

Bi-Directional Safety Analysis for Product-Line, Multi-Agent Systems

Josh Dehlinger
Department of Computer Science
Iowa State University
226 Atanasoff Hall
Ames, IA 50011
1 515-294-2735
dehlinge@iastate.edu

Robyn R. Lutz
Department of Computer Science
Iowa State University and
Jet Propulsion Laboratory/Caltech
226 Atanasoff Hall
Ames, IA 50011
1 515-294-3654
rlutz@cs.iastate.edu

ABSTRACT

Safety-critical systems composed of highly similar, semi-autonomous agents are being developed in several application domains. An example of such multi-agent systems is a fleet, or “constellation” of satellites. In constellations of satellites, each satellite is commonly treated as a distinct autonomous agent that must cooperate to achieve higher-level constellation goals. In previous work, we have shown that modeling a constellation of satellites or spacecraft as a product line of agents (where the agents have many shared commonalities and a few key differences) enables reuse of software analysis and design assets. We have also previously developed efficient safety analysis techniques for product lines.

We now propose the use of Bi-Directional Safety Analysis (BDSA) to aid in system certification. We extend BDSA to product lines of multi-agent systems and show how the analysis artifacts thus produced contribute to the software’s safety case for certification purposes. The product-line approach lets us reuse portions of the safety analysis for multiple agents, significantly reducing the burden of certification. We motivate and illustrate this work through a specific application, a product-line, multi-agent satellite constellation.

Categories and Subject Descriptors

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence – *multi-agent systems* D.2.4 [Software Engineering]: Software/Program Verification – *reliability* D.2.1 [Software Engineering]: Requirements/Specifications

General Terms

Design, Reliability, Verification

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ITCES’06, April 4, 2006, San Jose, California, USA.
Copyright 2006 ACM 1-58113-000-0/00/0004...\$5.00.

Keywords

Software safety, multi-agent systems, product-line engineering, system certification

1. INTRODUCTION

The emergence of distributed systems (e.g., formation flying of satellite constellations) as a viable and reliable architecture for mission-critical domains coupled with the advantages of adopting an agent-oriented perspective for software development has led to a number of proposed systems combining these two concepts. A multi-agent system (MAS) is an application “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 languages” [24]. Actual proposed systems including the Terrestrial Planet Finder-I (TPF-I) spacecraft [22] and the TechSat-21 [3], Sun-Solar System Connection, Search for Earthlike Planets and Universe Exploration all rely on constellation missions to achieve their scientific goals [18]. Agent-oriented software engineering (AOSE) appears to be an appropriate software development methodology for such systems [21].

Certification is a process whereby a certification authority determines if an applicant provides sufficient evidence concerning the means of production of a candidate product and the characteristics of the candidate product so that the requirements of the certifying authority are fulfilled [11, 13, 19, 20]. Software safety analysis techniques, similar to those used in this work, have previously been shown to contribute to the certification of software-intensive systems in [1, 16]. However, little work has been specifically aimed at software product lines of MAS. A software product line is defined as a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission [23]. The work presented here tailors the safety analysis techniques to a particular AOSE methodology, Gaia, to support certification of product-line, agent-based systems.

The main contribution of this paper is to extend Bi-Directional Safety Analysis (BDSA) to product line MAS and show how the analysis artifacts thus produced contribute to the software’s safety case for certification purposes. The product-line approach allows

us to reuse portions of the safety analysis assets for multiple, similar agents, significantly reducing the burden of certification.

First, we further the inclusion of safety analysis techniques into AOSE by providing a structured process to perform a Software Failure Modes, Effects, Criticality Analysis (SFMECA) for safety-critical, product-line MAS. The SFMECA is reusable for other agents in the system since our approach incorporates the product-line vision of a MAS from [6].

Second, safety analysis techniques contribute to system certification of product-line MAS by verifying software design compliance with reliability, robustness and safety standards. Because the safety analysis is performed on the product line as a whole (rather than serially on each individual product-line member), the safety analysis assessment techniques described in this work may significantly reduce the time and cost of certifying a safety-critical MAS.

This paper illustrates the process and contributions of this work using an agent-based implementation of a satellite constellation loosely based on the requirements for the TechSat21 [3, 21]. TechSat21 was a proposed mission, originally scheduled to launch in January 2006 but cancelled in late 2003 with much of the software reused on a subsequent mission [4]. It was designed to explore the benefits of a distributed, cooperative approach to satellites employing agents [3].

The remainder of this paper is organized as follows. Section 2 reviews background and related work in software safety analysis techniques and product-line MAS. Section 3 details our approach in utilizing the BDSA technique for safety-critical, product-line MAS to assist in certifying the composite system. Finally, Section 4 provides concluding remarks and future research directions.

2. BACKGROUND AND RELATED WORK

The research presented here integrates existing work in software safety analysis with software engineering for multi-agent systems (MAS) to aid in system certification. Certification may apply to the development process, the developer or the actual product [16]. Since it is insufficient to certify the process or developer for the software of safety-critical systems, building a safety case that provides “an argument accompanied by evidence that all safety concerns and risks have been correctly identified and mitigated” [10] aids in the certification of the product. Further, this work builds upon our previous work of integrating the reuse potential of safety analysis assets into the design and development of product-line MAS.

2.1 Software Safety Techniques

Software safety analysis centers on the investigation of how software can jeopardize or contribute to the safety of the system [15]. Two common techniques used in software safety analysis are Software Failure Modes, Effects and Criticality Analysis (SFMECA) and Software Fault Tree Analysis (SFTA). Bi-Directional Safety Analysis (BDSA) combines these two techniques in order to provide both a forward analysis to determine systems effects of software failure modes to effects on the systems and to determine if those failure modes are possible in the system to be certified [16].

SFMECA is a tabular, forward (inductive) search technique that starts with the failure of a component or subsystem and then looks at its effect on the overall system [15]. In [17], a list of generic failure-mode guidewords is given to aid in the process of constructing a SFMECA for failure in data communication and event processing. These guidewords, when applied to the failure of a component or subsystem, help engineers systematize the process of determining the possible effects of each failure mode on other components of the system that could lead to a hazard(s).

SFTA is a tree-based, backward (deductive) technique that typically has as its root node a system-wide, catastrophic event [15]. Analysis proceeds by determining the set of necessary preconditions causing the occurrence of the hazard. The set of possible causes are connected to the parent node by logic gates to describe their contributing relation. This process continues through each level of the constructed subtree until basic events are reached or until the desired level of subsystem detail is achieved.

A technique to cleanly extend SFTA to software product lines was introduced in [8]. A SFTA can be constructed for an entire product line and product-line members’ fault trees can be derived from the product-line SFTA. PLFaultCAT, a graphical tool to construct a product-line SFTA, exploits this technique and then allows users to automatically derive a product-line members’ fault tree given the variabilities to be included [8].

BDSA combines a search from potential failure modes to their effects with a search from possible hazards to the contributing causes of each hazard [17]. Although BDSA does not require SFMECA and SFTA to be used as the forward and backward search, respectively, we follow [12] and [16] in using these techniques in our BDSA.

2.2 Software Safety Techniques

Reuse of software-engineering assets continues to be a demand on software system development methodologies. Software product-line engineering models provide software engineers a reuse-conscious development platform that can contribute to significantly reducing both the time and cost of software requirements specification, development, maintenance and evolution [5]. In a product line, the common, managed set of features shared by all members is 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).

Agent-oriented software engineering (AOSE) has provided tools and techniques allowing for natural, high-level abstractions in which software developers can understand, model and develop complex systems [24]. Several AOSE methodologies have been proposed for various types of application domains including Tropos [2] and MaSE [9]. We selected Gaia [24] as the AOSE design methodology with which to incorporate safety analysis because of its extensive documentation and acceptance in the AOSE community.

The Gaia methodology centers on defining an agent based upon the role(s) that it can assume. Each role’s requirements specification is defined by its protocols (i.e., defines 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).

Using the Gaia methodology, Dehlinger and Lutz applied the notion of an agent having different possible levels of intelligence for a given role to investigate the reuse advantages of product-line engineering in developing multi-agent systems (MAS) [6]. For example, a role in a distributed system of nodes, depending on its environment and context, may have one of the following levels of intelligence:

- I4: receive/execute commands
- I3: local planning and receive/execute commands
- I2: local planning, interaction, partial system-knowledge and receive/execute commands
- I1: system-level planning, interaction, full systems-knowledge and receive/execute commands

The level of intelligence for a role may dynamically change during run-time depending on the system’s organization and/or goals. For example, at any given time only a single agent in a distributed system may have *role X* operating at intelligence level I1. However, several other agents with *role X* may operate at intelligence level I3 but be capable of dynamically increasing the role’s intelligence level to I1 if needed (i.e., a hot-spare/warm-spare concept). Similarly, some agents with *role X* may be restricted to operating only in I4 or I3 due to resource constraints or design decisions. Adopting this view, a MAS can be designed using the notions of product-line engineering to fully take advantage of the reuse principles inherent in product lines.

In [7], it was shown how safety analysis can ensure the safety and reliability of product-line MAS using SFTA. This work extends [7] to include a SFMECA safety analysis enabling BDSA for product-line MAS.

3. APPLYING BI-DIRECTIONAL SAFETY ANALYSIS TO MULTI-AGENT SYSTEMS

The use of Bi-Directional Safety Analysis (BDSA), detailed in Section 3.3, requires the use of forward and backward searches. In this work, we use Software Failure Modes, Effects and Criticality Analysis (SFMECA), discussed in Section 3.1, and Software Fault Tree Analysis (SFTA), discussed in Section 3.2, as the forward and backward search technique, respectively.

Using the Gaia methodology [24], we situate the safety analysis step, shown in Figure 1, as using the software engineering assets (i.e., the Role Schema) of the “Analysis and Design” phase but also augmenting the requirements specifications of the “Analysis and Design” phase. Thus, the safety analysis, in addition to generating safety analysis assets (e.g., SFMECA tables, software fault trees, etc.) used to make a safety case for the software during system certification, aids in verifying the safety requirements and discovering safety requirements missed in the initial requirements specification. Again, because the multi-agent system (MAS) is viewed as a product line, the safety analysis is providing safety case assets for any product line member.

In the *Analysis and Design* phase of the Gaia methodology [24], the software engineer specifies the requirements in a Role Schema, shown in Figure 2, when constructing a product-line MAS. Safety requirements for a role are listed in the form of safety properties that the agent must ensure in the *Responsibilities* section. However, Gaia provides no structured way by which to discover safety requirements. Similarly, Gaia provides no process by which to check that the safety requirements suffice to mitigate possible hazards. In the following sections, we detail how performing BDSA can help verify and complete the safety properties that a role must guarantee.

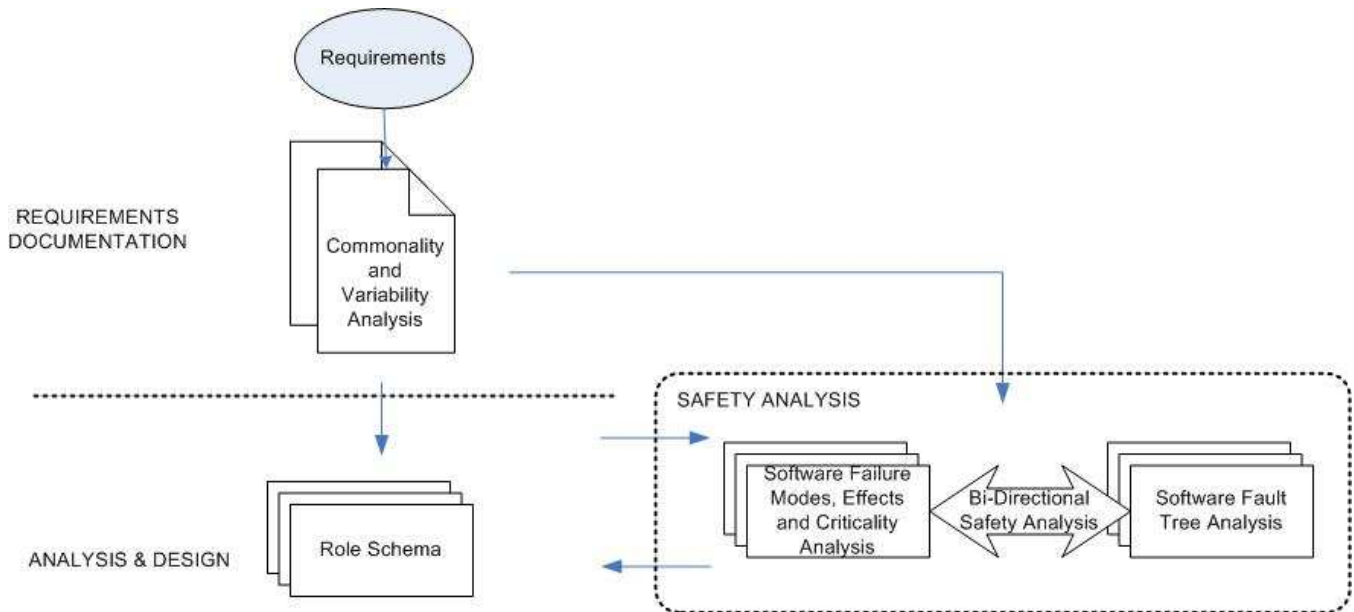


Figure 1. An overview of our process situated in the Gaia-based product-line approach in developing MAS.

Role Schema: Cluster Allocation Planner	Schema ID: F32-I1
Description:	
Assigns a new cluster configuration by assigning new satellite positions within the cluster. This is done to equalize fuel use across the cluster. With the I1 intelligence level, it is able to send cluster assignments to other satellites (i.e., spacecraft level agents) in order to arrange a new cluster configuration. This may occur when a new satellite is added or in the case of a failure of a satellite.	
Protocols and Activities:	
CalculateDeltaV, UpdateClusterInformation, MoveNewPos, DeOrbit, <u>AssignCluster</u> , <u>AcceptDeltaVBids</u> , <u>RequestDeltaVBids</u> , <u>SendMoveNewPosMsg</u> , <u>SendDeOrbitMsg</u>	
Permissions:	
Reads -	
<i>position</i>	// current satellite position
<i>velocityIncrement</i>	// current satellite velocity increment
<i>supplied satelliteID</i>	// satellite identification number
<i>supplied velocityIncrment</i>	// satellite velocity increment
Changes -	
<i>position</i>	// current satellite position
<i>velocityIncrement</i>	// current satellite velocity increment
Generates -	
<i>newPositionList</i>	// new position list to assign to the // satellites within the cluster
Responsibilities:	
Liveness -	
Optimize the fuel use across the cluster.	
Safety -	
Prevent satellite collisions during a new cluster configuration.	

Figure 2. An example of the requirements specification for a role of the TechSAT21 satellite constellation specified as a product line in Gaia’s Role Schema.

3.1 Forward Search Safety Analysis

The forward analysis uses a Software Failure Modes, Effects and Criticality Analysis (SFMECA). Gaia’s Role Schema conveniently partitions a role’s requirements specifications into events (functionality) that the role can perform and data that the role can access. Like [12], we partition the SFMECA into separate analyses on the data and events. We use guidewords of [16] to steer our investigation into the possible failures within a product-line multi-agent system (MAS). We here describe first the construction of the SFMECA event table for the Gaia Role Schema and then the construction of the SFMECA data table.

Each activity of a role (the non-underlined keywords listed in Gaia’s Role Schema under the *Protocols and Activities* section) is essentially an event (i.e., some functionality) that the role can execute. To construct a SFMECA table for the events that a role can execute, as in a standard SFMECA we use the following keywords to guide our analysis: “halt/abnormal termination”, “omission”, “incorrect logic/event” and “timing/order”. Because

the role definition depends on its variation point in the Role Schema, detailed in full in [6], the derived SFMECA captures the possible event and data failures for all the near-identical satellites. Figure 3 gives an example of a partial SFMECA entry for the *MoveNewPos* activity for our TechSAT21 example, an event causing a satellite to alter its position in the constellation.

The procedure to construct a SFMECA table for the events from the Role Schema using the event guidewords consists of the following steps:

For each role:

For each activity listed in the *Protocols and Activities* section of the Role Schema:

- a. Provide the event name in the appropriate column.
- b. Apply each of the keywords (“halt/abnormal termination”, “omission”, “incorrect logic/event” and “timing/order”) to the event. For each keyword:

Role	Event	Event Failure Mode	Local Effect	System Effect	Criticality
Cluster Allocation Planner	MoveNewPos	Halt/Abnormal Termination	The <i>position</i> , <i>velocityIncrement</i> and <i>newPositionList</i> data may be temporarily incorrect since the satellite did not complete moving to its new position. This could potentially affect other events such as <i>UpdateClusterInformation</i> and <i>CalculateDeltaV</i> .	The satellite will not have moved to the position expected by other satellites in the cluster potentially causing a collision.	Major
		Omission	The satellite fails to move to its new assigned position in the cluster possible causing the <i>position</i> , <i>velocityIncrement</i> and <i>newPositionList</i> data to be temporarily incorrect. This could potentially affect other events such as <i>UpdateClusterInformation</i> and <i>CalculateDeltaV</i> .	The satellite will not have moved but, rather maintain its previous position. Other satellites in the cluster may expect the satellite to have moved to a new position. This could cause a collision because of the discrepancies between actual and satellite position.	Major
		Timing/Order	The satellite fails to move to the new position until some later, undetermined time possibly causing its <i>position</i> , <i>velocityIncrement</i> and <i>newPositionList</i> data to may be temporarily incorrect. This could potentially affect other events such as <i>UpdateClusterInformation</i> and <i>CalculateDeltaV</i> .	The satellite fails to move to its new position at the time expected by other satellites in the cluster. This could cause a collision.	Critical

Figure 3. An excerpt of the SFMECA of the MoveNewPos activity of the Cluster Allocation Planner role in the TechSAT21 satellite constellation.

- i. Provide the event failure mode (i.e., the keyword used to discover possible failures).
 - ii. Describe the possible local effect(s) if the keyword failure happened to the event under consideration. The local event will likely only affect this role or this agent and its description should not include the propagation of its failure to other agents or components of the global system.
 - iii. Describe the possible system-level effect(s) if the keyword failure mode occurred. This column captures the possible emergent hazardous behavior from the interaction of the agents (e.g., that a collision could occur between satellites if a satellite does not change its position when other satellites are expecting it to).
 - iv. Give the criticality (e.g., critical, major, average, etc.) of this failure as determined by the global effect of this failure on the system as a whole.
- c. Apply any additional failure modes not captured by the provided keywords relevant to the current event and fill in the SFMECA row as appropriate.

Constructing the SFMECA data table using Gaia's Role Schema, the *Permissions* section lists each datum that the role can access, alter or generate. To construct a SFMECA table for the data that a role uses, we use the following keywords to guide our analysis: "incorrect value", "absent value", "wrong timing" and "duplicated value". The procedure to construct a SFMECA table for the data from the *Permissions* section of the Role Schema is similar to that for the events' table and is not shown here.

3.2 Backward Search Safety Analysis

The backward analysis search technique used in this work utilizes a Software Fault Tree Analysis (SFTA). The SFTA can be performed in parallel and independently of the forward analysis technique(s). SFTA starts with a system hazard and traces backwards to find the contributing causes of the hypothesized root hazard. Typically, the hypothesized hazard comes from existing hazards lists or known domain hazards. If, however, the SFTA is performed following the forward analysis technique(s), each unique *Possible Hazard* listed in the Software Failure Modes, Effects and Criticality Analysis (SFMECA) can be used as a SFTA starting hazard. For example, it is clear from the SFMECA

for the TechSAT21 that a fault tree in the SFTA should include the hazard “satellite collision” as a root node. This is a concept of the Bi-Directional Safety Analysis (BDSA) that will be discussed further in Section 3.3.

3.3 Bi-Directional Safety Analysis Results

Bi-Directional Safety Analysis (BDSA) is used to verify the completeness of the forward and backward search techniques. The forward and backward techniques can be viewed as complementary since the output of the forward technique (i.e., the potential system-wide hazards) should match-up with the inputs of the backward technique. Similarly, the output of the backward technique (i.e., the low-level, local errors that cause a system-wide hazard) should match-up with the inputs of the forward technique. For example, we can verify the completeness of the SFTA by ensuring that every unique hazard listed in the *Possible Hazard* column of the SFMECA table with a particular level of criticality or higher (e.g., major criticality) is a root node within one of the fault trees of the SFTA.

In our TechSAT21 example, BDSA was applied to ensure that all possible hazards labeled with a “major” or “critical” criticality level for the *MoveNewPos* event were captured as the root node of a fault tree. It was found that each “major” and “critical” level potential hazard in the SFMECA related to a collision of satellites and that the SFTA had already accounted for this hazard. However, comparing the event failures in the SFMECA that could possibly lead to a satellite collision with the leaf nodes of the fault trees where “satellite collision” is the root node led to the discovery that the SFTA failed to account for the possibility that a “timing/order” failure in the execution of the *MoveNewPos* event could be a contributing factor to a satellite collision. This omission in the SFTA is likely due to SFTA’s weakness in representing temporal/order-specific failures [15]. Thus, the BDSA not only helped in verifying that the results of the forward technique were the inputs for the backward technique and vice versa, but it also helped identify missing failure scenarios.

3.4 Applying the Safety Analysis Results to Assure Safety

Bi-Directional Safety Analysis (BDSA) helped ensure that the safety analyses used for the forward and backward search techniques were consistent. In our case, the Software Failure Modes, Effects and Criticality Analysis (SFMECA) and the Software Fault Tree Analysis (SFTA) were utilized to discover further safety requirements not already specified in the Role Schema for a given role.

To assess and derive safety requirements of the Role Schema using the SFMECA, the following steps suffice:

For each Role Schema:

- a. For each data/event listed in the *Data/Event* column of the SFMECA for the role in the Role Schema:
 - i. Decide at which level of criticality (i.e., critical, major, etc.) the role must provide mitigating requirements to ensure safety. This may correspond to what level of system certification is required of the system.
 - ii. For each listed data/event failure mode listed in the *Failure Mode* column of the SFMECA with a

criticality of at least the minimum criticality level needed for analysis (from Step i):

1. Consult the local effect of the failure mode in the *Local Effect* column of the SFMECA. Assure that the software mitigates the local effect. For data, the mitigating requirement could be some sanity check (i.e., checking some other piece of data or monitoring that the data is reasonable given the software’s current state). For events, the mitigation requirement could be some guard to ensure that the event is occurring under the right conditions and at the appropriate time given the software’s current condition.
2. Check to make sure that the product-line MAS software will prevent the hazard described in the *Possible Hazard* column of the SFMECA from occurring in the SFTA. That is, check that the hazard is mitigated in **both** the SFMECA and SFTA.
3. If the mitigation does not suffice to prevent the local effect, the software may not be compliant with system safety requirements.

Applying this process on the TechSAT21 example using the partial SFMECA table, shown in Figure 3, identified several new mitigation requirements to prevent the hazard of a “satellite collision” that were then added to the Role Schema. For the “halt/abnormal termination” failure mode, the mitigation requirement was that the *MoveNewPos* activity be atomic (either it executes completely or not at all). Alternatively, a new *NotifyFinishMoveNewPos* protocol could be introduced to have the satellite notify all satellites (or the master satellite) of the completion (or non-completion) of the *MoveNewPos* activity. Additionally, a mitigation requirement for the “timing/order” failure mode could be to assign a timestamp deadline by which each *MoveNewPos* activity must complete before. Without the BDSA and SFMECA process detailed above, safety requirements such as these could be overlooked.

The use of BDSA thus assists in certification of product-line MAS in two ways:

- *Demonstration of compliance.* The use of BDSA provides assurances that certain classes of failure modes that might occur in the individual agents will not produce unacceptable effects in the composite system (e.g., the constellation, or fleet). The artifacts produced in this investigation (SFMECA tables, SFTAs, and the Role Schemas responsibility statements) help demonstrate compliance of the failure-monitoring and failure-mitigation software tasked with the system safety requirements.
- *Enabling reuse of certification arguments.* The use of product BDSA can reduce the burden of certification for systems composed of identical or near-identical units. In systems where each agent is a member of a product line, the similarities can be leveraged for efficient reuse of the safety analysis artifacts. At the same time, the use of Role Schemas captures any variations among the roles that the agents may assume. The Role Schemas thus help ensure

that the reuse of the artifacts in the certification arguments accurately reflects any differences among the agents.

4. CONCLUDING REMARKS

This paper described an extension of Bi-Directional Safety Analysis (BDSA) to aid in the certification of safety-critical, product-line, multi-agent systems (MAS). This extension demonstrated a systematic process to derive a Software Failure Modes, Effects and Criticality Analysis from a product-line MAS analysis and design methodology. The safety analysis assets derived using this approach are reusable assets since they capture the behavior of the near-identical, product line members of the MAS. Using this approach, we showed how missing safety requirements can be derived from the safety analysis assets and how BDSA can assist in verifying the adequacy of the existing safety requirements and design. The resulting product-line safety assets and verification aid in efficiently assembling a safety case during system certification. Planned future work includes further investigation and evaluation of this approach on a large, product-line MAS under development.

5. ACKNOWLEDGMENTS

The authors would like to thank Qian Feng for her comments and useful discussions. This research was supported by the National Science Foundation under grants 0204139, 0205588 and 0541163.

6. REFERENCES

- [1] Arkusinski, A., "A Method to Increase the Design Assurance Level of Software by Means of FMEA," *Proc. 24th Digital Avionics Systems Conference*, 2:10.D.5-1-10.D.5-11, 2005.
- [2] Bresciani, P., Giorgini, P., Guinchiglia, F., and Perini, A., "TROPOS: An Agent-Oriented Software Development Methodology," *Journal of Autonomous Agents and Multi-Agent Systems*, 8(1):203-236, 2004.
- [3] Chien, S. et al., "The Techsat-21 Autonomous Space Science Agent", *Proc. 1st International Conference on Autonomous Agents*, pp. 570-577, 2002.
- [4] Chien, S., Sherwood, R., Tran, D., et. al., "Lessons Learned from Autonomous Sciencecraft Experiment," *Proc. Autonomous Agents and Multi-Agent Systems Conference*, 2005.
- [5] Clements, P., *Software Product Lines*, Addison-Wesley, Reading, MA, 2002.
- [6] Dehlinger, J. and Lutz, R. R., "A Product-Line Approach to Promote Asset Reuse in Multi-Agent Systems," *Software for Multi-Agent Systems IV*, Lecture Notes in Computer Science, to appear, 2006.
- [7] Dehlinger, J. and Lutz, R. R., "A Product-Line Requirements Approach to Safe Reuse in Multi-Agent Systems," *Proc. ICSE 2005 Workshop on Software Engineering for Large Multi-Agent Systems*, pp. 83-89, 2005.
- [8] Dehlinger, J. and Lutz, R. R., "PLFaultCAT: A Product-Line Software Fault Tree Analysis Tool," *The Automated Software Engineering Journal*, 13(1):169-193, 2006.
- [9] DeLoach, S. A., "The MaSE Methodology," *Methodologies and Software Engineering for Agent Systems – The Agent-Oriented Software Engineering Handbook Series: Multi-Agent Systems, Artificial Societies, and Simulated Organizations*, 11:107-125, 2004.
- [10] Despotou, G. and Kelly, T., "Extending the Safety Case Concept to Address Dependability," *Proc. 22nd Int'l System Safety Conference*, pp.645-654, 2004.
- [11] European Cooperation for Space Standardization (ECSS), *Space Engineering – Software, ECSS-E-40B (draft 1)*, 2002.
- [12] Feng, Q. and Lutz, R. R., "Bi-Directional Safety Analysis of Product Lines," *Journal of Systems and Software*, 78(2):111-127, 2005.
- [13] Hollow, P. McDermid, J. and Nicholson, M., "Approaches to Certification of Reconfigurable IMA Systems," *Proc. 5th Int'l Symposium of the Int'l Council on Systems Engineering*, 2000.
- [14] Lamport, L. and Lynch, N., "Distributed Computing Models and Methods," *Formal Models and Semantics, Vol. B, Handbook of Theoretical Computer Science*, Elsevier, 1990.
- [15] Leveson, N. G., *Safeware: System Safety and Computers*, Addison-Wesley, Reading, MA, 1995.
- [16] Lutz, R. R. and Woodhouse, R. M., "Bi-Directional Analysis for Certification of Safety Critical Software," *Proc. 1st International Software Assurance Certification Conference*, 1999.
- [17] Lutz, R. R. and Woodhouse, R. M., "Requirements Analysis Using Forward and Backward Search," *Annals of Software Engineering*, 3(1):459-475, 1997.
- [18] "NASA Strategic Roadmaps," *NASA – APIO: Strategic Roadmaps - Overview*, http://www.nasa.gov/about/strategic_roadmaps.html, (current February 2006).
- [19] Radio Technical Commission for Aeronautics, *RTCA/DO-178B: Software Considerations in Airborne Systems and Equipment Certification*, 1992.
- [20] SAE, *Aerospace Recommended Practice: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, ARP4761, 1996.
- [21] Schetter, T., Campbell, M. and Surka, D., "Multiple Agent-Based Autonomy for Satellite Constellations," *Proc. 2nd International Symposium on Agent Systems and Applications*, 2000.
- [22] "Terrestrial Planet Finder Project," *Planet Quest: Missions – Terrestrial Planet Finder*, http://planetquest.jpl.nasa.gov/TPF/tpf_index.html, (current February 2006).
- [23] Weiss, D. M. and Lai, C. T. R. *Software Product-Line Engineering*, Addison-Wesley, Reading, MA, 1999.
- [24] Zambonelli, F., Jennings, N. R. and Woolridge, M., "Developing Multiagent Systems: The Gaia Methodology", *ACM Transactions on Software Engineering and Methodology*, 12(3):317-370, 2003.