the extent to which the conclusions asserted in this work can be generalized) may be limited by our representative case study and the domain of interest (spacecraft swarms).

Our evaluation of Gaia-PL was performed on one relatively large MAS-PL application. The use of Gaia-PL on other MAS-PL applications might yield different results. Thus, the evaluation reported in this article serves as a proof-of-concept study that would need to be repeated on other MAS-PL applications to draw generalized expectations.

The PAM case study, and other spacecraft swarms, used in this study had requirements that fit nicely into adopting a software product line engineering approach. The requirements gathered for the PAM case study readily fit into a Commonality and Variability Analysis because the common and variable functionalities of the spacecraft were clear. A characteristic of the PAM case study aiding its adoption into a software product line engineering approach was its basis on the Autonomous Nano-Technology Swarm (ANTS) concepts [Rouff et al. 2005; Sterritt et al. 2005; Truszkowski et al. 2004; Truszkowski et al. 2006]. The requirement that all PAM spacecraft implement the functionality of the ANTS mission provided a natural mechanism to define the commonalities. Thus, the reuse reported in our evaluation is not limited to the single PAM swarm application explored in this work but has implications for reuse in other ANTS-based mission spacecraft [Peña et al. 2006b].

In addition, the variability requirements of the PAM mission partly focused on the differing functionality among the types of spacecraft (i.e., *Leader*, *Messenger* or *Worker*). Further, there was approximately a two-to-one ratio of variable requirements to commonality requirements. These factors certainly contributed to the clear advantage that Gaia-PL displayed compared to Gaia. For MAS-PL's with less variability within a role, or for MAS-PL's in which no variation points for a role can be identified, Gaia-PL may not provide such clear advantages.

However, in the case where a MAS-PL has little variability, Gaia-PL will not incur enough overhead to be a disadvantageous approach compared to Gaia. Unlike Gaia, Gaia-PL does require the documentation of variation points (if any) in a Role Variation Points Schema which will incur additional development time. Yet, for MAS-PL's that will have few variation points (and thus few variabilities), the Role Variation Points needed will be few and require very little development time. Thus, the Gaia-PL approach would still provide some advantage for those roles which have variation points while not incurring a large overhead.

5.3.3 Using Gaia-PL to Design Agent-Based Systems. Despite these threats, the evaluation of Gaia-PL indicates its advantages in designing, developing and documenting MAS-PLs that have some degree of variability. Gaia-PL's ability to hierarchically define the roles of an agent, capture the common and variable functionality of an agent and reuse the common functionality of a role to design and develop a wide-range of agents of the MAS-PL recommends its use. In particular, the use of Gaia-PL allows the software developer to take advantage of the reuse potential in Gaia-PL along with the other models, abstractions and analysis tools of Gaia to provide the mechanisms to efficiently design and develop a MAS-PL.

6. CONCLUDING REMARKS

This article advocated the inclusion of software product line engineering in agent-oriented software engineering and detailed the design and development of a multi-agent system product line (MAS-PL) using the Gaia-PL methodology. The Gaia-PL methodology produces reusable software engineering assets so that systems in the MAS-PL can be built efficiently with a high degree of reuse. In particular, we described and illustrated the reuse of the requirements specifications during initial system development of a MAS-PL as well as during system evolution. To highlight the advantages of Gaia-PL, we differentiated and evaluated our methodology against previous work by illustrating

Gaia-PL's ability to capture reuse and decrease redundant work in developing the required agents.

While this integrated approach is an important step in providing software engineers with a viable methodology to capture and utilize reusable assets for efficient design and development of agent-based software systems, additional work remains. Issues to be addressed include:

- Expansion and application of Gaia-PL into other portions of Gaia [Wooldridge et al. 2000; Zambonelli et al. 2003] to cover a broader selection of the models and phases in the development of MAS-PL
- Investigation into how Feature-Oriented Product Line Engineering [Kang et al. 2002] can further support our use of a feature model to facilitate the design, development, management and reuse of the assets of a MAS-PL
- Inclusion/adaptation of portions of the MaCMAS agent-oriented software engineering methodology [Peña et al. 2006a; Peña et al. 2006b] to handle the complexity of MAS-PL and build a core architecture to supplement Gaia-PL's focus on reuse during the requirements and early design phases
- Development of comprehensive tools to support and facilitate the creation and reuse of the core and variable software engineering assets of a MAS-PL

REFERENCES

ARDIS, M. AND WEISS, D. 1997. Defining families: the commonality analysis. In *Proceedings of the 19th International Conference on Software Engineering*, Boston, MA, pp. 649-650.

BEUCHE, D., BIRK, A., DREIR, H., FLEISCHMANN, A., GALLE, H., HELLER,G., JANZEN, D., JOHN, I., TAVAKOLI KLAGARI, R., VON DER MAAYEN, T. AND WOLFRAM, A. 2007. Using requirements management tools in software product line engineering: the state of the practice In *Proceedings of the 11th International Conference on Software Product Line Engineering*, Kyoto, Japan, pp. 84-96.

BRESCIANI, P., GIORGINI, P., GUINCHIGLIA, F. AND PERINI, A. 2004. Tropos: an agent-oriented software development methodology. In *Journal of Autonomous Agents and Multi-Agent Systems*, 8(1):203-236.

CASTRO, J., KOLP, M. AND MYOPOULOS, J. 2002. Towards requirements-driven information systems engineering: the Tropos project. In *Information Systems*, 27(6):365-389.

CERNUZZI, L. JUANT, T., STERLING, L. AND ZAMBONELLI, F. 2004 The Gaia methodology: basic concepts and extensions. In *Methodologies and Software Engineering for Agent Systems – The Agent-Oriented Software Engineering Handbook Series*, 11:69-88.

CHAN, K. AND STERLING L. 2003. Specifying roles within agent-oriented software engineering. In *Proceedings of the 10th Asia-Pacific Software Engineering Conference*, Chiangmai, Thailand, December 2003, 390-395.

CHIEN, S., SHERWOOD, R., RABIDEAU, G., CASTANO, R., DAVIES, A, BURL, M., KNIGHT, R., STOUGH, T., RODEN, J., ZETOCHA, P., WAINWRIGHT, R., KLUPAR, P., VAN GAASBECK, J., CAPPELAERE, P, AND OSWALD, D. 2002. The TechSat-21 Autonomous Space Science Agent. In *Proceedings of the 1st International Conference on Autonomous Agents*, Bologna, Italy, pp. 570-577.

CLEMENTS, P. 2002. Being proactive pays off. In *IEEE Software*, 19(4):28, 30.

CLEMENTS, P. AND NORTHROP, L. 2002. *Software Product Lines*, Addison-Wesley, Boston, MA.

COSSENTINO, M. 2005. *From requirements to code with the PASSI methodology*, Idea Group Inc., Hershey, PA, USA, chapter IV.

DEHLINGER, J. 2007. Incorporating product-line engineering techniques into agent-oriented software engineering for efficiently building safety-critical, multi-agent systems. Ph.D thesis, Iowa State University, Department of Computer Science.

DEHLINGER, J. HUMPHREY, M., PADMANABHAN, P. AND LUTZ, R.R. 2007. DECIMAL and PLFaultCAT: from product-line requirements to product-line member software fault trees. In *Proceedings of the 29th International Conference on Software Engineering*, Minneapolis, MN, pp. 49-50.

DEHLINGER, J. AND LUTZ, R.R. 2005. A product-line approach for safe reuse in multi-agent systems. In *Proceedings of the 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*, St. Louis, MO, pp. 83-89.

DEHLINGER, J. AND LUTZ, R.R. 2006. A product-line approach to promote asset reuse in multi-agent systems. *Software Engineering for Multi-Agent Systems IV*, Lecture Notes in Computer Science 3914, pp. 161-178.

DEHLINGER, J. AND LUTZ, R.R. 2008. Supporting Requirements Reuse in Multi-Agent System Product Line Design and Evolution. In *Proceedings of the 24th IEEE International Conference on Software Maintenance*, Beijing, China, pp. 207-216.

DELOACH, S.A. 2004. The MaSE methodology. In *Methodologies and Software Engineering for Agent Systems – The Agent-Oriented Software Engineering Handbook Series*, 11:107-125.

DOERR, J. 2002. Requirements engineering for product lines: guidelines for inspecting domain model relationships. Diploma Thesis, University of Kaiserslautern.

GIORGINI, P., KOLP, M., MYLOPOULOS, J. AND PISTORE, M. 2004. The Tropos methodology. In *Methodologies and Software Engineering for Agent Systems – The Agent-Oriented Software Engineering Handbook Series*, 11:89-106.

HARA, H., FUJITA, S. AND SUGAWARA, K. 2000. Reusable software components based on an agent model. In *Proceedings of the Workshop on Parallel and Distributed Systems*, Iwate, Japan, pp. 447-452.

JACOBSON, I., GRISS, M., AND JONSSON, P. 1997. *Software Reuse: Architecture, Process and Organization for Business Success*, Addison-Wesley Professional, Boston, MA.

JENNINGS, N. AND WOOLDRIDGE, M. 2000. On agent-oriented software engineering. In *Artificial Intelligence*, 117(2):277-296.

JUAN, T. AND STERLING, L. 2002. ROADMAP: extending the Gaia methodology for complex open systems. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, July, 2003, 3-10.

KANG, K.C., LEE., J. AND DONOHOE, P. 2002. Feature-oriented product line engineering. In *IEEE Software*, 19(4):58-65.

LUCK, M., MCBURNEY, P., and PREIST, C. 2004. A manifesto for agent technology: towards next generation computing. In *Autonomous Agents and Multi-Agent Systems,* 9(3): 203-252.

NUNES, I., KULESZA, U., NUNES, C. and LUCENA, C. 2009. A domain engineering process for developing multi-agent system product lines. Extended abstract. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, Budapest, Hungary, May 2009, pp. 1339-1340.

NUNES, I., KULESZA, U., NUNES, C., CIRILO, E., and LUCENA, C. 2008. Developing and evolving a multi-agent system product line: an exploratory study. In *Agent-Oriented Software Engineering Ix: 9th International Workshop, AOSE 2008 Estoril, Portugal, May 12-13, 2008 Revised Selected Papers*, M. Luck and J. J. Gomez-Sanz, Eds. Lecture Notes In Computer Science, vol. 5386. Springer-Verlag, Berlin, Heidelberg, pp. 228-242.

NUNES, I., KULESZA, U., NUNES, C., CIRILO, E., and LUCENA, C. 2009. Extending PASSI to model multi-agent systems product lines. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, Honolulu, HI. pp. 729-730.

PADMANABHAN, P. AND LUTZ, R.R. 2005. Tool-supported verification of product line requirements. In *Automated Software Engineering Journal*, 12(4):447-465.

PEÑA, J., HINCHEY, M. AND CORTÉS, A. 2006. Multi-agent system product lines: challenges and benefits. In *Communications of the ACM*, 49(12):82-84.

PEÑA, J., HINCHEY, M., CORTÉS, A. AND TRINIDAD, P. 2006. Building the core architecture of a NASA multiagent system product line. In *Proceedings of the 7th International ACM Workshop on Agent Oriented Software Engineering*, Hakodate, Japan, pp. 13-24.

POHL, K., BOCKLE, G. AND VAN DER LINDEN, F. 2005. *Software Product-Line Engineering*, Springer-Verlag, Berlin, Germany.

ROUFF, C., HINCHEY, M., RASH, J., TRUSZKOWSKI, W. AND STERRIT, R. 2005. Towards autonomic management of NASA missions. In *Proceedings of the 11th International Conference on Parallel and Distributed Systems*, Fukuoka, Japan, pp. 473-477.

SCHETTER, T., CAMPBELL, M. AND SURKA, D. 2000. Multiple Agent-Based Autonomy for Satellite Constellations. In *Proceedings of the 2nd International Symposium on Agent Systems and Applications*, Zurich, Switzerland, pp. 147-180.

SCHMID, K. AND VERLAGE, M. 2002. The economic impact of product line adoption and evolution. In *IEEE Software*, 19(4):50-57.

SOMMERVILLE, I. 2004. *Software Engineering*, Pearson Addison-Wesley, Boston, MA.

STERRITT, R., ROUFF, C., RASH, J., TRUSZKOWSKI, W. AND HINCHEY, M. 2005. Self-* properties in NASA missions. In *Proceedings of the 2005 International Conference on Software Engineering Research and Practice*, Las Vegas, NV, pp. 66-72.

SVANBERG, M., GURP, J. AND BOSCH, J. 2005. A taxonomy of variability realization techniques. In *Software – Practice and Experience*, 35(8):705-754.

TRUSZKOWSKI, W., HINCHEY, M., RASH, J. AND ROUFF, C. 2006. Autonomous and autonomic systems: a paradigm for future space exploration missions. In *IEEE Transactions on Systems, Man and Cybernetics*, 36(3):279-291.

TRUSZKOWSKI, W., RASH, J., ROUFF, C. AND HINCHEY, M. 2004. Asteroid exploration with autonomic systems. In *Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, Brno, Czech Republic, pp. 484-489.

TVEIT, A. 2000. A survey of agent-oriented software engineering. Report, Norwegian University of Science and Technology.

VAN OMMERING, R. 2005. Software reuse in product populations. In *IEEE Transactions on Software Eningeering*, 31(7):537-550.

WEISS, D.M. AND LAI, C.T.R. 1999. *Software Product Line Engineering: A Family-Based Software Development Process*, Addison-Wesley, Boston, MA.

WOOLDRIDGE, M. 1997. Agent-based software engineering. In *IEEE Proceedings on Software Engineering*, 144(1):26-37.

WOOLDRIDGE, M., JENNINGS, R. AND KINNY, D. 2000. The Gaia methodology for agent-oriented analysis and design. In *Journal of Automomous Agents and Multi-Agent Systems*, 3(3): 285-312.

ZAMBONELLI, F., JENNINGS, R. AND WOOLDRIDGE, M. 2003. Developing multiagent systems: the GAIA methodology. In *ACM Transactions on Software Engineering and Methodology*, 12(3): 317-370.

Received xxxx; revised xxxx; accepted xxxx.